Foresight: Countering malware through cooperative forensics sharing

by

Muhammad Fareed Zaffar

Department of Computer Science
Duke University

Date: __________________________

Approved:

Gershon Kedem, Advisor

Jeff Chase

Jun Yang

Miguel S.Lobo

Wilfred Amaldoss

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University 2008
ABSTRACT
(Computer Science)

FORESIGHT: COUNTERING MALWARE THROUGH COOPERATIVE FORENSICS SHARING

by

Muhammad Fareed Zaffar

Department of Computer Science
Duke University

Date: _______________________

Approved:

__________________________
Gershon Kedem, Advisor

__________________________
Jeff Chase

__________________________
Jun Yang

__________________________
Miguel S.Lobo

__________________________
Wilfred Amaldoss

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University 2008
Abstract

With the Internet’s rapid growth has come a proportional increase in exposure to attacks, misuse and abuse. Modern viruses and worms are causing damage much more quickly than those created in the past. The fast replication and epidemic nature of the spreads limits the time security experts have to respond and be able to protect and fortify their systems. A pathogen might infect thousands of machines and cascade across the network producing consequences that could overwhelm the Internet very quickly. Such attacks have the potential of making a human response to them all but ineffective. While pathogens are becoming much more aggressive, there is also a significant delay between the identification of a new threat and the generation of a cure for it. Worms and viruses have been able to cause significant damage in this ‘submission to cure generation” window of vulnerability. Having timely and credible security information is thus becoming critical to network and security management.

The main hypothesis behind our research is that sharing threat information and forensic evidence among cooperating domains yields important benefits for dealing with modern day pathogens in a timely fashion. The idea is that each host might have an incomplete, approximate or inexact information about a particular threat or attack. We can get a more comprehensive view of the extent and nature of developing threats by observing suspect behavior and combining information gathered from different vantage points. A better understanding of the pathogen allows for effective and timely immunization in order to thwart epidemic cascading of threats. We also propose cooperative policing mechanisms as an effective approach to trace large scale distributed threats like DDoS attacks. Increased cooperation between domains helps
to mitigate such attacks nearer to the sources so that their effects on the overall network are minimized.

This thesis leverages experiences and ideas from fields of cryptography, machine learning, security and multi-agent systems to build Foresight: an internet scale threat analysis, indication, early warning and response architecture. Foresight allows cooperating domains to share a global threat view in order to detect zero-day pathogens and isolate them using cooperative policing mechanisms.

- We describe a novel behavioral signature scheme to extract a generalized footprint for multi-modal threats. Blended or multi-modal threats combine the characteristics of viruses, worms, trojan horses and malicious code to initiate, transmit and spread attacks. By using multiple methods and techniques, blended threats can quickly spread and surpass defenses that address only a single type of malicious activity and hence are much more difficult to defend against. System performance analysis, through trace-based simulations, shows significant benefits for sharing forensics data between cooperating domains.

- We present Mail-trap, an anomaly based system that catches zero-day email borne pathogens and retards their growth through effective behavior monitoring of mail traffic and active forensics sharing between cooperating domains. Mail-trap relies on Foresight’s cooperative policing model to identify and pre-empt email-borne threats. Our results show that behavior monitoring alone can be an effective tool for malware detection. Cooperation between domains greatly increases the effectiveness of our approach. Domains are able to pre-empt attacks and respond to malware behavior that they have not seen before. We also analyze various immunization/prevention and containment techniques.

- We present AMP, a service architecture for countering distributed denial of service attacks using alert sharing and cooperative policing mechanisms. Our simulation architecture enables us to test the system with actual, benign and worm traffic
traces, and realistic network topologies. AMP does not require universal deployment and is complementary to other schemes for countering DDoS attacks, however with the use of collaborative policing techniques, the performance of the scheme can be improved greatly.

- We also present a prototype implementation for Paranoid, a novel global secure file sharing mechanism which can be used to allow secure resource access across administrative domains. We describe the design of a trust-based cooperation scheme to create a global community which is more accountable and hence less vulnerable to attacks and abuse.
Contents

Abstract iv
List of Tables xii
List of Figures xiii
Acknowledgements xvi

1 Introduction 1
1.1 Background ......................................................... 1
1.2 Malware Definitions ................................................ 3
1.3 An analysis of existing industry standards ......................... 6
  1.3.1 Anti-virus approaches .......................................... 7
  1.3.2 Firewalls ....................................................... 8
  1.3.3 Intrusion Detection Systems .................................... 8
  1.3.4 Patching ....................................................... 10
1.4 Characteristics of advanced worms ................................. 11
  1.4.1 Example: Nimda ................................................. 11
  1.4.2 Example: Phatbot/Storm ....................................... 13
1.5 Discussion .......................................................... 14
  1.5.1 Need for a global View ........................................ 14
  1.5.2 Need for automation .......................................... 17
  1.5.3 Need to address the ‘Submission to cure generation window’ . 18
  1.5.4 The need for forensic data .................................... 18
1.6 Cooperative forensics sharing and policing ....................... 19
1.7 Thesis and Contribution ............................................ 21
2 System Goals and Design

2.1 Foresight system design and architecture .................................. 30
  2.1.1 Sensors/Enforcers ...................................................... 33
  2.1.2 Enforcers ................................................................. 35
  2.1.3 Collaboration Agent ..................................................... 36
  2.1.4 Knowledge Manager ..................................................... 39

3 Countering Email based Malware .............................................. 44

  3.1 Email-borne malware .............................................................. 45
  3.2 Email, SMTP and envelope header ........................................ 46
    3.2.1 Establishing Normal Email patterns .................................. 49
    3.2.2 Worm behavior testbed .................................................... 50
  3.3 Mail-trap Architecture ......................................................... 54
  3.4 Enforcers ............................................................................. 59
  3.5 Knowledge Manager ............................................................... 60
    3.5.1 Accuracy of the scoring technique ..................................... 67
    3.5.2 Signature ...................................................................... 68
    3.5.3 Cooperation between trusted peers ..................................... 71
    3.5.4 Traffic Policing ............................................................... 73
    3.5.5 Mail-trap Traffic Marking .................................................. 74
  3.6 Observing misbehavior remotely ............................................. 76
    3.6.1 Reputation .................................................................... 78
  3.7 Implementation, Simulation, Experiments and results ................. 79
    3.7.1 Local Misbehavior Detection ............................................. 80
    3.7.2 Simulation as a verification technique .................................. 84
    3.7.3 Simulator Experiments ..................................................... 85
List of Tables

3.1 Sample Email Envelope header .......................... 48
3.2 Malicious Worm behavior on windows platform ........ 52
3.3 Mydoom Source Code snippet ............................ 54
3.4 Mail-trap operation .................................... 62
3.5 MyDoom Subject selection, rot13 ........................ 65
3.6 Malicious Worm behavior on windows platform ........ 83
3.7 Email Marking based on last-hop server .................. 103
4.1 TCP traffic statistics:CAIDA ............................ 124
4.2 Detection Scheme ..................................... 148
4.3 Detection Scheme ..................................... 149
6.1 Latency observed (in seconds) to open a Paranoid file on a local host. 183
6.2 Latency observed (in seconds) to open a Paranoid file on a remote host. 183
List of Figures

1.1 The Evolution of the security threat over the years (Source: Symantec) 2
1.2 The number of probes issued per hour: Slammer worm (Source: Inside the slammer worm) .................................................. 3
1.3 Nimda propagation vectors (Fsecure) ................................. 12
1.4 Lack of a Global View ..................................................... 15
2.1 High Level Foresight Architecture .................................... 30
2.2 Data Sharing Matrices .................................................. 38
2.3 Conceptual Design ....................................................... 40
2.4 Insides of KM ............................................................ 41
3.1 Workings of a very simple email system .............................. 47
3.2 Average daily Emails sent per user per hour ......................... 50
3.3 Average daily Emails sent per user per hour (Nocturnal Habits) 51
3.4 Average Weekly Emails sent per user per hour ..................... 52
3.5 Mailtrap Sensor Architecture .......................................... 55
3.6 ILoveYou Virus .......................................................... 70
3.7 Server-prof overhead .................................................... 70
3.8 Log-normal probability distribution, Number of emails sent per hour 81
3.9 Epidemic spreading of the worm at various propagation rates .... 87
3.10 Epidemic spreading of the worm (Realistic mailbox behavior) .. 88
3.11 Epidemic spreading of the worm with attachment probability (propagation rate 1)) ......................................................... 89
3.12 Effect of Alert Sharing on worm propagation (propagation rate 2) . 90
3.13 Effect of Alert Sharing on worm propagation (propagation rate 4), different alert sharing policies ................................. 91
3.14 Different Alert Sharing policies ........................................ 92
3.15 Number of attack emails reaching the server ....................... 93
3.16 Effect of degree of cooperation for different worm propagation rates 94
3.17 Effect of varying signature accuracy on the spread of the worm when using Mail-trap ..................................................... 95
3.18 Effect of data sharing for polymorphic worms for varying degrees of cooperation .............................................................. 96
4.1 Architecture : Detector placement options ............................ 111
4.2 Distributed denial of service attack ...................................... 114
4.3 Parital Filters enforced after alert disseminated ..................... 118
4.4 Server Traffic patterns (Peak hr), Normal Distribution ............ 121
4.5 Server Traffic patterns (Peak hr), Lognormal Distribution ........ 122
4.6 Server Traffic patterns, off peak hour ................................. 123
4.7 Threshold based attack detection ....................................... 125
4.8 Client statistics .............................................................. 127
4.9 Client statistics .............................................................. 128
4.10 Server Performance CPU Utilization ................................ 129
4.11 Server Performance(Page Faults) ...................................... 130
4.12 Server Performance Interrupts ......................................... 130
4.13 Cumulative distribution of number of packets and domains at an egress router ................................................................. 135
4.14 Policing of misbehaving traffic by the KM ........................... 138
4.15 DoS Simulation Topology .................................................. 143
4.16 Total traffic at server during attack .................................... 144
4.17 Total attack traffic .......................................................... 150
4.18 Attack Type 1 ............................................................... 151
4.19 Attack Type 3 ............................................................... 152
Acknowledgements

This effort is dedicated to my wonderful parents Afzal and Khalida, my lovely wife Mahira and my siblings, Nausheen, Naveed and Adeel. It is impossible to thank them for the role that they have played in my life. They were always there when I needed them. Indeed I would not have been able to get from start to finish without their love, support and prayers.

I would like to thank my advisor Gershon Kedem, Wilfred Amaldoss and Miguel Lobo for all the support I received over the years. I would specially like to thank Jeff Chase for all the support and guidance over the years. He went above and beyond the call of duty to make me comfortable at Duke and to help me focus enough to finish this thesis.

I have been fortunate enough to have a great friend in Umair Shah. Shah sahab has been a friend, a mentor, a sounding board and a constant source of advice. His ability to handle ten different ventures at the same time is truly inspirational. Finally, I would like to thank Balaji Krishnapuram. My Duke experience would not have been the same without him.
1

Introduction

1.1 Background

Modern viruses and worms are causing damage much more quickly than those created in the past [70]. Viruses and worms are able to spread with great speed and are capable of harnessing the size and scale of the Internet to launch high-powered distributed attacks. Due to the fast replication speeds and the epidemic nature of the attacks security experts and systems administrators have very little time to respond and be able to protect and fortify their systems. Such attacks have the potential of making a human response to it all but ineffective. The Slammer worm for instance infected more than 90% of the vulnerable hosts within 10 minutes [59] [57] causing significant disruptions of service and losses throughout the Internet.

The number of security incidents has gone from approximately 10000 in 1999 to 137,529 in 2003 [12]. Although CERT no longer maintains this statistic, there is no sign of things slowing down. To compound the problem further, malware is becoming increasingly potent and malicious, therefore, the damage caused by such incidents is likely to increase manyfold. The pathogens that were observed in the past caused a
lot of damage even though they were designed to do little more than just propagate themselves. Slammer for instance carried no malicious payload and caused harm by saturating networks and disabling Microsoft’s database servers. We are already seeing malware carrying increasingly well engineered and malicious payloads.

While the threats have evolved significantly over the years (see figure 1.1), the Internet’s rapid growth has led to a disproportional increase in exposure to attacks, misuse and abuse. Computer systems have become increasingly interconnected and bandwidth prices have fallen significantly. A new pathogen might infect thousands of machines and cascade across the network, producing consequences that could overwhelm the Internet very quickly.

According to one estimate, the Internet now connects over 180 million computers and is being used to provide services that people are increasingly dependent upon. In the past, system administrators around the globe were able to react quickly to a threat, ensuring that the damage was minimized. However the scale of the problem has changed dramatically and such mechanisms have become inadequate. Many attacks are now fully automated and spread with blinding speeds (see figure 1.2), capable of harnessing the size and scale of the Internet to launch high-powered dis-
tributed attacks. In such a scenario, it is obvious today that human-centered reactive solutions alone are not adequate to address the problem.

Modern viruses and worms have been able to create much havoc even before they are identified as threats. There is also a significant delay between the identification and isolation of a new threat and the generation of a cure for it. Worms and viruses have been able to cause significant damage in this ‘submission to cure generation’ window of vulnerability.

1.2 Malware Definitions

In this section we outline the definitions of the different types of malware referred to in the thesis. We would, however, point out that the line between viruses, worms and various other pathogens is is no longer sharp. Formally defined, a Virus refers to a program code which has the capability to replicate by itself. A computer virus needs a host program to attach itself to and replicates along with the host program. A Virus typically depends on some form of human intervention to propagate, whether this is opening an email attachment, clicking a malicious link, or transferring an infected disk from one machine to another. An Email virus is a special type of
virus which is sent as an email attachment. It replicates by automatically mailing itself to a large number of email addresses usually derived from the user’s address book and local files. Melissa was the first mass email worm. It emailed itself to the first 50 entries in a user’s address book. More recently, Klez used its own SMTP engine and a Microsoft Outlook vulnerability to automatically execute itself and to spread. Spam is closely related to email viruses. Spam is typically harmless but always annoying, electronic equivalent of junk mail. Spam has become such a huge problem that in a single day in May 2003, AOL detected and blocked 2 billion spam messages. Microsoft blocks an average of 2.4 billion spam messages per day.

However there has been a changing trend away from activity-triggered propagation which has increased the potency of the malware[60]. For lack of a better definition, Worms are a special type of virus that do not depend on any form of human intervention or host program to propagate. Both Viruses and worms exploit weaknesses in computer software, replicating and attaching themselves to other programs. The ability to take over the communication components (email, peer to peer file sharing in windows for example) of the host system allows a worm to spread thousands of times faster than a traditional virus. Since worms can replicate and infect by themselves, they are by far the most potent type of malware, and can infect many millions of computers globally in a matter of hours. Worms sometimes contain a destructive payload but successful ones almost always create a denial of service effect that can impact productivity. Network aware worms typically cause heavy traffic that saturates networks preventing access to critical resources. We refer to the propagation methods of a virus or worm as the propagation vectors. A vector is the agent that carries and transmits the pathogen. Payload refers to the malicious code which is carried by the attack vectors. Typical payloads take the form of spyware, trojan horses, backdoors and other malicious components. Not all worms that were observed in the past had payloads, some simply propagated themselves.
A trojan horse which is usually installed as part of a worm payload, refers to a class of malicious code that performs the task intended by the user along with another task that the user is unaware of. This unintended task usually leads to destructive after-effects.

Denial of service attacks, hereafter referred to simply as “DoS” attacks, can severely limit the ability of an organization to conduct normal business on the Internet, leading to economic losses. In a denial of service, *DoS*, attack, a server is rendered unable to respond to valid requests due to a large amount of malicious traffic. Clients trying to access the service are unable to get through to the server. Most denial of service attacks exploit the lack of authentication in the IP protocol. Denial of service attacks can be either bandwidth-saturating or resource saturating. Bandwidth saturating attacks are made possible by the attacker having access to a higher bandwidth than the victims; one way of achieving this is through a distributed denial of service attack. Resource saturating attacks aim to utilize finite resources like number of open connections and disk space etc. by sending specific types of packets. In a distributed version of the same attack, a *DDoS* attack, the flood of malicious traffic comes from a group of compromised computers. ‘Zombies’, as compromised computers are usually referred to, are operated using remote control by people who have stealthily gained access to them. Such traffic is distributed across different entry points of the attacked network making it harder to locate and shut off. Collections of such *zombies* are sometimes referred to as *botnets*.

The last few years have seen a huge rise in botnet activity. A *botnet* refers to a collection of software robots running on a collection of ‘zombie’ machines. The botnets allow remote control of these ‘zombie’ machines effectively giving massive untraceable distributed computing power to their owners. According to Symantec estimates, the number of active botnet computers had already exceeded four million in 2006 [78]. Botnets exploit their ‘zombie’ machines to launch various attacks
such as distributed denial of service attacks, spam, click fraud, spyware, information harvesting etc.

While the attacks have increased in sophistication and coordination, the last few years have also noticed a growing convergence of threats. A newer generation of multi-pronged pathogens, *Blended* or *Multi-modal Threats*, with extensive *feature lists* are able to use a combination of viruses, worms, bots, phishing, trojan horses and malicious code in order to exploit server and internet vulnerabilities to initiate, transmit and spread an attack [87]. By using multiple methods and techniques, blended threats can quickly spread and surpass defenses that address only a single type of activity. Such threats typically include more than one means of propagation and are able to spread with or without human intervention. Some well-known examples of *Blended* threats have been *CodeRed, Goner, Klez, Bugbear* and *Nimda*. The impact of these newest threats can be gauged by the fact that the estimated economic impact of Code Red was in billions! [94].

*Polymorphic* worms on the other hand, have been known to encrypt and/or mutate their contents in each generation. This makes it increasingly difficult for traditional string-matching techniques to catch such threats. Furthermore, traditional antivirus solutions rely heavily on a post-analysis of virus characteristics to develop a signature that can be used to identify viruses. The speeds of propagation that recent threats have achieved make this type of reactive solution ineffective.

1.3 An analysis of existing industry standards

The security community actively seeks to identify and control the spread of viruses and worms. However, given the dynamic and evolving nature of the problem, we feel that they are always one step behind. Most of the newer anti-virus approaches are a result of hindsight gained from successful attacks.
1.3.1 Anti-virus approaches

Current Anti-virus products can be largely divided into three categories, Virus scanners, integrity checkers and behavior blockers. **Virus scanners** try to find known viruses using static signature matching techniques. The disadvantage of this technique is that signatures need to be frequently updated and detection is only limited to known threats. Users are protected only against all threats that predate their last signature update. **Heuristic scanners** on the other hand try to find virus specific behavior or epidemiological intelligence by recognizing instructions typical of viruses. The disadvantage of this approach is that heuristic scanners have a high false alarm rate and rely on the user to correctly interpret the results of heuristic scanning. **Integrity checkers** rely on the fact that a virus will cause changes to a system while infecting it. Integrity of specific areas can be verified and unknown viruses detected without frequent updating of the program. Checksum calculations are the most reliable integrity checkers. However this technique can be applied only to areas which are not normally modified regularly. **Behavior blockers** warn about suspicious behavior of programs but a major draw back is that users should be able to correctly interpret the information provided by a behavior blocker and reject false positives. Behavior blocking, like intrusion detection software, has a horrible reputation for false alarms. **Memory resident programs** running in the background can perform either integrity checking, or scanning in order to detect virus behavior.

Network-aware worms pose a unique threat to anti-virus approaches since they do not necessarily have a file component that can be identified by traditional host and perimeter software. Furthermore, traditional antivirus solutions rely heavily on a post-analysis of virus characteristics to develop a signature that can be used to identify viruses. The speeds of propagation that recent threats have achieved
make this type of reactive solution ineffective. As noted earlier, there is a significant ‘ submission to cure generation’ window of vulnerability which can be exploited.

1.3.2 Firewalls

Firewalls provide an easy mechanism for enforcing network security policy. Security has been historically provided by creating a perimeter to isolate an organization from the outside world, overlooking the possibility of an internal threat. Typical firewalls provide a policy-based access control to restrict users and applications to specific segments of a topology, ports, and protocols. This approach provides a soft-target for malware and threats that can breach the external perimeter. Also, large networks tend to have a large number of intentional or unintentional entry points. Backdoors, Trojans, Wireless access, tunnelling and dial-up access make it relatively simple to establish unauthorized entry points into the system. Firewalls alone would have little or no chance to catch any threat initiated through such channels. An infected machine would look like a trusted computer using a legal protocol/port combination. By the time the threat is discovered and policy changes manually implemented at firewalls, a worm may have already propagated to other domains. In view of such weaknesses, firewalls have acted like more like drawbridges during recent worm outbreaks after the damage had already been done. Advanced firewalls offer application awareness and a finer-grained access control. These firewalls can potentially check for a limited range of application-layer attacks at the expense of extra processing time.

1.3.3 Intrusion Detection Systems

An intrusion detection system consists of several components. Sensors record events in the system or traffic on a network link. The resulting log-files are fed into an analysis engine which either utilizes a signature database or misuse rules to decide whether
an intrusion has occurred. Intrusion detection systems get their bad reputation due to the high number of false positives they traditionally generate.

Intrusion detection systems (IDSs) are designed to monitor and log abnormal network behavior, but do not necessarily provide the actual protection to keep critical resources secure. Intrusion detection uses either anomaly detection or misuse detection. Anomaly detection based IDSs model normal behavior and any deviation is considered an attack. Misuse detection based systems model behavior that clearly indicates an attack. Intrusion detection systems can be further characterized as host-based, multi-host based or network based. In host-based systems, data from a single host is scrutinized by monitoring audit log files and other file and operating system behavior. In network based systems, the network traffic and data from hosts attached to the system are analyzed. In the case of network based systems, a module typically runs on one host that monitors the network. Intrusion detection systems can either be real-time or off-line. A real time system monitors the system continuously and reports intrusions as soon as they are detected. An off-line system does a post-mortem analysis at set intervals and records the suspicious activity that takes place. Typical intrusion detection architectures can be centralized, distributed or hierarchical. In hierarchical systems, some of the data is passed up through various layers and is analyzed at each level of the hierarchy. Distributed Intrusion Detection systems spread the detection across all the hosts being monitored. Distributed intrusion detection systems build on an agent based architecture to analyze multiple systems as a whole.

Intrusion detection systems use information derived from audit trials, log files or packets from the network. One weakness of this scheme is that data might be destroyed or modified by an attacker or threat. The second problem with this approach is that the an IDs has to infer the intent of an attacker or a threat from the data. A third problem with such approaches is that an Intrusion detection system would
continue to use system resources even when there is no threat. In short, misuse-based ID schemes do not give us sufficient forewarning about a threat and anomaly based schemes do not give us enough contextual information. The granularity of an anomaly scheme can be increased to provide better context but that would also increase the overhead incurred by the scheme. If data is acquired with a significant delay, detection of a threat could be too late to be useful.

A logical transition from IDSs has been to intrusion prevention systems (IPSs) which are more adept at stopping ongoing threats. IPS’s are installed on servers and desktops as opposed to network segments. Although IPSs can drop threatening traffic and prevent worm proliferation, they suffer from having a limited view of the global threat scenario.

1.3.4 Patching

Most worms attack known vulnerabilities for which a patch is long available. Code Red exploited a well known buffer overflow vulnerability in the MS Index Server. The Sapphire/Slammer worm also launched an attack against a well-known Microsoft SQL Server vulnerability that had been discovered more than six months earlier [59]. Lately however, the period of time between identification of vulnerability and development of an exploit has diminished dramatically. Further problems are created by the fact that new and serious vulnerabilities keep cropping up with surprising regularity and patching is still effectively a manual process where IT departments have to prioritize due to the sheer volume of the alerts. The need to test out patches off-line is a similar issue; organizations are reluctant to apply patches to production system for fear of breaking something in a working system.
1.4 Characteristics of advanced worms

With the advancement in attack technology and increasing connectivity between hosts, next generation threats and worms are likely to saturate networks much quicker than before. A new threat would be able to infect thousands of machines across the network producing consequences that could overwhelm the network quickly and cause severe damage to critical applications. As discussed in the previous section, the last few years have seen a growing convergence of threats along with an increased sophistication. Worms, bots, spam, phishing, viruses are all becoming increasingly coordinated and interconnected. Some of the viruses and worms that were observed in the past caused a lot of damage but most were designed to do little more than propagate themselves. The Slammer worm for instance infected more than 90% of the vulnerable hosts within 10 minutes [59] [57] causing significant disruptions of service and losses throughout the Internet. However, slammer carried no malicious payload and caused harm by saturating networks and disabling Microsoft’s database servers. In the future, it is likely that we will see malware carrying increasingly well engineered and malicious payloads. To make the above mentioned characteristics of advanced worms more concrete, we describe two existing malware that exhibit sophisticated characteristics as well as advanced coordination.

1.4.1 Example: Nimda

Nimda [64] is an excellent example of a multi-modal worm which was more effective than Code Red or the Morris worm primarily because it had five different infection vectors. It was first discovered in September 2001 and used SMTP, HTTP, MIME, TFTP and TCP/IP to propagate itself. As shown in figure 1.3, Nimda has a file infection component, a mass mailing vector, a web worm vector as well as a LAN propagation component. The worm also behaves differently on different machines.
Figure 1.3: Nimda propagation vectors (Fsecure)

depending on how it was initialized. Nimda takes advantage of multiple vulnerabilities to propagate and has the potential to infect both user workstations as well as servers. The primary propagation vector is email where the worm spreads as an email attachment (readme.exe) that contains worm code. Nimda traverses all directories attaching itself to all web content allowing further propagation through web browsers as well as browsing of a network file system. The worm also creates trojan horse versions of existing applications that first execute Nimda code and then complete their intended function. Finally, the worm scans for vulnerable IIS servers scanning for and exploiting various Microsoft IIS (4.0/5.0) directory traversal vulnerabilities and looking for backdoors left by previous worms such as Code Red [13]. It then copies worm code to these machines using the tftp protocol.
1.4.2 Example: Phatbot/Storm

Our next example, phatbot, belongs to the Agobot family of bots with additions to make it more flexible and dangerous. There are a huge number of variants of this particular botnet due to the public availability of its source code. Phatbot has a polymorphic engine that enables it to evade anti-virus signatures as it spreads and has about eight different propagation vectors. It relies on P2P networking technology which makes variants much harder to locate and clean. Phatbot contains a network sniffer, a keystroke logger and a rootkit installer along with various other exploits. A combination of these can be used for harvesting product keys, login/passwords, and emails for spam purposes, launch distributed denial of service attacks, click fraud as well as sending a deluge of virtually untraceable spam.

Phatbot uses a hard-coded list to disable various processes including anti-virus processes and firewalls as well as competing trojans and viruses. It installs a variety of exploits such as application backdoors that allow for remote access or hidden access and is able to run socks, HTTP or FTP server proxies on demand. It utilizes these installed processes or operating system vulnerabilities and backdoors installed by previous worms to propagate itself. To add to the problem, the modular design of the code allows any number of exploits to be added as infection vectors.

Storm is one of the most prominent botnets with estimates of size ranging from one million to fifty million ‘zombie’ machines! Most researchers agree that storm is written for profit therefore the worm spreads stealthily, without making much noise. Storm has an advanced peer-to-peer delegation of duties with no centralized control which enables it to maintain operation even in the face of a compromise. Storm has a polymorphic payload that changes every 30 minutes making typical signature based detection techniques ineffective. Storm is spread as an email worm attachment. Once the attachment is activated, it installs trojans and rootkits on the compromised
machine. Storm also has the novel idea of attacking its pursuers with an automatic denial of service attack. Storm even has a very interesting self preservation module that launches a DDoS attack against sites downloading too many copies of the bot.

1.5 Discussion

The above examples raise a few important points about newer threats that must be addressed in an integrated solution.

1.5.1 Need for a global View

Having timely and credible security information is becoming critical to network and security management. Unfortunately most current sources of threat information and detection techniques suffer from this limitation. The multi-modal nature of newer threats highlights this problem clearly. Existing security systems rely on local information to detect threats and apply security policies. An analysis of Nimda suggests that an intrusion detection or prevention system that only acts on local information is clearly handicapped in handling a multi-modal distributed threat. In the case of a large-scale distributed threat, the behavior observed at any one domain might not be large enough to raise an alarm. Or worse, smarter worms show varying characteristics on different systems, in order to exploit different weaknesses. That way, a single domain would not be able to observe the complete propagation vectors of a ‘Blended or Multi-modal Threat’. Slammer had a one dimensional propagation vector and was easily distinguished through its network signature but with its multiple propagation vectors, Nimda was particularly difficult to track. Complete information about the worm was not available initially and it continued to spread through its various propagation vectors on different domains when it was thought that it was only an email worm. It was only later through a complete post-mortem analysis that a complete signature was eventually developed.
A malware detection or prevention system that only acts on local information is clearly handicapped in handling a multi-modal distributed threat. Similarly, a malware detection scheme that only targets a particular propagation vector is handicapped. The problem of effective detection and prevention has interesting parallels with the story of the six blind men and the elephant. The story has been mentioned in various cultures with slight variations and provides direct insight into what happens when a global view of an unknown is not taken.

Six blind men try to figure out an unknown, the elephant, based on their observations. Each person has a unique perspective of the animal from the part that he/she can feel. The man that gets to touch the tusk states that an elephant is like a spear. The man that gets hold of the tail argues that the elephant is like a rope. The man feeling the trunk argues that it resembles a huge snake. The one touching the side thinks it is more like a wall. But the man with the leg is sure the elephant resembles a tree. The point of the narrative is that speculating on the whole based on a limited view can lead to very large errors in judgment. If a unified global picture is not presented, one is likely to miss the elephant altogether.

Lack of a global view of the Internet is a huge challenge in developing internet scale threat analysis, indication and early warning capabilities. Forensic evidence gathered from past threats suggests that internet wide effects of a lot of the threats
could have been mitigated had there been an effective collaboration between domains. A case in point is the Slammer worm. Within an hour of the attack, many sites had started filtering all UDP traffic with the destination port of 1434 by updating router and firewall configurations[59]. Had this information been disseminated to all the domains in a timely manner, large scale network clogging and the resulting service disruptions could have been avoided while a cure for the worm was found or the systems patched. An effective quarantine of the infected parts of the Internet could also have been initiated as well. However as we recall, Slammer had infected 90% of the vulnerable hosts within 10 minutes. So while this filtering would have controlled unnecessary bandwidth consumption, it would not have done anything to prevent/stop the virulent spread of the worm.

Another advantage of aggregating information from different sources is that a system can potentially trace large scale distributed threats like distributed denial of service attacks and mitigate them nearer to the sources. A typical scenario can be created to illustrate this. Let us suppose a few machines across the Duke network are being used as zombie machines as part of a coordinated attack against another university’s network. Assuming the traffic generated by the zombie machines at Duke stays below the radar, there would be no way for the Duke system administrators to find out about the misbehavior till they are informed by the victim network. Newer schemes are being increasingly utilized in botnets that actually make this a realistic threat scenario.

Lastly, efficient sharing of data between different trusted vantage points on the Internet would give a clearer view of threats, their signatures and observed propagation vectors. The cooperation model proposed by this thesis is inspired by the way a human brain interprets the different sensory signals. Until recently, scientists believed that information obtained by the five senses in human beings was processed in separate parts of the brain. Newer research confirms that a unified view of the world
is obtained through an interaction of all five senses in the brain [10]. To assist our
sense of smell which is poor, compared to other animals, we often rely on additional
information from our visual system. We augment the security perspective available
at any domain by active sharing of information between domains and show that a
unified view is of greater value than the individual perspectives in countering newer
threats.

The idea is that each host might have an incomplete, approximate or inexact
information about a particular threat or attack. By observing suspect behavior and
combining information gathered from different vantage points, each domain is able
to get a much clearer picture of the threats and take remedial measures. A global
view also enables the overall system to benefit from the experience of other peers
who have experienced similar threats.

1.5.2 Need for automation

The Slammer worm used a random scanning scheme to select its victims which meant
that the worm had reached 55” million scans per second within approximately 3”
minutes [59]. Any human initiated response against such a threat would have been
hopelessly slow. Since Slammer had a one dimensional propagation vector and was
easily distinguished through its network signature, the situation would be even worse
in the face of a ‘blended threat’. This brings us to the need for automated detection
and response to suspicious network behavior.

There is a need for automated processes in place to assess the urgency and an-
alyze the impact of the threats and to take remedial actions in quick time. Human
mediated response although essential at some level is too slow in countering an or-
ganized, malicious threat to the Internet infrastructure. In some cases, by the time
a human could even notice a problem, the suspicious activity could already be over
removing all traces of its existence. To further complicate the issue, threats are becoming smarter. Most older pathogens were too noisy and that ultimately proved to be their downfall. Slammer for instance used a random scanning scheme to select its victims and had reached 55 million scans per second within approximately 3 minutes [59]. However cleverer pathogens like Storm have shown signs of staying dormant to do harm at some later stage. This makes them more difficult to detect. That, in our opinion, is the direction in which network-aware pathogens are headed. Had the slammer worm stopped replicating after 10 minutes, staying dormant to do harm at some later stage, it would have been quite a challenge to detect and cure an infection of this scale.

1.5.3 Need to address the ‘Submission to cure generation window’

It is widely accepted that an anti-virus program or Intrusion Detection/Prevention system that relies only on signature matching is only as good as its last signature update. High speed pathogens have been specifically designed to exploit this ‘submission to cure generation’ window. Newer threats are truly capable of overwhelming the network resources within minutes. Human centered response to such threats is too slow. In some cases, by the time a human could even notice a problem the suspicious activity could already be over removing all traces of its existence. Therefore, there is a need to target this window of vulnerability so that developing threats do not assume epidemic proportions during this period.

1.5.4 The need for forensic data

Computer forensics is likely to play a major role in dealing with security breaches. A major impediment to swift response is that identifying the sequence of events from the initial compromise to the point of detection is still largely a manual process. Since most recent attacks have been savvy enough to cover their tracks, log files are hard
to preserve and often show little about what happened after the initial compromise. There is a need to collect relevant forensics data rather than rely solely on evidence left unintentionally by a worm. We also want to be able to correlate forensics and audit data from geographically distributed sources and vantage points in order to identify imminent threats and prevent them before they cascade.

This thesis advocates that there is a need to develop internet scale threat analysis, indication and warning capabilities and that the lack of a global view of the Internet and cooperation between peers has long frustrated such efforts.

1.6 Cooperative forensics sharing and policing

Over the years, the Internet has been seriously challenged by newer and advanced threats that have used various different propagation vectors to spread. As a result, it has been difficult to create accurate threat signatures and disseminate them. This thesis advocates that lack of a global view of the Internet is a huge challenge in developing internet scale threat analysis, indication and early warning capabilities. We reiterate that a global view of the Internet is essential in developing internet threat analysis, indication, warning and response capabilities. We view security as a community goal and hence, promote a vision where computer systems are more predictable, more accountable and therefore less vulnerable to attacks and abuse. We hope to create a cooperative responsible community of computer users and believe that safer computing is a goal that can only be achieved through cooperation. We present a framework that facilitates cooperation between domains through active alerts and data sharing.

By sharing threat information and forensic evidence between cooperating domains, we get a more comprehensive view of the extent and nature of developing threats. A defense system looking for an email component for Nimda, would have
missed out on the file sharing component that could have been observed somewhere else on the Internet. Cooperation thus enables us to create better fingerprints for the pathogens quickly. Being able to quickly and securely share this information with neighbors, allows for effective and timely immunization in order to thwart epidemic cascading of threats. Even in the case where immunization is not possible, such cooperation allows a better understanding of the underlying threat. Also, by facilitating the aggregation of information from different sources, a system can potentially trace large scale distributed threats like DDoS attacks and mitigate them nearer to the sources. The Internet community can pool its collective resources to have a pro-active, prevention and damage containment policy complementing a reactive, curative approach towards security incidents.

We believe that domains are ultimately responsible for misbehavior emanating from their networks and should be able to self-police themselves. Each domain can collect a host of forensic data and information about developing threats. However, in the case of a large-scale distributed threat, the behavior observed by any one domain may not be large enough to raise an alarm or to observe the complete propagation vectors of a ‘Blended Threat’. Efficient sharing of such data across administrative boundaries gives a clearer view of threats and their causes.

In addition to sharing alerts and threat related data, domains can also count on trusted partners to perform security related requests for each other. For example domain A under a denial of service attack can request peering domains to police suspicious traffic before it enters domain A. The response to such requests would be observable and can be used as a basis of a reputation scheme within the community to isolate misbehaving networks.
1.7 Thesis and Contribution

We investigate and apply the principles of forensics sharing and cooperative policing to the design of a new wide area security sub-system called Foresight. Foresight targets the time between the first infection of a zero-day worm and the generation of a cure for the worm. Most worms have created havoc during this period of vulnerability. Our framework enables the secure sharing of forensics data between cooperating domains. Our experiments and measurements indicate cooperative forensics sharing and policing as a successful approach that can stem the growth of an unknown threat by observing and analyzing a threat while it is happening and reacting to it in real-time so that the damage is minimized. Foresight also doubles as a preventive and containment platform through timely forensic data dissemination.

Foresight explores a trust-based cooperation scheme which allows highly secure information and evidence sharing between trusted peering domains. Foresight utilizes Paranoid; a global secure file system [101] in order to share log files and forensics data securely and to ensure their preservation in the face of a rogue server. Paranoid is an encrypted, secure, global file system with user managed access control. The system provides efficient peer-to-peer application transparent file sharing. We present the design, implementation and evaluation of the Paranoid file system and its access-control architecture. The system lets users grant safe, selective, UNIX-like, file access to peer groups across administrative boundaries. Files are kept encrypted and access control translates into key management. The system uses a novel transformation key scheme to affect access revocation. The file system works seamlessly with existing applications through the use of interposition agents. Our scheme is secure against a subverted group server since the application of the transform key does not reveal, nor does it utilize in explicit form, the private key of either the group owner or the recipient.
We evaluate the effectiveness of Foresight in locating, tracking and effectively retarding the growth of zero-day network-aware security threats by applying the design principles to different individual contexts.

• We evaluate our data sharing framework in the context of Mail-trap, an application designed to target email as a propagation vector for worms. Mail-trap uses an anomaly based behavioral monitoring scheme to isolate a generalized foot-print or behavioral signature of a zero-day threat by aggregating forensics and auditing data across administrative boundaries. These behavioral signatures are better able to capture the multi-modal and polymorphic nature of unknown threats while capturing forensics knowledge in a reusable form. We quantify the merits of our anomaly based scheme using a test-bed environment. We present results from our simulation environment, that lets us simulate a wide range of zero-day worms with varying characteristics. Our simulation results show that cooperation between domains is an effective way of gaining immunity against unknown threats and stopping worm outbreaks from reaching epidemic proportions. We also describe the design of MASH, a lightweight application that exploits the entropy in incoming spam emails and enables the sharing of spam blacklists across administrative boundaries. The application allows domains to share their email data with anyone across the Internet without fear of security or compromise.

• We measure the performance of our cooperative policing approach in the context of a variety of distributed denial of service attacks. AMP, our service architecture, uses the dynamically configured network components provided by Foresight to perform traffic monitoring, filtering and detection of commonly known DDoS attacks. Our scheme uses cooperative policing mechanisms to detect and reduce the impact of DDoS attacks by reducing the overall attack
traffic in the system. The use of the Foresight architecture also allows us to quickly identify and inform domains about resource misuse which is not large enough to raise local alarms. We evaluate the scheme using a packet level simulation environment. Our results show that cooperative policing enables us to maintain operation even during high powered distributed attacks. It does not require universal deployment and is complementary to other schemes for countering DDoS attacks. AMP has a viable business model where it can be deployed incrementally as a service by domains. Only those domains who want to offer the service need to deploy it, and similarly, only those web servers that wish to subscribe to the service need to install an additional component.

- We present the design and simulation results for a novel ‘multi-modal’ signature scheme that lets us better represent multi-modal threats. We measure the effectiveness of our cooperative forensics sharing using a simulated multi-modal worm. While Mail-trap focuses on combining information from different places to track email based worms, we use SIMDA to show the benefits of combining different kinds of data from different sources. Our results show that cooperation leads to a better awareness of the threat and domains are able to acquire immunity against unknown multi-modal attacks, even ones they have not seen before.
The previous chapter emphasized the need for a new wide area security sub-system that is able to observe and analyze a threat from a global perspective while it is happening and react to it in close to real-time so that the damage is minimized. The thesis of this dissertation is that such a system can also be used as a preventive and containment platform through timely forensic data dissemination across domains. In this chapter we describe the high level goals of the system along with the design of the Foresight architecture. The overall goal of Foresight security subsystem is to complement existing security setups through alerts and forensics sharing between cooperating domains. Adding such defensive depth is important because the effort required to compromise each layer of security adds complexity and potential delay to the attacks.

Foresight aims to detect, isolate and create a behavioral signature for network-aware threats through behavior monitoring. Once such a signature is generated, it facilitates the timely sharing of this information amongst a cooperating community. This allows for an effective and timely immunization. The system utilizes a cooperative policing framework in order to thwart epidemic cascading of threats. Foresight
promotes a vision where computer systems are more predictable, more accountable and hence less vulnerable to attacks and abuse. Keeping in view the thesis of this dissertation, we had the following high level goals for the design of Foresight.

**Detect and Isolate network-aware threats:** The main goal of our work was to develop and evaluate new techniques to detect, isolate and create a *behavioral signature* for zero-day network-aware threats and to develop techniques and infrastructure to share this information securely and quickly within a trusted community. The strength of a behavior blocking scheme lies in the strength of the signature. If the signature is not good, it leads to inaccurate conclusions and poor decisions are made in trying to stop the threat. Given the dynamic nature of the threats, it is increasingly challenging to create a signature of a threat. Therefore, a secondary goal was to provide a better signature generation scheme, able to express an evolving multi-modal threat in a better way. Existing malware signature generation schemes look for some static element which is common to existing or known malware [19]. The attackers simply end up removing or changing this element and the problem continues. The emergence of polymorphic and multi-modal threats has increased the problem manyfold. Worms and viruses spread with multiple propagation vectors and show different behavior on different domains. Our aim was to create a better signature for a zero-day threat through collaboration between cooperating domains on the Internet.

**Internet scale Early Warning and response Mechanism:** Foresight explores a cooperation based scheme to facilitate information and evidence sharing between peering domains and autonomous systems. The idea is that each host might have an incomplete, approximate or inexact information about a particular attack. By combining information gathered from different vantage points, each domain is able to get a much clearer picture of the threats and take remedial measures. A global threat view enables the system to isolate a generalized footprint of the threat
by aggregating forensics and auditing data within a trusted community of domains or Autonomous Systems. Timely dissemination of this information is essential in slowing down pathogens.

In addition to this, it collects a detailed audit trail of the events leading to a threat rather than relying solely on evidence unintentionally left by an attack. Observable network behavior of a host coupled with an application level view can identify threats. This also gives essential clues as to the causes of an infection with a high probability. A fusion of forensics data collected from multiple geographically dispersed domains is therefore essential to determine the way a malware exploits the underlying hosts. Better forensics are essential to detect, trace back the threat, locate effected components, undo its effects and identify potential causes.

**Trust-based Cooperative Traffic policing:** Foresight views security as a community goal and we wanted to create a global, accountable, trusted community in order to facilitate cooperation between domains. Sharing threat information and forensic evidence within the trusted community gives us a more comprehensive view of the extent and nature of developing threats. The goal of the system was to extract a generalized footprint of a threat through cooperative threat monitoring and data sharing. From a routing point of view, the Internet can be considered to be partitioned into a number of independent sub-networks called the Autonomous Systems (AS’s). This thesis uses AS’s or domains as the basic cooperating entities or building blocks of our trusted community. AS’s are assumed to be under a single ownership, trust and administrative control and hence are subject to different needs, constraints and opportunities. Throughout this document, we use the terms AS’s and domains interchangeably. Using Autonomous Systems (AS’s) as the basic entity in our system simplifies the number of entities greatly. However the same concepts can be applied to any sized subnetworks with a single administrative control.
Foresight, domains also rely on trusted partners to perform security related requests for each other. Since domains are ultimately responsible for misbehavior emanating from them, an important goal of Foresight is to encourage systems to perform self-regulation and policing. For example, domain A, under a denial of service attack can request peering domains to police suspicious traffic before it enters domain A. The response to such requests is observable at domain A and can be used to enhance or decrease a peering domain’s reputation. Foresight provides community oriented services by rewarding responsible and good samaritan behavior. Misbehavior results in isolation from the community over the long run. Foresight explores an incentive based model where domains avoid individualistic behavior in order to achieve a cooperative optimum goal between the group.

**Build a global secure access control system:** The distributed and loosely coupled design of such a global system raises interesting challenges. Traditionally, domains are reluctant to share security threat related information because that may disclose private information, local topological information, security policies and vulnerabilities. For such cooperation and sharing of data between domains to be viable, the sharing system needs to deliver hard security guarantees. Foresight facilitates global data and forensics sharing with enhanced security and privacy, minimal administrative overhead and application transparency. The Paranoid secure access control system provides the participating domains with a global computing environment, without the fear of compromising the security of information they consider private or privileged.

**Accuracy of the scheme:** An essential question regarding the effectiveness of an anomaly based security system is the degree of false positives and negatives created by such a system. Weak evidence leads to inaccurate conclusions and poor decisions that can cause more damage and liability than a threat itself. False positives have long been a problem with security systems and strangely enough, the better
the system gets i.e low false positives, the more dangerous false positives become, because when the system is really good, users are more likely to trust it.

Intrusion detection systems display the other side of the spectrum where the false positives are too high. Due to the high alarms, users become complacent and it is easy for actual intrusions to slip through unnoticed in the generated alerts. Ideally, a security subsystem would provide very low false positive and negative rates. Our goal to make a reliable system with low false positives.

A related goal of the system was to ensure the overall level of traffic did not exceed a certain maximum level. An early warning system that broadcasts high false alarms to the whole community would be worse than an attack. In other words, the protocols needed to be ‘polite’, making sure that the traffic generated by the system did not overwhelm actual traffic.

**Flexibility and inter-operability:** Past experiences suggest that any level of security is eventually penetrated. One primary concern in designing such a system is the highly variable requirements of all the different domains participating in the scheme. Not every domain or administrative system has access to the same resources, has the same network or usage profile. The situation is further complicated by different administrative policies regarding sharing or information and data across domains. Therefore, the design of the system is flexible enough to allow individual domains to clearly specify their data sharing policies.

This thesis claims that domains having similar profiles share similar threats as well. For instance, educational institutions have similar traffic profiles and hence there is incentive for them to share information between each others rather than with for instance, ISPs in Pakistan. Foresight adopts an open ended building block style design to allow newer and improved detection techniques to be incorporated in the system.
**Automatic response mechanisms:** The previous chapter establishes that next generation computer worms could saturate networks on the order of minutes, rather than days and hours. A human centric response would be insufficient to counter the threat and maintain mission integrity in the face of worm attacks. Foresight therefore aims to create robust networks capable of responding automatically and dynamically to self propagating zero-day worms and exploits. An effective alarm system needs to be able to propagate the alarm quickly through the community in order to be able to affect the epidemic nature of the threats. The main idea is to make a system that facilitate the quick creation and dissemination of a threat signature through cooperation. We do note however that this goal is in direct conflict with the low false positive goal. No system can therefore hope to achieve both goals simultaneously for all classes of threats. We favor a vector control approach by targeting individual propagation vectors of worms for which our solution is workable and gives good performance results.

**Vector Control:** The scope and scale of possible attacks is broad enough to effectively rule out any silver bullet cure for malware. With this in mind, our design borrows the idea of vector control from epidemiology. The idea behind vector control is to check the spread of a disease by limiting or eradicating the propagation vectors of vector born diseases. Similarly, we wanted to focus on the various transmission vectors of malware in order to retard threats and improve the overall security and survivability of the network against malware. Even in case of partial vaccination, unvaccinated domains would experience a reduced infection rate due to the presence of the vaccinated domains in the network. Our results suggest that vaccinated domains act as “firelines” in the spread of attacks, slowing or preventing their further transmission to others. While the protection offered is not curative, the preventive measures adopted by the system are able to prevent damage to critical system resources during the ‘Submission to cure generation’ window of vulnerability.
2.1 Foresight system design and architecture

This section describes the design of Foresight; a security system designed to meet the goals and requirements described earlier. Figure 2.1 presents a high level design of the system. The four primary system components are the Knowledge Manager, Collaboration Agent, Sensors/Enforcers and a global file access control system that allows users to selectively, securely, and easily share information with others.

Foresight relies on a set of sensors to observe anomalous network or host behavior. Foresight **Sensors** are essentially programs that monitor a specific variable, activity or condition of a machine or the network. Foresight utilizes a hybrid of host and network based programmable sensors to collect a finer-grained, threat specific information from a domain. Each Sensor monitors its local resources and sends data packets to the Knowledge Manager whenever a **significant** event happens. **Significant** events and the sensors are discussed in detail in later sections.
Foresight is based on a hierarchical design where the Knowledge Manager (KM), aggregates the information collected by various sensors in a network. The Knowledge Manager is responsible for collecting and correlating local and global threat information, generating signatures and alerts for threats, policing of traffic through enforcers and storing/sharing of forensics data. The KM collects and correlates information from local sensors as well as globally distributed sensors (through the Collaboration Agents) in order to create a unified global view of the threat scenario. The KM enables us to perceive the trusted community as a whole and not split up into separate independent units. The idea being that each domain is no longer be dependent on just local information when confronting unknown threats. This information is stored into a globally accessible secure storage, which can be accessed through the Paranoid group access protocol [102].

The Knowledge Manager uses enforcers to prevent further damage or compromise to critical system resources and components in the face of an attack. Enforcers are logically separate entities from the sensors and are able to shape and police both incoming and outbound traffic according to the policy handed down by the Knowledge Manager. The actual actions range from quarantine of machines, disabling of file systems, disabling ports, re-tooling the firewall rules to drop offending traffic or simply informing a human administrator. A major effect of viruses and worms is the clogging of the network and mail-servers etc. Enforcers also act as drawbridges for traffic in such a situation, shielding the system from both inside and outside threats by using firewalls, content filters and routing blacklists.

The Collaboration agents (CA) is responsible for sharing all security related information with peers. They are responsible for the creation and running of trust and reputation relationships between domains as described by chapter 5. The CA works in collaboration with the Knowledge Manager to keep track of reputation scores with peering domains. The CA is also responsible for creating and maintaining
access groups that have rights to shared forensics data. The Knowledge Manager provides feedback to the CA which in turn, updates trust relationships and access groups. The CA and Knowledge Manager are logically separate entities which can actually be implemented on the same machines.

Foresight uses *Paranoid* [102] group semantics to address privacy concerns and to secure globally shared data and provide global file access control. File contents are locked via encryption and are unlocked only with the correct key. Access control thus transforms into a key management problem. Users are implicitly authenticated by their ability to gain access to keys. Paranoid uses a novel approach using transform keys to address the key distribution and revocation problems. The access control protocol lets each domain define individual access groups according to their trust relationship with peers. Each trusted member has access to group accessible files without having a shared group secret. Such a scheme prevents a group member from adding members to the group by sharing the group secret. The system works because only trusted group members with the right capabilities have informational access to the shared data. We have successfully implemented the scheme in the design of a globally accessible secure file system.

One of the biggest problems preventing the global sharing of data is security and privacy concerns. Our use of secure group semantics addresses that problem by allowing each domain to select its data sharing group. We utilize an interaction based reputation scheme that lets a domain quickly revoke access privileges from non-cooperation domains. The lack of a group shared secret means that access to shared resources can always be traced back to the domain responsible. This prevents free-riding behavior from domains. Lack of accountability would encourage a situation where users would utilize shared resources and alerts without committing their own resources to the joint efforts. The decentralized key management and access control
scheme is also robust to denial of service attacks and cannot become a single point of failure during attack situations.

2.1.1 Sensors/Enforcers

Foresight utilizes a hybrid of host and network based programmable sensors to collect a finer-grained, threat specific information from a domain. Sensors are essentially implemented as daemons that monitor a specific variable, activity or condition of a machine or the network. Each sensors has two operation modes and maintains a bidirectional communication channel with the Knowledge Manager. Foresight sensors perform a statistical anomaly based analysis of the data they observe. In the passive mode, sensors monitoring different variables report ‘interesting’ deviations from normal values to the Knowledge Manager. In the advanced mode of operation, the sensors adapt to provide a finer grained logging and auditing. This helps to keep the volume of data generated to manageable proportions and resource expensive operations are only performed when needed. Most of the data collection happens through external sensors. The advantage of this approach is that external sensors can be easily modified, added or removed from a host without disturbing any other operation. However, host-based sensors are used to provide contextual information to the data collected by the external sensors.

Gen-prof is the simplest sensor in the architecture. Gen-prof is essentially a packet sniffer with a simple analysis engine attached to a reporting module. It records traffic flow data, number of connections initiated to and from a single client and sudden increases in traffic through specific ports or protocols. The reporting module has a bidirectional communication link with the Knowledge Manager. Gen-prof can either be installed on a dedicated machine, passively collecting data, or it can be co-located with a network firewall. Very little state is maintained by Gen-prof and it keeps accounting information about each machine on its network over a user
specified window. It keeps a list of protocols and ports for each IP address along with the number of distinct connections made to and from this machine in the last $X$ minutes. The choice of time window ‘$X$’ is interesting since it directly effects the storage requirements at the sensor. We chose an arbitrary value of 30 minutes in our design. Any changes in the accounting data or any unknown traffic is reported to the Knowledge Manager.

A specialized sensor called **Server-prof** monitors the health of the web-server and the file server. The use of the term file server here is overly simplistic, however the same principles apply to any storage architecture attached to the network. Server-prof looks for performance degradation symptoms such as high CPU utilization, resource exhaustion such as memory and buffers unavailable for legitimate traffic and packet queue backup leading to indiscriminate packet drop. It samples the network usage, cpu, memory, disk usage and processes running on a server. For each process, the daemon maintains a list of files that it has accessed in a given time slot. The choice of time-slot is arbitrary and can be set through the KM interface. The Server-prof has access to the server log-files and in the active data collection mode, it can provide information on services, processes and log files to the Knowledge Manager. The server health data is periodically logged onto the Knowledge Manager. Server-prof is implemented as a light weight host-based sensor. Detailed operations for this sensor are discussed in chapters 4 and 7.

**Mail-prof** is a specialized sensor designed to work with the mail server. Mail-prof collects simple accounting data regarding email traffic coming from any host at a particular time. This includes the number of emails being sent from any particular account or IP address within a given time-slot. This accounting data is used to perform statistical anomaly based analysis of the traffic. Mail-prof detects sudden spikes in the out-bound email traffic from a machine and reports to the KM. This behavioral approach to monitoring outgoing email traffic works with a high degree of
accuracy since malicious email activity is statistically different from normal traffic. This daemon passively monitors all traffic intended for the SMTP server however, the same functionality can be attained from monitoring the SMTP log files. Further details on the Mail-prof are given in chapter 3.

The **Edge-prof** provides similar functionality to Mail-prof but it is assigned to ingress/egress routers where we use them to monitor and manipulate traffic entering and leaving a domain. Conceptually, this daemon is co-located with the router where it has access to and can customize the router data plane. Currently we monitor traffic through user level processes that include a combination of raw packet captures like tcpdump [91] and tcpflow [92] in order to capture per packet and per flow data from the traffic. The design of a router based version of Edge-prof is beyond the scope of this work. The Edge-prof periodically logs the data collected to the Knowledge Manager.

A domain administrator chooses what sensors and enforcers to install, contingent upon needs, topology and resources available. A strategically designed sensor and enforcer architecture can monitor network and host-based anomalies to detect malware in time to protect critical resources.

### 2.1.2 Enforcers

Enforcers are a logically separate entity from the sensors and are able to shape and police both incoming and outbound traffic according to the policy adopted by the local domain. While sensors observe data or traffic anomalies, enforcers are used by the KM to dynamically isolate misbehavior. The actual actions range from quarantine of machines, marking suspected email activity, disabling of file systems, disabling ports, retooling the firewall, rules to drop offending traffic or simply informing a human administrator. A major effect of viruses and worms is the clogging of the network and mail-servers etc. Enforcers also act as drawbridges for traffic in such
a situation, shielding the system from both inside and outside threats by using fire-
walls, content filters and routing blacklists. Foresight employs specialized enforcers
to counter email borne threats (Chapter 3) and DDoS attacks (Chapter 4).

Both enforcers and sensors can be co-located in edge-routers, servers, dedicated
machines on the network and/or subscribing hosts. The enforcers maintain a bidi-
rectional communication with the Knowledge Manager. The same design choices
that apply to sensors are relevant here as well. Host-based enforcers can be used to
start/stop processes and services, adjust security permissions and disallow access to
certain resources etc. Host based Enforcers are placed on hosts as part of service
level agreements with the AS’s. Enforcers can also be used to protect the security of
data using cryptography and replication in a scheme similar to the one proposed by
Gehani [26]. Dynamic adjustments of the access control configurations of resources
can also be performed.

2.1.3 Collaboration Agent

The Collaboration Agent is the point of interaction between domains. Foresight
relies on the creation of bilateral trust relationships between administrative peers.
The Collaboration Agent or (CAgent) is responsible for creating and maintaining
trust and reputation relationships between domains. The details of the trust and
group managements schemes are given in chapters 5 and 6. The CAgent works
in collaboration with the Knowledge Manager, to keep track of reputation scores
with peering domains. This trust notion is used by each domain to grant secure
access to forensics data by creating and maintaining access groups that have rights
to encrypted data. Once trust is established, Foresight uses Paranoid [102] group
semantics to address privacy concerns and to share secured data across administrative
boundaries.
Each domain decides what type of data it wants to share with the community and advertises this information globally. For instance, a domain ‘A’ that collects email related data inside its network and is willing to share it with the community will advertise this information. Another domain might be reluctant to share email data with others but willing to share HTTP log file data. We store this information in an XML file in a publicly accessible place such as the web-server. The choice of web-server is purely arbitrary. Our current scheme only advertises the type of data that a domain is willing to share. A richer set of information can potentially be stored in this XML advertisement which could include operating system information, software versions/patches, log file formats, topology information etc. This could help neighboring domains to better understand a domains profile and share data with similar profiles. For instance academic institutions would have similar usage patterns and hence similar vulnerabilities and could benefit from data sharing. The current version of Foresight does not address advanced profile advertising.

Neighboring domains that are interested in sharing a particular type of information with a domain ‘A’ can send requests to be included in the access group for this information.

The CAgent maintains an import and export matrix to keep track of what data is to be shared with whom between peering domains. Figure 2.2 illustrates the matrix with an example. The matrix shows the home domain having an email data sharing agreement with domains A, C, G and H. The matrix also stores the IP address of the peering Collaboration Agent as well as the frequency of the data sharing. We assume bidirectional data sharing between peers implying both import as well as export of forensics data. Technically, these are two separate operations and can be handled by storing a tertiary value in the matrix signifying, unidirectional, bidirectional or no sharing. Domains often have asymmetric resources and expertise and hence in real life some resource strapped domains would probably be better off only receiving
forensics data. The import/Export matrix is based upon the access groups for each kind of data shared by a particular domain. For instance the Email group shown in figure 2.2 contains domains, A,C,G and H.

A separate frequency matrix keeps track of how often the data is to be shared between domains. The frequency matrix is necessitated by the need to keep a check on the amount of traffic that flows between domains. The frequency matrix therefore imposes a structure on the interaction between domains. Reliable content distribution methods use acknowledgements to ensure all receivers get complete and correct information, this however leads to an acknowledgement implosion issue. The issue can be resolved by the use of negative acknowledgements though. However, for the sake of Foresight, negative acknowledgements cannot work due to the nature of the traffic generated by the system. While the frequency matrix does impose a structure on the communication between domains, not all the traffic generated by the system is a periodic occurrence, and negative acknowledgement requires the receivers to know they have missed something. So while a frequency matrix keeps track of when to
contact a peering CA, it also gives an upper bound on the staleness of the state of the system. All communication between collaboration agents is digitally signed and time-stamped.

The frequency values are stored as a pair of numbers representing a potentially asymmetric transfer of information between domains. Apart from the periodic push/pull information exchange, each time a security event of ‘interest’ occurs, the CAgent notifies all the recipients that are subscribed to that event. CAgent is designed as a collection of threads, each assigned a specialized task. One listening thread specializes in communicating with peering domains to answer queries on the encrypted log data. A separate thread forwards data and logs received from others to the Knowledge Manager. The CA and Knowledge Manager are logically separate entities which can actually be implemented on the same machines.

There is an inherent trust issue when domains are talking to each other. How do we identify who we are talking to? Foresight assumes each domain is identifiable and verifiable by its unique public/private key pair. Domains are implicitly authenticated by their ability to access keys. We assume that the key generation exchange and verification protocols to be secure and outside the scope of this research.

2.1.4 Knowledge Manager

The previous chapter motivates the need to have increased coordination between local and global sources of data. The Knowledge Manager (KM) lies at the heart of the whole scheme that enables this coordination between local sensors, enforcers and peering domains (see Figure 2.3). It is responsible for

1. Collecting and correlating local and global threat information

2. Generating signatures and alerts

3. Policing of traffic through Enforcers
4. Storing/sharing of forensics data

The KM collects and correlates information from local sensors as well as globally distributed sensors (through the Collaboration Agents) in order to create a more comprehensive view of the current threat scenario. As discussed earlier, this model is inspired by the way a human brain creates a unified view of the world through an interaction of all five senses. The KM collects data from all local sensors, aggregates it and shares it with the rest of the community. The KM enables us to perceive the trusted community as a whole and not split up into separate independent units. This enables local domains to learn from others. Combining different kinds of information across administrative boundaries enables us to create a better threat signature for emerging threats. Using their sensor architectures, each domain observes a growing threat from its unique perspective and creates a signature that best fits the symptoms.
it observes. A union of these signatures is a much better representation of the overall pathogen. This is specially true in the case of polymorphic threats.

The implementation of the Knowledge Manager consists of three primary components responsible for coordinating the working of the Knowledge Manager. Sensord manages the communication between the Knowledge Manager and the different senors installed on the network. CollabD is responsible for interacting with the Collaboration Agent. The CAgent daemon updates the reputation and hence the group management components based on its interaction with the CollabD. ParanoidD handles the secure storage portions of the system. The secure file system is accessible both from the Knowledge Manager as well as the Collaboration Agent. The KMEngine coordinates the interaction between the different daemons and the installed modules on a Knowledge Manager. The design of KMEngine is kept flexible so that it can be used with a variety of specialized modules depending on what kind of data a domain wants to generate and share with the community. Examples
of these modules include a separate module for email based pathogens section 3, a module for countering denial of service attacks (chapter 4), spam, firewall data.

The Knowledge Manager has an open ended building block style design so that newer and improved malware detection techniques can be incorporated in the system. The actual sensor and enforcer architecture varies from domain to domain depending on what modules are installed at that domain. A separate module for instance can be quickly added to collect snort signatures and share through Foresight architecture. A similar philosophy can be applied to Spam Assassin filters.

The KM is responsible for performing traffic shaping and policing on behalf of local users as well as peering domains. The KM instruments enforcers in order to make sure local traffic does not misbehave. Local sensors perform a statistical anomaly based analysis of the local traffic. Once an alert is generated, the KM collects the data from the sensors and tries to create a signature for the threat which it them shares with its trusted peers. How a domain chooses to respond to a misuse of its resources is specified in a security specific service level agreement (SLA) negotiated between the hosts and their controlling domains. Our current design includes option to either block or rate limit a particular traffic flow. The KM uses enforcers to prevent further damage or compromise to critical system resources and components in the face of a suspected attack. The KM also performs policing requests on behalf of neighbors. This includes controlling locally originating traffic as well as transit traffic. The details of the KM operation are given in the context of Mail-trap and AMP in chapters 3 and 4.

Because of the ever growing nature of the Internet it is impossible for a single machine or a group of machines to store global knowledge concerning all potential participants. The centralized approach based on centralized management is thus difficult if not impossible to manage for a reasonable sized network. Our scheme
proposes partitioning a network into manageable size groups with their independent collaboration agents and secure data repositories.
In the chapter, we present Mail-trap, a proof of concept system designed to illustrate and evaluate the collaborative forensics sharing aspects of the Foresight architecture. The Foresight architecture facilitates cooperation and dissemination of timely alerts. Mail-trap targets email-borne vectors of malware. We use an anomaly based behavioral monitoring scheme to detect misbehaving mail traffic. Once misbehavior is established, active sharing of information between cooperating domains allows us to create a better picture of the threat. Mail-trap illustrates the use of Foresight’s cooperative policing model to identify and pre-empt polymorphic email-borne threats. Cooperation between domains greatly increases the effectiveness of our approach. Domains are able to pre-empt attacks and respond to malware behavior that they have not seen before. It can also be used to identify and isolate ‘zombie’ machines that are used to launch smart attacks that are able to evade local behavior monitoring. Our results show that behavior monitoring alone can be an effective tool for certain classes of email borne malware detection. Conceptually though, the Foresight architecture can work with any set of tools and programs that detect malicious or suspicious attack traffic.
In the second part of this chapter, we explore issues related to Spam. We present MASH, an application that exploits the entropy in SPAM data to increase the detection rate of incoming spam traffic through sharing of local spam information. Our results with MASH are comparable to industry standards on spam detection schemes. This shows cooperation between domains as an effective way to detect and isolate machines that are part of existing botnets. Our scheme does not require universal deployment however the efficiency of all the applications improve significantly with increased cooperation. Mail-trap and MASH are meant to complement existing email filtering schemes and the data-sharing techniques can be applied to various other anti-virus/spam programs.

3.1 Email-borne malware

Email is by far the preferred and easiest means of virus propagation [43]. It has the ability to transport all sorts of data including executable code, programs, scripts and macros and while email-based worms have shown little variation in recent years, several high profile worms have used Outlook and SMTP email programs to clog the Internet. Email has been an important propagation vector for malware and is almost universally present in all multi-pronged attacks. Spam is a closely related problem to email based malware. Spam is the electronic equivalent of junk mail that has become a huge problem on the Internet. According to IronPort’s [77] internet security trends report, 120 billion spam messages were sent every day worldwide in 2008. Initial spam was targeted towards selling certain products or promoting websites etc. Spam has shown increased sophistication in the last few years. The trend has changed significantly towards recruiting machines for reusable attack platforms such as botnets. Recent viruses like SoBig.F for instance were sent out specifically to recruit “zombie machines” for spammers. The virus opens up the infected machine
to spammers, who can then route spam e-mails through these machines. Since the IP addresses of these machines are new, they do not appear in the IP address blacklists and millions of spam e-mails can be routed through them before they get blacklisted.

Existing email-virus prevention techniques focus on identifying malicious attachments at the incoming email server. While this approach is quite successful in handling known viruses with known payloads, newer and unknown pathogens pose a different problem. This approach also fails to deal with email worms that spread without attachments. Newer worms have propagated by sending links to infected mini http-servers through emails. In fact 80 percent of spam detected in 2008 contained some form of URL. Users inadvertently download worm code when they click on the links.

3.2 Email, SMTP and envelope header

Before we start describing the Mail-trap architecture, we need to understand the underlying email protocols and their weaknesses. The SMTP protocol lies at the heart of the email systems. Email works as a result of communications between email clients (ex. eudora and Lotus Notes), Mail servers (POP3/IMAP and SMTP) and DNS servers. Figure 3.1 shows the working of a very simple email system in operation. The sender composes an email using an email client like Outlook and addresses it to someone@somewhere.net. The email program sends it to the SMTP server where the sender’s account is. The SMTP protocol normally begins with a DNS lookup of the MX-record corresponding to the domain name of the recipient’s email address. The SMTP server contacts the DNS server to get the location of somewhere.net and establishes a two-way transmission channel to the receiver SMTP server [73]. Essentially SMTP is a lock-step connection oriented protocol. While a sending SMTP server can advertise any information in the email header, the IP
address of the sending server is visible to the receiver. We use this property later on to track misbehavior.

The email message essentially contains the Header and the Body of the email. Email headers are the lines added at the top of a message as it passes through the email system. Standard email programs only show a few of the fields like To:, From:, Subject:, Date; however, the email header contains the complete route a message took as it passed through the email systems. The Received: headers of any email message give the most interesting and often misleading clues about where a message originated and what path it took to get to the receiver. Any MTA that receives or transfers an email message adds its header sequentially to the email message. As we go down a message header, the last received header is the first one added to an email message. This effectively means that reading the
Table 3.1: Sample Email Envelope header

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return-Path</td>
<td><a href="mailto:lfocomet@dfoco.de">lfocomet@dfoco.de</a></td>
</tr>
<tr>
<td>Received:</td>
<td>from nat-moc6.aster.pl (nat-moc6.aster.pl[212.76.37.180]) by one.cs.duke.edu(8.14.2/8.14.2) with ESMTP id m2TIUe15024487 for <a href="mailto:fareed@cs.duke.edu">fareed@cs.duke.edu</a>; Sat, 29 Mar 2008 14:30:42 -0400(EDT)</td>
</tr>
<tr>
<td>Received:</td>
<td>from [212.76.37.180] by mail4.megsystems.net; Sat, 29 Mar 2008 22:16:09 +0100</td>
</tr>
<tr>
<td>Date:</td>
<td>Sat, 29 Mar 2008 22:16:09 +0100</td>
</tr>
<tr>
<td>From:</td>
<td>“Jane Doe” <a href="mailto:lfocomet@dfoco.de">lfocomet@dfoco.de</a></td>
</tr>
<tr>
<td>X-Mailer:</td>
<td>The Bat! (v3.0.0.15) Home</td>
</tr>
<tr>
<td>Reply-To:</td>
<td><a href="mailto:lfocomet@dfoco.de">lfocomet@dfoco.de</a></td>
</tr>
<tr>
<td>X-Priority:</td>
<td>3 (Normal)</td>
</tr>
<tr>
<td>Message-ID:</td>
<td><a href="mailto:229386707.08511425582196@dfoco.de">229386707.08511425582196@dfoco.de</a></td>
</tr>
<tr>
<td>To:</td>
<td><a href="mailto:fareed@cs.duke.edu">fareed@cs.duke.edu</a></td>
</tr>
<tr>
<td>Subject:</td>
<td>Email Header example with entries spoofed</td>
</tr>
<tr>
<td>MIME-Version:</td>
<td>1.0</td>
</tr>
<tr>
<td>Content-Type:</td>
<td>multipart/alternative</td>
</tr>
</tbody>
</table>

Received Headers: In the reverse order we can trace a path back to the sender of the message. A simple email header is given below to explain the different fields. For instance, if we analyze the header given in Table 3.1, the first server to receive the email was mail4.megsystems.net from 212.76.37.180. Since the receive headers are to be read in the reverse order, one.cs.duke.edu received the email from nat-moc6.aster.pl. A careful look at the header suggests the first header is spoofed, since the IP address that sent the message to mail4.megsystems.net is the same one sending it to one.cs.duke.edu. In any case, the Return-path and Reply-To addresses have nothing to do with the servers sending the email. Upon further investigation, the message body indeed showed the email message to be carrying malware, along with a URL directing to a well-known spam server.

The email header shown in Table 3.1 highlights a few important issues with the SMTP protocol. The protocol suffers from an inherent lack of authentication: this means that a receiving server cannot verify that a particular message was sent by
the person it claims to come from. The only information that we can safely rely on is the last-hop mail-server in the envelope header of an email message. In this case, that last-hop server is one.cs.duke.edu getting the message from the IP address 212.76.37.180. Everything else in the headers can be easily spoofed. The situation is further complicated by the fact that operating systems access rights are enforced on processes and every process running on a users behalf has necessarily all the privileges of the user. This is why an email program has the same privileges and credentials as the user and the virus running in it does too. ‘Zombie’ or compromised machines sending out email-viruses are therefore exactly the same as normal users on those machines sending emails.

3.2.1 Establishing Normal Email patterns

The main idea behind the Mail-trap application is that malicious or viral email activity is statistically different from normal email traffic. Our observations of normal email activity were based upon our collection of over 1 million emails collected over a three month period across three administrative domains. Figure 3.2 shows the average emails sent per user per hour from one of our data sets over a 24 hour period. The data shown in the figure shows the average results for 10 functional email accounts over a two week collection window.

While, the number of emails sent per hour varied significantly over the the course of a day, email activity was particularly clustered around the the lunch hour. From a gradual start it peaked around mid-day, slowly tapering off as the day came to and end. Having collected the same data over two weeks, 3.4, we showed significant regular patterns across the dataset despite the daily fluctuations. In face, the daily chart was almost exactly the same across the two week period. Figure 3.3 displays the email behavior of the author over a four year period. The pattern clearly shows nocturnal habits of the user! Closer inspection the data however reveals clear usage
patterns in the data. Similar efforts have been made with the Enron email corpus which was made publicly available following the Enron scandal. This data collection provides us with the benchmark data for our threshold based behavioral monitoring of the email activity. It also serves to confirm our design choice of not sampling email traffic since email activity, as shown by the numbers above, is limited to manageable proportions.

3.2.2 Worm behavior testbed

We created an isolated sand-boxed windows environment to observe the network behavior of a few well known email worms and viruses. We installed modified or actual versions of worms such as Melissa, IloveYou, Bagle and Mydoom on the machine and observed their email vector of propagation over a five minute time window. Bagle
and Mydoom were unique since their authors apparently released the source code themselves. While Bagle is written in assembly language, the source code is quite well commented. Mydoom on the other hand is written in C. We chose these particular pathogens for two reasons. Firstly, we were able to locate real live samples or code for all of them through our email archives as well as web resources. This allowed us to disable the network propagation aspects of the programs and output the data to files. This helped prevent undue infected traffic on the network as well as providing a realistic sample. Secondly, they represented a good mix of all the different characteristics that are present in most mass-mailing worms. The results from our experiments with the live samples are summarized in Table 3.2
Melissa was included in the list of viruses because it was one of the earliest and the one which was simplest in some ways. An analysis of the source code of the virus reveals that the virus propagates with the same subject and message body. This was entirely consistent with our experimental observations. The file attachment size however varies based on what document actually was infected. The virus takes the

<table>
<thead>
<tr>
<th>Melissa</th>
<th>ILoveYou</th>
<th>Bagle</th>
<th>Mydoom</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Emails Attempted</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>yes</td>
<td>yes</td>
<td>Attachment</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>18</td>
<td>10</td>
<td>SMTP Engine</td>
</tr>
</tbody>
</table>

Table 3.2: Malicious Worm behavior on windows platform
top 50 entries from a user’s mailbox in MS outlook and sends out infected versions of the document as email attachments to those 50 users. Iloveyou virus on the other hand targets all the email addresses in the mailbox. Other than the number of emails sent, Iloveyou virus was pretty similar to Melissa since the subject and message body were the same across all the sent emails. Mydoom was the most interesting virus that showed the most interesting behavior characteristics. We expected the total number of emails sent to be much higher for mydoom since its email harvesting mechanism is much more advanced compared to melissa and iloveyou. The virus generated its own email addresses rather than relying on the ones in the address lists. While it harvested domain names from the infected machines, the worm body came with its own set of user names that it appended to the domains (see table 3.3). The worm also deliberately avoided certain domain names such as microsoft and hotmail etc. This was probably done to avoid detection at these well known websites.

Bagle source code also included code ruling out popular anti-virus domains, administrator accounts and email services in order to avoid detection. Our experiments with the malware were limited to observing the immediate network behavior of the machines once they were infected. Significantly though, the number of sent emails by each of them was in sharp contrast to legitimate users we observed earlier in our data collection process. We limited our data collection to a 5 minute window starting once the network behavior was observed. This approach let us collect data regardless of variable incubation periods. The behavior identified in our experiments clearly suggested a distinct difference in observable network behavior as compared to normal clients. The data also confirmed our belief that observing traffic patterns at the SMTP server alone is not sufficient since technological advancements in newer threats have resulted in some pathogens utilizing their own server components to propagate.
3.3 Mail-trap Architecture

Mail-trap uses multi-level network and host-based sensors to detect misbehaving email traffic. Behavior monitoring approaches to malware detection assume that malware traffic is typically noisy and different from normal traffic. The Mail-trap sensor architecture essentially monitors the behavior of outbound email traffic in a network. We suggest that due to its distinctive behavior, malicious email activity can be detected at various points in a network. Every anomaly based technique introduces its own set of false positive and negatives. Our choice of detection scheme was based on the observable network behavior of the worms in our test-bed. However, we do note here that our scheme would work with any detection scheme capable of identifying misbehavior. The focus of Mail-trap is to evaluate the Foresight architec-
Figure 3.5 shows a skeletal design of the sensor architecture utilized by Mail-trap.

The *Mail-prof* sensor described earlier in chapter 2 is assigned to the mail server. It is an external sensor, implemented as a daemon that sits on the critical path between the SMTP server and the rest of the network, passively observing traffic. Since Mail-prof is an external sensor, it imposes no performance penalty on the server. Unlike other sensors, Mail-prof need not perform any sampling of the traffic since the volume of SMTP traffic is not high under normal traffic conditions (we verified this assumption through real life data collection). An alternative design would place the daemon on a network firewall placed in front of the Mail-server. As described earlier, *Mail-prof* collects simple accounting data regarding email traffic coming from any host over a given time window. As part of the data collection,
Mail-prof is also responsible for extracting and storing summary information about the message sizes and attachment types along with the email ‘envelopes’ or headers. Mail-prof maintains a bidirectional communication channel with the KM through its communication module and the size of the data collection window can be set dynamically by the Knowledge Manager. The data collected is used to perform statistical anomaly based analysis of the traffic. The type of data collected per host includes the number of emails being sent from any particular account or IP address within a given time-slot.

Mail-trap relies on detecting anomalous SMTP traffic in order to detect malware. Due to privacy and security reasons as well as the size of the data involved, we do not log actual emails at the sensors and limit our scheme to email envelope headers. One of the reasons for making this choice was that we were unable to collect live email data on any network. We were allowed however to collect envelope header information on all of them since most of that information is readily available from server log files. Also, our sensors exclusively observe the behavior of outgoing emails traffic. Our extensive analysis of server log files and malicious emails suggests that incoming email based malware does not have distinctive behavior characteristics different from normal traffic, unless its part of a denial of service attack or an email bomb attack. This is also in line with our philosophy of domains being responsible for their own misbehavior. The design of Mail-prof is shown in Figure 3.5. It is essentially a packet sniffer feeding a TCP flow reconstructor that collects SMTP traffic. The parser strips the envelope header from the reconstructed flows and feeds the analysis engine which keeps track of all the individual hosts and their email behavior data.

Mail-prof maintains mean and deviation statistics for the number of emails sent per time slot by each IP address. We chose the arbitrary value of 10 minutes to signify this time slot. While our system assumes a certain training period during which the sensors learn the normal network behavior of a particular machine in a
network, the thresholds for abnormal behavior as well as the time windows can be set manually through the Knowledge Manager. Mail-prof has two distinct modes of operation. The passive mode of operation is the default mode of operation where the sensor is simply collecting the data locally. If the sensor observes suspected abnormal email activity coming from a machine, it immediately raises an alert and passes on the suspect traffic to the KM. Mail-prof takes 2.5 deviations above the average as abnormal network activity. Having raised an alert, Mail-prof resumes its passive mode of operation.

**Edge-prof** monitors network traffic emanating from a host that bypasses the mail server completely. If the traffic is bypassing the designated mail-server, the likelihood of it being a malicious activity increases even more. In fact, a trivial solution to slow down malicious out-bound email traffic would be to force all mail traffic to go through a rate-limited mail-server. In such a situation, the mail-server would become an excellent vantage point to observe all mail traffic patterns. In the absence of such a solution however, we require the ability to monitor traffic at strategic points on our network in order to observe malicious traffic.

Edge-prof is implemented as a user level process co-located on an egress firewall machine that essentially performs the same operations as Mail-prof on outbound email traffic. The exact location and placement of the Edge-prof can change depending on the size of the network as well as topology. We chose the egress firewall since the sensor is able to observe all outbound traffic. Edge-prof maintains the same statistics for each machine on the network. This is important because as discussed earlier, newer threats routinely come with their own SMTP engines and servers. Most of the newer email viruses such as Bagle.F, Netsky.D, Mydoom.U and SoBig.F come with SMTP engines of their own, completely bypassing the local servers. Similar trends have been noticed with botnet activity where the malware contains enough server components to be self-sufficient as far as propagation.
The Edge-prof can also be used to generate alerts as a result of SLA violations by users of a domain. For instance, if a domain does not allow users to host their own SMTP servers, all email traffic or port 25 traffic originating inside a domain which is not bound for the mail server can raise an alarm. In order to avoid double counting, the Edge-prof does not collect any data which originates at the mail-server. **Gen-prof** is the simplest form of sensor used in the Mail-trap application. It records traffic flow data, number of connections initiated to and from a single client and sudden increases in traffic through specific ports or protocols. For Mail-trap, the protocol of interest is SMTP and the port is 25. Gen-prof is essentially a packet sniffer with a simple analysis engine attached to a reporting module. The reporting module has a bidirectional communication with the Knowledge Manager. As described earlier, very little state is maintained in Gen-prof and it keeps accounting information about each machine on its network over a user-specified window. The sensor architecture is not invoked for inbound messages since our experience with incoming email-borne malware as well as spam suggests that incoming behavior does not have any distinctive behavior.

A behavioral approach to monitoring email traffic in the network works with a high degree of accuracy since malicious email activity is typically noisy and is statistically different from normal traffic. Such an approach however provides little user level contextual information as to what triggers the abnormal activity. In order to address this particular problem, we Net-man, a host based sensor capable of providing deeper contextual information about a particular network behavior.

**Net-man** is a light-weight program which is installed on client machines to essentially track processes that spawn network activity. This information can be used to create a better picture of the overall threat. Our current implementation of Net-man only focuses on network activity. Most viruses and worms come with malicious payloads that install components of themselves on the infected machines, open up
back-doors as well as infecting files and network shares. A more advanced sensor can easily be created to include monitoring of the files that are modified by a process as well as any registry modifications. While we do not provide this functionality, we do however note that advanced process monitoring is relatively easy to implement on unix platforms through the use of interposition agents. We explore a simulation based approach of an advanced host based sensor in chapter 7. There are many full featured commercial as well as open-source solutions available for host monitoring and we felt the effort required to implement a cross-platform solution would have been orthogonal to our thesis.

Net-man logs port, ip address and process level information for each network activity. Logs are not kept indefinitely and are cycled every 20 mins keeping the overall log-file size low. Our initial design included a complete sand-boxed email client program to open email traffic. Every system call made on behalf of the email program could be tracked using interposition agents. However, we chose to go with a lightweight sensor which would be easier to install and less intrusive. The Knowledge Manager can specifically query Net-man for a particular port or IP address data. The Net-man helps add context to the anomaly behavior provided by the other three sensors.

3.4 Enforcers

As discussed earlier, the enforcers implement the security policy as handed down by the Knowledge Manager. The security policy is specified using a security level SLA between the domain and the end-users. However for the purpose of this implementation, we do not drop or delete any email traffic, instead, Mail-trap simply marks email activity as suspicious at the email server. This policy is similar to most industry spam fighting techniques and is completely non-intrusive and yet capable of identifying suspicious email activity.
Two separate policy enforcers implement the email security policy. The first sensor, installed at the mail server is responsible for marking emails as suspicious and putting them in separate folders. We implemented this functionality using the Content Management API on ‘Sendmail’. This API enables third party software to access email messages as they are being processed by the MTA. This allows our Mail-trap filter to add a header to the email message marking it as suspicious. That header can be further used to assign the incoming email traffic to specific folders by individual users. A second type of enforcer is needed in order to temporarily block all SMTP traffic emanating from a host. This functionality is implemented using a modified open-source packet-filtering firewall on a Linux box.

The placement of sensors and enforcers inside a domain is very important. A strategically designed sensor and enforcer architecture can ensure all SMTP traffic is observable at the sensors and can be blocked by the enforcers. This helps ensure any misbehavior is observable and stoppable.

3.5 Knowledge Manager

The Knowledge Manager lies is at the heart of the Mail-trap architecture. The novel idea behind our scheme is that each domain is no longer dependent on only local information in order to combat malware. While a domain might be able to detect the malicious behavior of an infection caused by an unknown email virus once the mail has been opened, it can rely on the experiences gained by others who might have experienced similar a similar email or might have already suffered an infection. This enables Mail-trap to mark unknown emails as being malicious without having ever seen the threat before.

The KM essentially enables aggregation and correlation of data between global and local sources in order to track misbehavior better. Domains collect information
about local misbehaving email traffic using the sensor architecture and share it across administrative boundaries with their trusted neighbors. This leads to the creation of a collective database of email-borne malware as well as infected hosts. All incoming traffic is checked against this database and appropriate actions taken.

Furthermore, downstream recipients of an email, that has already been identified as malware at the sending domain, can be informed about the suspicious email so that they can mark the emails accordingly and take appropriate actions. The good thing about our architecture is that other than marking email as suspicious, we perform no other intrusive activity. The sensor and enforcer architecture work completely in the background. The only installable component of Mail-trap is Net-man which is not directly involved in any detection activity and only serves to provide contextual information about the suspicious emails.

The forensics and alert sharing reduces the exposure of domains against new and unknown attacks. The data sharing component of the scheme, namely the KM and the CA, can easily work with with any such programs that may locally mark traffic as malicious or suspicious, examples include spamassassin, DNS blacklists etc. MASH as described in later sections is one such application that shows the results of sharing malware and spam related information across administrative domains. Our experimental results show that cooperation across domains increases the effectiveness of all such schemes.

Each KM instance has complete knowledge about all the sensors, enforcers and valid IP addresses that fall under its administrative control. This enables it to trace all alerts to the particular sensor and IP segment that they correspond to. For instance, the KM maintains a list of all IP addresses that have the net-man application running. It also keeps track of which sensor is responsible for what range of IP addresses. This enables the KM to ascertain if a local machine is spoofing an address. For instance, if their is a local traffic alert and the sensor that generated
while(1){
    report(IP time t){
        s = query_sensors_assigned(IP);
        e = query_enforcers_assigned(IP);
        block(IP);
        raw_data=Collect_local_info(s t±Δ)
        sig =generate_signature(raw_data);
        Update_DB(sig);
        fanout =generate_fanout(raw_data);
        rate_limit(IP port);
        Alert(e);
        Alert(fanout);
    }
}

Table 3.4: Mail-trap operation

the alert is not supposed to receive traffic from that particular IP address, it is a clear indication of IP spoofing.

Since the topological information associating IP addresses with sensors and enforcers is rarely changed after the initial design, it is kept as a pre-programmed association matrix at the KM. The actual Mail-trap application is available through a password protected web-interface which also enables an administrator to see a domain wide picture as well as enabling him to set thresholds and update window sizes etc on sensors. Through a well designed sensor architecture, the KM can immediately identify misbehavior to a particular network segment.

Mail-trap operations can be looked at as a four step process. The first step of the process is dependent on the various sensors to generate an Alert. The sensors monitor out-bound email traffic to detect misbehavior. The misbehavior thresholds correspond to average normal email activity behavior. We measure the mean and standard deviation of normal traffic behavior for each client using the sensor archi-

62
tecture. The sensor thresholds are arbitrarily set to a traffic behavior that exceeds the normal by more than 2.5 deviations. Once an alarm is generated at any of the sensors which translates to a client sending more than its threshold level of SMTP traffic, the sensor informs the KM of the misbehavior.

**Data Collection** This triggers off the second phase of data collection from the appropriate sensors. The Knowledge Manager operation is summarized in Table 3.4. The Mail-trap application triggers off the data collection process by locating sensors responsible for the particular IP address causing the alarm. Since SMTP is a connection oriented protocol that works in a lock-step fashion, the source IP addresses of traffic cannot be spoofed. Once the relevant sensors are determined, the application queries them in order to collect raw data regarding the threat. As mentioned earlier, Mail-prof maintains the headers of all the emails sent by a client for a 10 minute time slot. Once the KM is alerted to suspicious behavior coming from a particular machine, it requests the Mail-prof and the Edge-prof to provide all the collected headers for the particular machine to the Knowledge Manager. The Neman application, if installed on the machine is queried to provide the process level information to the KM. The data, once collected, is then passed onto the signature generation function.

As pointed out earlier, while the email header can be completely spoofed, including the **From:** field as well as the **Received:** headers, the IP address of the machine initiating the SMTP protocol cannot be spoofed, regardless of whether an email virus is using the local mail server or utilizing its own SMTP engine. Once the raw data is collected, the KM passes off control to the Mail-trap signature generation function. While the data is being collected, the KM instruments the local enforcer responsible for the particular IP address to deny SMTP traffic originating at that particular IP. This is a simple safeguard mechanism that helps prevent undue fan-out of the malicious email activity while the forensics data is collected and a
signature generated. It is very important to have a well placed enforcer architecture for precisely this reason. If the enforcer is not placed in the critical path between each machine and the Internet, an infected machine would be able to spread without any checks.

**Signature Generation** The Mail-trap application creates a generalized footprint of a threat through a fusion of the forensics data available though the different sensors. In our case, the raw data means all the data collected by the sensors over the collection window and includes

- email envelope headers from Mail-prof and Edge-prof,
- traffic aggregate statistics from Gen-prof
- network traffic statistics from Net-man, including process names, ports and protocol information

Existing malware signature generation schemes have shared one common flaw: they look for some static element which is common to existing or known malware [19]. The attackers simply end up removing or changing this element and the problem continues. We observed the same phenomenon with our sand-box virus experiments. Apart from the older Melissa and Iloveyou viruses, there was very little static content in the actual emails that the virus used to spread. Each of the other viruses came with a whole set of subject and message bodies that it randomly picked from.

While Mail-trap does not rely on any static content to detect misbehaving traffic, our signature generation scheme tries to incorporate contextual and behavioral information into the signature in order to supplement static signatures. Aggregating data across administrative boundaries also helps create a more complete picture. A simple set of experiments with Mydoom illustrates the point clearly. We conducted 5 different experiments with Mydoom worm in our sand-boxed machine. Each time
static const struct {
    char pref;
    const char *subj;
} subjs[] = {
    { 12, "" }, // 
    { 35, "grfg" }, // test
    { 35, "uv" }, // hi
    { 35, "uryyb" }, // hello
    { 8, "Znvy Qryvirel Flfgrz" }, // Mail Delivery System
    { 8, "Znvy Genafnpvgba Snvyrq" }, // Mail Transaction Failed
    { 8, "Freire Ercebeg" }, // Server Report
    { 10, "Fgnghf" }, // Status
    { 10, "Reeebe" }, // Error
    { 0, "" }
};

Table 3.5: MyDoom Subject selection, rot13

A different set of subjects was chosen for the emails sent out by the virus. A fusion of all 5 results however revealed the complete set of 8 subject lines that Mydoom emails can take. Static analysis of Mydoom source code confirmed the observation. The relevant source code shown in table 3.5 shows the encoded subject lines that are randomly chosen for emails sent by mydoom. A domain relying on only local observations would be unable to get a complete picture of the underlying worm. In any case, if the worm chooses completely random subject lines and bodies, pretty much any analysis on the email message would be rendered ineffective.

We adopt a novel signature scheme that helps aggregate behavioral signatures observed across administrative domains and helps display information in a human read-able form. Our signature scheme is a modification of basic concepts introduced by the Common Language Project [34]. The detailed behavioral signature scheme is described in chapter 7. The Common Language Project describes an attack as a series of steps taken by an attacker to achieve an unauthorized result. We represent the malicious behavior observed at each domain in this format. Domains exchange this information between trusted domains which leads to the creation of a better signature of the underlying threat. In the absence of a complete analysis of the
worm behavior, the aggregated view of the threat across multiple domains provides the most information in order to tackle the problem during the cure generation window.

Before an attack signature is generated, the KM needs to separate malicious email headers from benign headers that it receives from the various sensors around the network. While the KM only looks at the emails sent over a specific time window, the problem arises in removing the false positives from the corpus of data collected by the sensors. Such a situation would arise if a user is sending email while a malicious program is using the machine to send malicious mails in the background. While advanced clustering algorithms have been suggested to separate ham from the malicious emails. We apply a simple heuristics approach to this problem since we observe there is no exact solution to this problem in the face of advanced polymorphic worms. We assign a score to each email in the collected pool based on the following criteria. Meeting each of the criteria counts adds a score of 1 to the overall score of an email. Each email can end up with a potential score of between 0 and 5. Any email with a total score that exceeds a particular threshold is labelled as malicious.

- Any email with a header with an external IP address in the Received: entry in the header gets a score of 1 since there ought to be only one entry in the received header for emails originating in a domain. As explained earlier, each MTA adds its header to an email and the presence of outside IP addresses in the header of an email that has just originated raises an immediate alarm. Exceptions can be added for unix style forwarded email using whitelists where we can specifically allow email coming from local SMTP servers.

- The total score of all emails that bypass the SMTP server gets incremented by 1. While this might seem overly restrictive, most ISP’s already block outbound
SMTP traffic from their users. Exceptions to the rule can be built into the system with special-case service level agreements for particular machines.

- Since SMTP is a connection oriented protocol, IP spoofing is not possible. While a malicious email may be spoofed in its entirety, the IP address of the machine sending the message cannot be spoofed. An email is added to the malicious list if the reverse DNS lookup of the SMTP client does not match with its advertised name.

- All emails with attachments of type exe, BAT, COM, CMD, CPL, HTA, JS, JSE, LNK, MSI, PIF, REG, SCF, SCR, VB, VBE, VBS, WS, WSC, WSF, WSH are scored as positive. While a file having an attachment by itself is not necessarily a sure sign of malware, we chose to mark an email as suspicious on this metric simply because we considered the probability of such attachments being sent as part of legitimate traffic during a burst of email traffic as low.

- The KM calculates the average file size of the attachment and message body. All the emails with message sizes and file sizes that fall within 1 deviation of the average are considered suspicious and their score gets incremented by 1. While older worms can with static file sizes, newer ones are coming with increasingly random file sizes per iteration.

  We chose 2 as the cut-off point for labeling an email as malicious. The choice of assigning equal weights to each of the above mentioned criteria was however completely arbitrary.

3.5.1 Accuracy of the scoring technique

The success of the Mail-trap forensics sharing scheme is directly correlated with the quality of the evidence shared by the scheme. While the sensors can be trusted
to gather the data accurately, we conducted a set of experiments to evaluate the performance of the evidence generation scheme adopted by Mail-trap. We rely on a heuristic based scoring scheme to separate malicious emails from normal emails once an alert is detected by the sensors. As a trivial case, we chose the 4 year sent email data from one of our users as the test data set. While running this experiment, we assumed that one of the sensors had already alerted the KM with anomalous network traffic. We then evaluated the entire test-dataset against a 10 minute infected email corpus produced by the modified Mydoom worm. We measured the false positive rate as the probability of including a non-viral email as part of the viral set. Using the counting scheme alone, the false positive rate for the data-set was 0. We note that this was possible because Mydoom attachment size averages to $22K$ per infection.

We can make a reasonable argument to include the first line from the MIME attachment while differentiating the good emails from the bad. However, we chose not to perform any statistical analysis on the subject and message body data. Spam fighting techniques have shown that email body content can be used to detect malware with high success rates. While we note that similar schemes can be adopted here to remove false positives, our decision was based on our analysis of worm and viral code where we deduced that completely randomizing all the text in an email message as well as headers was essentially trivial. If every email is distinctly different from all the other emails in the detection window as recorded by the Mail-prof, there would be no way of distinguishing the malicious email from good emails apart from the anomalous network behavior.

3.5.2 Signature

Once the malicious emails are identified, the KM creates a generalized signature of the threat. A signature for email-borne malware consists of three components,
• Source: The source of an email borne threat includes data collected from the email envelope header as well as the process, port and network traffic information from the Net-man. The process port and network traffic information adds contextual information to the overall picture where we can actually track the processes responsible for the anomalous traffic. The attack data includes the IP address of the machine suspected of sending malicious traffic. The KM queries the Net-man on that machine to find out the process and port involved in sending the offending traffic. If net-man is not available on a machine, Gen-prof is used to provide contextual information to a lesser degree about the traffic. While it is unable to provide process level information, it can still provide IP and port level information about misbehaving traffic.

• Data: The most important part of the signature is the actual malicious emails. All the raw data that leads to an alert being generated forms a part of the attack signature. This raw data includes the overall envelope headers as logged by the Mail-prof and Edge-prof, file attachment names, types, size and related statistics. We do not use any data collected from the actual body of the emails since the privacy issues generated by sharing such forensics data across administrative domains are complicated to handle. That information also forms part of the attack body. As we described earlier, all the emails that score above 2 in our scoring algorithm are counted as malicious and form part of the signature.

• Target: The final component of the signature is the fan-out. The fan-out includes recipients of all the emails that were sent out before an alert was generated and the email was labeled as malicious. All the emails whose headers are included in the signature end up in the target. This information does not need to be shared with anyone else. The KM however needs this information
Once a signature is generated, the KMengine transfers the signature to the local signature repository where the signature is indexed and stored. The behavioral signature for ILoveYou and MyDoom from our sand-box experiments are shown in figures 3.6 and 3.7. Since the signatures were generated over a five minute time window, they captured a limited view of the worm epidemic. The attachment file size was constant for ILoveYou but MyDoom showed a variable file size averaging close to 22K.
3.5.3 Cooperation between trusted peers

One of the basic arguments put forward by our thesis is that cooperation between domains enables better detection as well as better response to malware traffic. In this section we describe the sharing mechanism as well as the data that is shared between domains. Each domain has to make two important decisions when it wants to share data with others. The first question is who to share with? and the second question is what to share?

Who: Mail-trap answers the first question by utilizing the trust based community model described in chapter 5. Our scheme utilizes the Paranoid secure group mechanisms to communicate across administrative domains and to share forensics data. The Paranoid scheme enables a domain to grant read/write access to people outside the domain without compromising on either security or privacy. Each domains decides which other domain it wants to share its forensics data with based on their past interactions. This decision can either be based on the trust mechanism or on any other metric. Examples include proximity, both geographical and topological and similarity of profiles etc.

Once trust is established, Paranoid group semantics enable a domain to grant read/write access to anyone else outside a domain without the need to share a group secret. Data that is to be shared is kept encrypted with the local domain. While encrypting the data limits the use to domains which have access to security keys, the Paranoid group semantics limit access to only those domains that are trusted.

What: The question of what to share is answered through the Mail-trap application. Mail-trap holds each domain responsible for misbehavior coming from its domain. Once the signature for a threat has been generated, each domain is responsible for alerting the recipients of malicious emails that triggered off the alarm. The fan-out calculated as part of an attack signature earlier identifies the recipients of
malware originating from a domain. Since SMTP is a connection oriented protocol, the recipient information collected by a domain is completely accurate. Once the signature is generated, the KM contacts its corresponding KM in each of the downstream domains that receive the email. If the recipient domain does not have a KM, the entry is simply ignored.

We assume that the KM use a pre-set port to establish a bi-directional connection with peers. Once contact is established, the KM passes on the attack signature to the other domain. Each recipient domain can decide what steps it wants to take there onwards. The attack signature includes the originating IP address as well as process that sent the malicious activity and the header of the offending email. In our current scheme, each domain can mark its incoming traffic appropriately based on this data. If the remote domain trusts the alerting domain, it also assimilates the signature information in its local database, automatically giving it immunity against an attack that it has not seen inside its administrative boundaries.

The actual email can be marked suspicious either at the server using a milter, or using a Mail delivery agent such as procmail or a specialized email client. We do not get into the details of the email marking mechanism since each domain handles its incoming emails differently and at multiple levels.

The KM further contacts all the domains that are in its trusted group and alerts them about the new threat along with passing on the signature. A secondary sharing of forensics data happens through the use of the Collaboration Agent. The Collaboration Agent maintains an Import/Export matrix along with the frequencies with which it shares the data with its trusted neighbors. Once a signature is generated, Mail-trap shares the forensics data it has collected with domains that have subscribed to its data.

Mail-trap is specifically targeted at zero-day threats during the window of vulnerability while the exact signature of a threat is not available. Once a new threat
has been analyzed properly and a signature for it is generated, the problem reduces to keeping anti-virus and worm programs properly updated. Most worms have been able to generated massive amounts of malicious traffic in this time period. Our approach lets us retard the growth of a pathogen during this time period by increasing the overall awareness about a new threat. As discussed earlier, each domain observes a growing threat from its unique perspective and creates a signature that best fits the symptoms it observes. A union of these signatures is a much better representation of the overall pathogen. This is specially true in the case of polymorphic worms. Melissa and Iloveyou worms were easier to identify during our experiments specially because the worm contents, the subject, body, headers and attachments names etc were static across multiple infections. In the case of a massively polymorphic unknown worm, the union of observations gives a better picture of the pathogen. This picture can be used to retard the further propagation of the worm much more effectively. Our simulation based experiments for polymorphic multi-modal worms are discussed in detail in chapter 7.

3.5.4 Traffic Policing

Mail-trap has two types of policy enforcers to implement the email security policy. The first enforcer is brought into play immediately after malicious email activity is detected in the form of an alert. This enforcer enables the Knowledge Manager to temporarily block or rate-limit all SMTP traffic emanating from a particular IP address. We implemented this functionality using a modified open-source packet-filtering firewall on a Linux box. The same functionality can also be implemented as a host-based enforcer co-located with the Net-man sensor.

This provides an immediate stop-gap functionality that enables a domain to nip the misbehaving traffic in the real-time while the administrator and the user account abused is notified. Since email is not a time sensitive medium, false positives are
not a huge issue. An alert can be generated to notify an administrator to take the offending machine off-line and to further investigate the source of the problem. One of the problems in dealing with security incidents is the lack of forensics data regarding a security event. More often than not, security teams are left with data intentionally left behind by a pathogen. A smarter worm can clear its tracks completely even before an alarm is raised. By collecting network behavior of a machine from multiple vantage points, our scheme helps reduce that problem. The raw-data collected through the sensor architecture provides excellent forensics to track down the source of a security event.

Mail-trap uses a secondary enforcer at the mail server which is responsible for marking incoming emails as suspicious. This enforcer relies on the malware signatures in the local database in order to mark incoming emails. We implemented this enforcer as a ‘milter’ using the Content Management API on ‘Sendmail’. This API enables third party software to access email messages as they are being processed by a Mail Transfer Agent. This allows our Mail-trap filter to add a header to the email message marking it as suspicious. This special header similar to the one added by popular spam fighting programs can be further used to direct incoming email traffic to specific folders by individual users using a procmail like email filter. While an outbound email filter would also help reduce propagation rates of malware, we have not implemented such a scheme since most newer worms come with their own SMTP agents which bypass the server altogether. A simple scheme can however be adopted to use an SMTP relay with the same filter as the out-bound traffic.

3.5.5 Mail-trap Traffic Marking

Each domain maintains a local database of malicious email activity. This database is used to make decisions about incoming emails. Each domain can define its own security policy as to what weight it wants to give to the forensics or alerts shared
by another domain. Subsequently, each domain can make its own decision as to what signatures or alerts it wants to add to its local database. Once such an alert is included in the local database however, a domain potentially acquires immunity against attacks that it has not seen before. The strength of the immunity however depends on the overall strength and accuracy of the signature. In the case of worms with static content in the headers, the signature scheme works quite well. The worst case scenario occurs when an attack is completely polymorphic with completely randomly generated message headers and body. In such a situation, the virus signature simply becomes a black-list for the source IP addresses. Since the source IP address is considered compromised, all the email traffic emanating from it is considered malicious. We use a time-out mechanism that keeps an IP in the black-list for a day for each reported alert.

Each local database contains a list of all the IP addresses and the frequency with which they have been reported as having sent malicious email, we refer to this list as the black-list. This list contains misbehavior observed locally as well as through alerts from trusted neighbors. This serves as a local blacklist for incoming email messages. The local database also includes packet header information such as attachment names, attachment sizes. The process information included in the signature, while essential to create an overall picture of the threat, is not relevant for marking incoming email traffic.

Each incoming message is checked to see if it comes from a black-listed server. When it comes to checking the headers of incoming email against the local blacklist, we do not restrict ourselves to only the last MTA that delivers the email to the local MTA. The reason for this is the fact that while every other entry in the message can be spoofed, the presence of a known compromised server in the Received: headers of any email is a red flag. Therefore while the absence of black-listed IP’s does
not mean the message is clean, the presence certainly suggests spoofed headers or compromised mail servers.

Apart from marking the emails as suspicious, if the last hop MTA is not on the black-list and the packet is still marked as malware, the enforcer also forwards the email to the Knowledge Manager as coming from a potential ‘Suspect’. This is an indication of a mail server either being mis-configured (open relay etc), compromised or not following good security practices. We discuss this scenario in detail in the next section and use it to add more functionality to Mail-trap.

The Mail-trap filter adopts a simple scoring scheme very similar to the one described in section 3.5 in order to mark the remaining traffic as suspicious. Every email that is sent with an attachment with a similar name and size to the one in the database and a suspect subject line is considered suspect. While we do not keep the attachment content, we mark traffic based on attachment type, name and size statistics. Similarity for the subject line is calculated using the Levenshtein distance algorithm over the subject line of the message. The Levenshtein distance between two lines is given by the minimum number of operations needed to transform one string into the other. We chose an arbitrary value of 5 as the levenshtein distance needed to mark an email as suspicious.

We chose not to use any content based traffic marking of incoming traffic for two main reasons. Firstly, we observe that existing malware fighting schemes perform content based policing. Secondly, as noted earlier, creating a completely polymorphic worm with variable content, headers, body and attachments is trivial.

3.6 Observing misbehavior remotely

Mail-trap’s sensor architecture helps to identify and isolate bursts of mail traffic along with abnormal traffic patterns. Such traffic includes email viruses activity,
worms, as well as bulk spam traffic. Mail-prof and Edge-prof for instance are able to
detect the increase in mail traffic either through a central server or through a zombie
machine. Similarly, Net-man is able to identify and provide process level contextual
information regarding any such traffic. We do however observe that malicious traffic
that does not exceed the local mail thresholds is able to get through the system.

Mail-trap alert sharing mechanism allows cooperating peers to acquire immunity
against attacks that they have not seen. Mail-trap adopts a behavioral monitoring
approach to detect misbehavior in outbound email activity. In-bound traffic however
does not have any discernible traffic patterns. We also note that incoming malware
or spam traffic shows little or no behavioral signature unless it is opened up in sand-
boxed machines and shows visible signs of malware immediately upon activation. A
slight addition to the scheme however is used to identify machines that are either
mis-configured, or compromised.

The basic idea behind the approach remains the same: combining forensics from
various vantage points on the Internet enables domains to detect misbehavior that
is not large enough to raise local alarms. We explore similar issues related to misbe-
havior that is not detectable locally in creating a solution to counter DDoS attacks
in chapter 4).

We illustrate the point with an example. A smart program or worm would delib-
erately try to cover their tracks by being unobtrusive in their behavior. A machine
that is compromised by such a worm or bot-net would send out enough emails per
hour to be part of large attack network but would stay below the radar as as to
avoid creating traffic anomalies. An anomaly based scheme such as ours would be
unable to detect the misbehavior. The overall traffic generated at the originating
domain would be very small to raise local alerts. The only way to identify this mis-
behavior would be at the target domains, either through an automated process or
through user interaction. Once such misbehavior is identified, sharing of this information across administrative domains is the only way to enable identification of such ‘zombie’ machines.

One example of the automated process was described in the previous section where if the last hop MTA for an email was not on the blacklist and the email was still marked as malware, the enforcer forwarded the email to the Knowledge Manager as coming from a potential ‘Suspect’. This was considered as an indication of a mail server either being mis-configured (open relay etc), compromised or not following good security practices.

**Suspect List** The Knowledge Manager maintains a separate list of all the IP addresses that are labeled by Mail-trap as suspect. This would include mis-configured, compromised or non-cooperating machines. Each domain shares this list along with its local signature database at every sharing cycle. A domain that finds out one of its machines is on the ‘Suspect list’ can potentially identify ‘zombie’, hijacked or mis-configured machines. Receiving multiple such complaints about a particular IP from a distinct domain can force a KM to trigger further investigations on the machine.

### 3.6.1 Reputation

As described in chapter 5, the reputation of any domain in our scheme relies heavily on its observable behavior at each domain. The CA alerts each of the domains who have a representative on the ‘Malicious list’ with an alert message. This allows domains to better filter their own traffic as well as identify misuse of their resources. There is also an incentive for domains to respond to alerts from neighboring domains regarding misbehavior. A domain can also be made aware of misuse of its resources through down-stream domains which actually receive the offending traffic and pass an alert back upstream.
Any misbehaving domain that continues to let in malicious emails and spam either deliberately, as is the case with some off-shore AS’s, or inadvertently, through mis-configurations etc would continue to lose reputation in the community.

In the case of stealth or massively distributed attacks, the misbehavior at the source domain might not be large enough to be noted at the source as well. In case a domain continues to be non-cooperative and offending traffic continues to flow, its reputation would continue to suffer and it would eventually be cutoff from the community since all their email traffic would be marked malicious.

3.7 Implementation, Simulation, Experiments and results

In this section, we describe our simulation environment, evaluation methodology, implementation and results. A data sharing scheme such as ours truly benefits through greater cooperation between domains. So while we adapted, implemented and individually tested all the different components of the Mail-trap application, we decided to create a simulator to highlight the important performance benefits of our scheme under various simulated workloads and attack scenarios. We note here that Mail-trap relies on verifiable individual components that are already present in most networks and hence easily verifiable. For instance, the sensor and enforcer architectures use a combination of packet sniffer, flow re-constructors and programmable firewalls. The only other enforcer that utilized the content management API on sendmail was implemented and tested individually using an IP black-list on a local SMTP install.

We measured the performance of our detection and cooperation scheme along the following metrics.

- **Attack Detection**: This measures the performance of the overall detection scheme. There are three essential measures of verifying the attack detection in the case of emails. Firstly, a) **Detection Rate**: The fraction of all email
malware that are caught b) **False Positives**: are innocent emails that get mistakenly identified as malware. For most users, missing legitimate email is an order of magnitude worse than receiving spam.  
c) **False Negatives**: are malicious emails that manage to evade the detection scheme.

- **Cooperation effects** A number of metrics were used to measure the effect of cooperation on the overall scheme. Apart from the detection and false positive rates,  
a) **Vulnerable Population** This represents the % of population which is vulnerable to a particular attack. As the cooperation becomes more effective, the total percentage of vulnerable population would decrease as a result of acquired immunity.  
b) **Viral Emails sent** This is an estimate of the total number of emails as a result of a successful attack.

### 3.7.1 Local Misbehavior Detection

The first set of experiments was performed to answer two important questions. Firstly, does anomaly based misbehavior detection work? Secondly, what are the limitations of the approach?. We measured the performance of our sensor architecture using the test-bed environment we created earlier. We calculated the overall detection rates and the number of false positives introduced by the threshold based detection scheme. We chose the user account statistics displayed earlier to simulate normal email traffic on our testbed. Figure 3.8 shows that log of the number of emails sent per hour follows the normal distribution, with a mean of 1.01 and a standard deviation of 0.72. The normal activity showed in the graph corresponds to one of the user accounts we monitored over a four year period. This provided us with the benchmark normal traffic behavior to calibrate our Server-Prof and Edge-prof sensors. We did not however account for variation over the time of day in order to be consistent over multiple runs of the same experiment and used the traffic statistics shown earlier to drive the client traffic.
We used an arbitrary value of 3 deviations in excess of the average to signal an alert from the sensors. Therefore, the sensors raised an alarm if the number of emails sent from the user account in a particular time window went above 18. Our results showed a false positive rate of around 1% due to the choice of 2.5 deviations. While this does not necessarily represent a normal workload, it clearly distinguishes from the abnormal behavior, both in terms of number of emails sent as well as rate at which emails are sent.

As is also obvious from the graph, the normal activity follows cycles, roughly corresponding to the start of day, lunch hour and the end of the day.
cascading of email virus traffic however does not follow this trend. Email viral activity usually happens in response to a user opening an attachment or clicking a link to an infected server. High incubation periods therefore do not affect the accuracy of our detection scheme. The accuracy of the scheme however is dependent on the propagation rates of the virus in relation the propagation observation window size of the sensors.

We used a collection of live viruses for which we had source code available. We were able to limit the spray of outbound traffic by disabling the network propagation component of the code and redirecting the output to a file. Since both our mass-mailing viruses were so distinctly different from normal traffic, we achieved 100% detection rates with them. This was easily predictable since both the viruses were sending emails well in excess of the maximum emails observed from the user account. We ran our next set of experiments with modified versions of ILOveyou and Mydoom by adding incremental delays to the virus propagation function in order to simulate stealthier worms. The results were calculated using detection rates over 100 runs of 1 hour each for each of the propagation rates. The results of the experiments are summarized in table 3.6 shown below. The viral propagation rates shown in the table are in number of emails sent per minute.

Clearly, Mail-trap’s detection technique had a very high success rate with existing viruses. The detection rate however clearly dipped as the propagation rate of the virus started to go below a certain threshold. In our experiments, the cut-off propagation rate was close to 18 viral emails per hour. This corresponded to close to 3 deviations from the average email usage for the particular user account. In such a situation, the virus clearly was not spreading at epidemic rates and while the false positive rate was still low, there was no distinctive traffic pattern in the resulting network traffic.
A third virus (modified version of Mydoom) was injected into the client machine that divided its traffic evenly between the mail-server and its own web-server. The detection rate for this virus fell quicker as opposed to the same virus using just a single server for propagation. The clearly demonstrated the need for a central point of coordination such as the Knowledge Manager where each sensor logs its data periodically. Adding the KM to the scheme restored the detection rates to the the previous values. The sensors were able to detect a higher number of pathogens, even ones that were slipping past without the increased cooperation. However, the overall detection time increased due to the transfer of data between the sensors and the KM.

The results from our experiments highlighted a few important points. Firstly, high detection rates can be achieved through behavior monitoring of outbound network traffic. Smarter worms that try to model normal traffic patterns of email users would however be able to escape local misbehavior detection. We would potentially have to augment Mail-trap’s network based approach with a heuristic approach based on the email header content.
3.7.2 Simulation as a verification technique

One of the biggest problems with malware fighting techniques is the absence of a realistic testing environment. We felt that the local behavioral monitoring components of Mail-trap could evaluated sufficiently using the sand-boxed test client and server setup described earlier. Such an environment however, would be inadequate to evaluate the benefits of large scale cooperation allowed by the application. We therefore created a simulator to demonstrate the usefulness of the architecture and evaluate its benefits.

Simulation as a verification scheme has its own set of advantages and disadvantages. The biggest disadvantage of simulation is that real security applications cannot be tested on it. Trace based simulation however gives us the flexibility of testing individual components and scenarios that highlight the important aspects, effectiveness and limitations of our scheme. It is possible to model any parameter in a simulation. The disadvantage however is that time passes in discrete steps rather than being continuous. Simulated networks do not contain performance bottle-necks etc like bandwidth and latency constraints, routers firewalls and bridges.

We evaluated several different testing environments before we decided to create our own simulator. We used a Java based multi-agent programmable environment (Netlogo) to create a simulator in order to test the performance of our scheme under various simulated attacks. The trace data used to drive the simulator was derived from the live data collected as described earlier. Simulation as a verification scheme has its own set of advantages and disadvantages. The biggest disadvantage of simulation is that real security applications cannot be tested on it. However, the performance of our scheme depends on the inherent entropy in the data that Mail-trap shares between domains. The actual data that we used to run our simulations was either live data or data derived by observing the behavior of live malware in con-
trolled environments. We felt that the actual packet level simulation of the overall architecture would not add any value to our analysis.

Ideally we would have liked to have implemented the functionality on a live network but were unable to do so due to obvious accessibility, privacy and security concerns. Our implementation was dictated by the desire to keep the simulator and work-load as real-life as possible.

3.7.3 Simulator Experiments

Our simulator creates a set of domains each having a group of clients engaging in email activity. The domains are connected to each other in a random network topology. While we did not simulate an extremely large topology, we note the same principles apply even if the scale were much larger. We parameterized individual client behavior based on the email data corpus we collected. Individual sensors, enforcers, Knowledge Manager and Collaboration agents were modeled at each domain. A zero day infection was introduced into one of the clients and it spread through the simulator population. The sensors programmed into the simulator observed the traffic anomalies and raised appropriate alarms.

The simulator allows us to evaluate the performance of the collaborative forensics sharing under

- different population size
- Varying degrees of cooperation between independent domains
- Different worm propagation rates
- varying accuracy of the worm signature
- Degree of polymorphism displayed by the worm
- Different alert sharing policies
• Number of trusted neighbors of each domain

We conducted a series of experiments to evaluate the performance of our scheme in each of the above mentioned scenarios. We programmed our simulator to create a random topology of 100 domains each with 50 clients. We then modeled the email sending behavior of the individual clients on the email behavior we derived from our datasets. We then introduced a zero-day email based worm to the email system and observed the spread of the new pathogen. Since most email worms require some sort of human involvement to spread. We assumed 90% of the email using clients to be smart enough not to let the worm spread once they received it. In the case of Mydoom or similar worms, this would translate to not opening the file attachment. Later on, we relaxed this assumption to observe the epidemic spread of the worm under those conditions. The situation is also slightly different from real-life viruses since our simulation assumes homogeneity in the client population. In the real world, a virus may be targeted for the windows platform and not all population uses windows.

Figure 3.9 shows the spread of a worm infection under various propagation rates. Previous results for email based pathogens actually refer to similar behavior observed by existing worm pathogens. Such exponential growth rates are fairly accurate for worms that require little or no human involvement for spreading. Examples of such worms include Code Red etc. These results however are not very realistic for an email borne virus because each email user does not open every email immediately. Therefore, we programmed each client to have a randomly generated email checking frequency. The frequency values we assigned to each client were distributed equally from once an hour to once a day, following a standard normal distribution. Figure 3.10 displays the graph of the epidemic spread of the zero-day worm through the 100 domains and 5000 email users under various worm propagation rates with the new mailbox behavior. These sets of experiments served to validate our simulator model.
We conducted a separate set of experiments to measure the effect of the probability of opening the file. The previous set of experiments assumed that each user had a 10% probability to open the file attachment. The results for this set are summarized in figure 3.11.

3.7.4 Effect of Alert Sharing on propagation

Our next series of experiments involved measuring the performance of alert sharing on the overall propagation of the worm in our simulated environment. For this set of experiments, we assumed 100% cooperation between the users and universal deployment of the sensor and enforcer architecture. We relaxed these assumptions in
Figure 3.10: Epidemic spreading of the worm (Realistic mailbox behavior)

later experiments. We assumed zero-day worm as the starting point for the alert generation scheme. The strength of the alert sharing in real life depends on three important factors, the propagation vectors of the worm, the degree of polymorphism in the worm and effectiveness of the traffic marking scheme. Our simulated worm behavior had only one propagation vector. Assuming a perfect detection rate we evaluated the performance of the alert sharing scheme by calculating the number of immune nodes per time cycle. The number of immune nodes was taken as the nodes who received an alert regarding the worm. So along with infected, vulnerable, we now have a list of immune clients in the network that actually have the alert. The experiments assumed that the immune clients were no longer vulnerable to the worm. We assumed an infection rate of 40% for these experiments, Figures 3.12 3.13 represents the spread of the worm with and without the alert sharing for two different worm propagation rates. We modelled two different alert sharing scenario, in the first scenario, each domain immediately shared the alert with its set of friends after receiving it. In the second scenario, each friend shared the alert with its friends every 30 minutes. The results for both the scenarios are summarized in figure. While
the number of warned clients or immune clients in our scenario rose quite quickly and was quite handy in throttling worms spreading with propagation rates of 4 and 2, the number of infected machines rose quite significantly for higher worm propagation rates. On closer examination, this was dependent on the length of the signature generation window and the alert propagation policy adopted by individual domains. The dotted lines in figure 3.14 represent the same worm with immediate alert forwarding adopted by the domains. The results from the experiments were consistent across various network topologies as well as infection rates. Higher infection rates displayed a higher number of infections but the overall trends were very similar.

We modeled a signature generation time of 5 minutes in the above mentioned experiments and worms with higher propagation rates were able to send out a higher
number of infected emails during that time period. Mail-trap however does include a proc-mail filter that allows clients to screen emails already in the mailbox against known attacks. If a domain has already been warned about a worm, the actual number of infections would be lower than the figure shown in our simulations. This serves as an upper bound to the total number of infections. The simulator also assumes that an infected client continues to send out infected emails once it is infected. The use of enforcers can block this traffic as well. A next set of experiments was performed to measure the effect of cooperation on the overall traffic flooding the network. Typical

**Figure 3.12:** Effect of Alert Sharing on worm propagation (propagation rate 2)
Figure 3.13: Effect of Alert Sharing on worm propagation (propagation rate 4), different alert sharing policies

Worm outbreaks come with a high rate of traffic at servers and routers. Figure 3.15 represents the number of total attack emails reaching a server on average with and without the cooperative alert sharing. The results clearly suggest that the total attack traffic is considerably lower in case of alert cooperation between domains. This trend was visible even under partial vaccination, resulting from varying degrees of cooperation between domains.

3.7.5 Degree of cooperation

Our earlier experiments assumed 100% cooperation between the domains. This set of experiments was performed to measure the effect of the degree of cooperation in the system. We assumed earlier that everyone was participating in propagating alerts to their friends. The number of trusted neighbors of each node also affects the performance of the overall data sharing. Intuitively, the greater the degree of cooperation, infection rates would be lower.

We created a random topology of domains with each domain having a random set of trusted friends it shared its alerts and data with. We assumed immediate alert
propagation in these experiments. We evaluated the performance of the scheme under varying degrees of cooperation for different worm propagation rates. Non-cooperating worms, while able to use the alert themselves for local clients, did not forward alerts to neighbors. The probability of opening each infected email by each client was kept constant throughout the experiments in order remain consistent across various runs. The results from these experiments are summarized in Figure 3.16. Since the population size in our simulation was small, the actual infection rate was dominated by the probability of email checking. This accounts for the similar nature
Figure 3.15: Number of attack emails reaching the server

of the spread of the worm despite varying propagation rates. However, the effect of cooperation was quite evident. The number of infected machines for the most virulent worm was 40 times less with 100% cooperation when compared to 50% cooperating domains.

The important result to be highlighted here is that the significant percentage of the cooperating clients received warnings about the attack within minutes of the attack occurring. While the virus spread was increased due to the lack of cooperation and it took longer for the alert to reach the cooperating domains. Once the alert was received through different paths, the nodes were able to successfully prevent further infections.

Mail-trap’s traffic policing mechanism is only designed to mark in-coming email traffic. Our simulator is therefore designed to depict this behavior. Consequently, we only apply out-going filtering at the zero day domain. If similar mechanisms were to be deployed at other domains depicting malicious out-going behavior, the overall number of infections would be lower than the one predicted by our simulator.
Figure 3.16: Effect of degree of cooperation for different worm propagation rates

3.7.6 Accuracy of worm signature through collaboration

One of the most significant aspects of our simulator is the degree of accuracy of the signature created by Mail-trap. All our previous experiments assumed that getting an alert from a cooperating friend automatically lets us acquire immunity towards the new pathogen. This assumption is inaccurate in the face of worms that display a degree of polymorphism in their behavior. Such a worm would display different behavior on individual domains. Each domain would therefore have a possibly incomplete and unique perspective of the worm propagation based on what propagation behavior it observed. MyDoom for instance uses a variety of subject lines and file attachment names in its propagation. Some of the worms even induce variation in the payload sizes. A collaborative view of the threat through sharing of the worm data information creates a better view of the worm. Cooperation between domains in such a scenario becomes even more important.
In this set of experiments, we programmed the worm to display unique behavior at each domain and observed the effects of cooperation between domains on the overall spread of the epidemic. Each participating domain therefore created a unique signature of the worm depending on the data that it observed. We estimated the degree of polymorphism in the worm behavior by the accuracy of the worm signature generated by each domain. For this set of experiments, we assumed to accuracy of the worm signature as being inversely related to the degree of polymorphism in a worm behavior. For instance, if the accuracy of the signature is 10%, the degree of polymorphism or the differentiation in the worm would be proportional to 90%. For a 0% polymorphic worm, the degree of accuracy of the signature would be 100%, for a worm that showed 90% polymorphic behavior, the accuracy of the signature would be 10%.

As we noted earlier that while there was little or no polymorphism displayed by the worms in our sample set, the ability could be easily built into newer worms. The
Figure 3.18: Effect of data sharing for polymorphic worms for varying degrees of cooperation
results of the experiments to evaluate the degree of accuracy of the worm signature are shown in figure 3.17. We assume 100% cooperation between domains in this experiment and it shows that even for partial behavior observed by each domain, the overall number of infections is much lower than epidemic cascading.

While the inaccuracy of the worm signature greatly reduces the accuracy of the sharing scheme. In the worst case scenario, the scheme reduces to become a black-list of the IP addresses that are infected. We setup 5 different experiments to measure the epidemic spread of the worm, varying the level of cooperation between domains from 100% to 50%. The cooperating domains and the worm traffic were generated randomly so as not to bias the results. We also assumed a constant worm propagation rate across the experiments along with the infection probability. The results of the experiments are summarized in figure 3.18. The results show that even for 50% cooperation, Mail-trap was successful in slowing down the epidemic growth of the worm. Partial cooperation also means partial vaccination and despite that, the spread of the worm was significantly slower as compared to without Mail-trap.

3.8 Sharing SPAM related data

In this section we analyze forensics data sharing and its benefits in the context of spam or junk mail. Spam as described earlier, is the electronic equivalent of junk mail that has become a huge problem on the Internet. Conservative data collection estimates suggest that spam forms close to 90% of the total email traffic sent on the Internet. Anti-spam techniques vary from content based email-filtering schemes, to spamtraps, blacklists and greylists. Most spam fighting techniques rely on the fact that email threat activity across the Internet is highly similar and exploit this similarity to better detect misuse or abuse of network resources. While anti-spam technology has been quite successful in tracking and filtering spam, advertised success
rates close to 95% are not uncommon, spammers have responded by increasing the spam volume by orders of magnitude to make sure users get more spam despite anti-spam programs in action. Indeed, most antispam techniques have acted like pesticides that end up creating newer, resistant strains of bugs.

Since anti-spam techniques are quite well advertised, spammers and attackers know exactly what they are up against and have kept a step ahead by using this knowledge to get past filters. In fact tools have been developed that let spammers test whether their messages would get past successful anti-spam tools or not []. Spam blacklists and white-lists are similar attempts at tracking malicious servers through a central repository system. DNS based blacklists typically list IP addresses that send unsolicited bulk email to decoy accounts created specifically to attract such emails. The main idea behind the scheme is that spammers typically re-use servers and IP addresses to send and re-send millions of emails. The blacklists are a way to detect misbehavior coming from particular IP addresses in a centralized way. Good DNS based black lists such as spamhaus have shown performance results close to 80%. Spammers have continuously changed tactics in an attempt to keep ahead of the anti-spam industry and this result was evident in the data. Most malicious email users adopt a hit and run philosophy where they compromise a machine and abuse it extensively before moving onto a new machine by the time the IP starts getting listed on DNS blacklists etc. Since the lists are publicly available, they also allow malicious users to detect if they are vulnerable to detection.

We observe that despite these problems, forensics or data sharing is particularly well suited to detect spam and similar junk or malicious emails because, unlike other security events, malicious email across domains is similar and as long as there is entropy in the data, there are gains to be made by sharing information across administrative boundaries.
3.8.1 Design

We designed, Mash (Malware Sharing), as a stand-alone application to enable sharing of spam blacklists across administrative boundaries. The application allows domains to share their email data with anyone across the Internet without fear of security or compromise. Data is shared in an encrypted format. Access to data is through the use of cryptographic capabilities so information cannot be abused by malicious users.

MASH is built on top of the Foresight architecture and utilizes the Paranoid group semantics to create trust relationships. Trust is reflected in Read-Access rights to the data collected by each domain. MASH relies on the combined efforts of its users to collectively label malicious emails. Each participation user marks emails that they receive as malware, spam or junk. A small light-weight application capable of transferring user marked emails to the KM is installed by each participating user. Each user initializes the application by noting the location of their ‘Suspicious’ and/or spam folders at install time. Each participating user is expected to mark suspect emails by saving them to these folders. MASH ensures that emails that are marked as malicious are passed onto the Knowledge Manager at regular intervals.

The Knowledge Manager uses these marked emails to create a list of machines that send malicious traffic as marked by the users of a domain. We refer to this as the Malicious list and it acts as the collective memory of a domain. While email content based approaches are quite popular in the malware detection industry, spam or junk mail suffers from the same problem as other malware traffic. Everything in the packet headers can be spoofed, except for the last-hop SMTP server. We do not perform any content based classification on this traffic instead, we focus solely on the last-hop SMTP servers. We later show in our experiments that using MASH in combination with existing content based schemes improves the performance of both.
The accuracy of such a scheme relies heavily on the ability to mark the traffic responsibly. In a possible scenario, a malicious user could potentially spoil the whole scheme by falsely labeling good emails as bad. The use of non-repudiable digital signatures however allows trace-back of recommendations and lowers the probability of abuse by deterring users from casually labeling emails. We also adopt a simple counting scheme to lower the probability of malicious users corrupting the local ‘Malicious List’. Each entry in the ‘Malicious List’ holds a count of the number of distinct users that have labeled an email coming from this domain as malicious. We started off with a two strike policy with regards to entry in the malicious list. An IP address is considered compromised if it is marked by three or more individual users as having sent malicious emails. During the course of our experiments we varied this value to see its effects on the overall performance of the scheme in order to find an optimal value.

The ‘Malicious List’ is used to mark incoming traffic. The application only relies on the last-hop router to create the Malicious list. This approach might seem overly conservative however, it greatly reduces the number of false positives in the detection. MASH requires only one enforcer in order to mark incoming traffic. We implemented this enforcer as a ‘milter’ using the Content Management API on ‘Sendmail’. This allows the MASH filter to add a header to the email message marking it as suspicious in case it comes from an IP on the ‘Malicious list’.

Each incoming email header is compared to the ‘Malicious List’. The presence of a compromised IP address as the last hop Received: header of an email is considered sufficient to mark an email is suspect. The confidence value of this decision is based on the number of strikes in the ‘Malicious List’ for the particular IP. This approach allows us to use the local reputation of an IP address to label incoming email traffic. The presence of a hit in the Received headers of an email is a clear sign of a an email
either coming from a spam engine, coming from a ‘zombie’ or compromised machine on in the simplest scenario, coming from a mis-configured mail server.

3.8.2 Data sharing

Each participating domain periodically polls its neighbors to collects their ‘Malicious lists’. The Knowledge Manager relies on the Paranoid group mechanisms to determine who to share the data with. Trust relationships between domains work in exactly the same way as blacklists or grey-lists. The encrypted data is not shared with a domain that has a history of bad behavior. The frequency of sharing this information is determined using the data sharing matrix maintained at the Collaboration Agent. MASH utilizes the Paranoid Secure File system to share the list of offending IP addresses across administrative boundaries and our experiments show that such a collective effort to mark malicious emails increase the detection probability. It also provides a platform for domains to inform other domains of misuse or abuse of their resources that they might not be able to detect.

Each domain that receives the data another domain can decide the particular weight that it wants to give to the data by taking into account the trust value associated with the reporting domain. This leads to the creation of a list of well known misbehaving machines at each domain.

Each domain also sends out periodic alerts to all the domains who are part of its trusted group and have a machine in the ‘Malicious List’ of another domain. The application also attaches the headers of emails that cause the alert in the first place. This alert allows domains to potentially find out about misuse of their resources that does not raise local alarms. Since we hold each domain responsible for misbehavior that it transmits, each KM that receives an alert regarding a misbehavior from one of its machines is responsible to stop it. This ties in nicely with the trust mechanism described earlier. Since membership in the data sharing group is based upon
continued good behavior, there is an incentive therefore for all the domains in the community to promote responsible behavior. That includes detecting misbehavior, correcting any mis-configured servers so that they verify what they forward or relay.

We collected real-life spam and ham email data from 4 geographically diverse networks over a 4 year period and 10 individual email accounts. We augmented this data with publicly available spam email data. This consisted of a mix of spam, email viruses, URL re-directs etc. This traffic was used as a benchmark for evaluating the performance of our spam detection technique based on last hop servers. The overall size of the data exceeded half a million spam emails.

3.8.3 Local Correlation

The first set of experiments involved detecting similarity in the email data from one single email account over different time periods. For the purpose of this experiment, we used 5 different samples of email data spanning 2 week each from the same email account. The email data had already been labeled as being malicious or good and this information was used to verify our marking scheme. We correlated the data from the first 4 weeks to order to mark the email data for the final week of data collection. Each email was marked as being suspect if it came from a last-hop server that had already sent malicious email to this account. We measured the detection rate of our email marking scheme along with the number of false positives. Lastly, we compared our incoming mail against a years worth of malware data collection. We also experimented with a two strike policy for labeling an IP as malicious. Each offending IP was labeled as malicious if it sent more than two offending mails over the period. The results of the marking scheme are summarized in table 3.7. Our results show that email filtering based on just the last-hop IP addresses provides results comparable to publicly available blacklists.
The results of the email marking scheme clearly reflected the similarity across
time periods in the malicious emails being sent to the account. While the small
data sets detection rates were low, it must be pointed out that this detection rate
was only based on the last-hop smtp servers. DNS based blacklists display similar
performance results on independent data sets. Increased detection rates based on
just DNS based blacklists ends up increasing the false positives in the system. Most
commercial spam detection schemes rely on content based detection which we were
not performing. Our traffic marking scheme however can be used to complement
such approaches as described later. The year long data increased the detection rate
significantly although the number of false positives were increased as well. A two-
strike policy for detection reduced the number of false positives significantly. The
results show the significant entropy in the email data.

The first set of experiments explored similarities in the email data for a particular
account. For the next set of experiments, we extended the learning data-set to include
data from external accounts as well as administrative domains. We used email data
spanning 4 domains for this experiment which we labelled domain A,B,C and D.
Domains A and B were academic institutions while C was a software engineering

<table>
<thead>
<tr>
<th>Local Datasets</th>
<th>Detection</th>
<th>False+</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20%</td>
<td>2</td>
</tr>
<tr>
<td>1,2</td>
<td>37%</td>
<td>3</td>
</tr>
<tr>
<td>1,2,3</td>
<td>44%</td>
<td>3</td>
</tr>
<tr>
<td>1,2,3,4</td>
<td>52%</td>
<td>5</td>
</tr>
<tr>
<td>Year’s data</td>
<td>75%</td>
<td>12</td>
</tr>
<tr>
<td>Two-strike policy</td>
<td>62%</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sharing</th>
<th>Detection</th>
<th>False+</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>79%</td>
<td>5</td>
</tr>
<tr>
<td>A,B,C,D</td>
<td>82%</td>
<td>6</td>
</tr>
<tr>
<td>spamassassin</td>
<td>91%</td>
<td>2</td>
</tr>
<tr>
<td>spamassassin and Data</td>
<td>98%</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.7: Email Marking based on last-hop server
Dataset D collected from a publicly available spam email corpus. It was difficult to conduct experiments on the data since not all the data sets were of comparable sizes and time frames. Datasets A was over 14000 emails. Datasets B and C were 60000 while dataset D was 5000 emails. The results of our experiments are shown in the figure below. Our results showed that the timestamps on the data collected was equally important as size. While datasets A and C were collected concurrently, Dataset B was much older. We included it in our experiments to see if there was any correlation in the threat data across time as well. The results showed little or no correlation between datasets from a few years back and the current datasets.

We observe that the same idea can be used to improve the performance of existing spam detection schemes. Examples of such programs include popular anti-spam programs such as spam-assassin. The last two lines of the experiment show the performance results from running our test data against a standard install of spam assassin resulting in detection rates of close to 90%. We note here that the detection rate of the spam assassin software could potentially be increased by training it over the training data corpus. We however conducted a second set of experiments of running spamassassin along with our Malicious list. The results showed 98% detection rates over the test data.

3.9 Conclusions

We used Mail-trap to illustrate the benefit of a collaborative defense strategy towards a certain class of unknown pathogens. Mail-trap utilizes Foresight’s sensor and enforcer architecture to create a timely picture of a threat and to disseminate a signature among peering domains. The quicker identification helps to reduce the total number of infections in the window of vulnerability while the signature for a threat has not been generated. The main idea being that cooperative policing
and forensics outperform individual efforts in the face of newer and advanced network based attacks. We explored an anomaly based behavior monitoring scheme to detect local misuse of resources, combining the collective resources of the community enables us to detect stealthy misbehavior. It also enables domains to identify misbehavior which cannot be detected locally. The results obtained from our simulation environment were in line with our expectations. The anomaly based detection scheme however was not sensitive to very slow moving worms. The success of our approach in identifying polymorphic threats was proportional to the degree of cooperation between domains. However, even partial vaccination was successful in slowing down the epidemic growth of worms. We presented MASH, an application enabling the sharing of SPAM related data across administrative domains and used historical data to evaluate the efficiency of such a scheme. The results obtained were similar to industry standards in this area.
Cooperative policing during denial of service attacks

In this chapter, we present AMP, a service architecture for countering distributed denial of service (DDoS) attacks using a collaborative mechanism based on the Foresight architecture. AMP uses the dynamically configured network components provided by Foresight to perform traffic monitoring, filtering and detection of commonly known attacks. Once an attack is detected, it utilizes the cooperative policing mechanism to push filters as close to the source of attack traffic as possible. The scheme does not require universal deployment and is complementary to other approaches for countering DDoS attacks. The use of collaborative policing techniques however, reduces the overall attack traffic in the network leading to lower congestion. The scheme provides an incentive for domains to cooperate. Cooperating domains receive a preferential quality of service during attacks. In addition, AMP is economically viable, it can be offered as a service to the customers by service providers. We give a detailed design of our system which we have implemented on our simulation test-bed. Performance evaluation of our system shows that using our scheme we were able to recover close to 90% of throughput lost during simulated attack.
4.1 Denial of service attacks and botnets

Denial of service attacks come in a variety of forms and shapes. In general, in a denial of service (DoS) attack, a computer is rendered unable to respond to valid requests and traffic due to a large amount of malicious traffic. Typical denial of service attacks target the availability of a service by attacking the computational resources available to a machine. These resources include bandwidth, disk-space, CPU time, memory etc. Due to the flooding nature of the attack traffic, legitimate clients are unable to get through to the server. Denial of service attacks, hereafter referred to simply as “attacks”, can severely limit the ability of an organization to conduct normal business on the Internet, leading to heavy economic losses.

In a distributed denial of service (DDoS) attack, the flood of malicious traffic comes from a group of computers, ‘zombies’, operated by remote control by people who have stealthily gained access to them as shown in Figure 4.2. Such traffic is distributed across different entry points of the network making it harder to locate and shut off. This also allows the attacker, access to a much greater bandwidth than the victim. Locating the entry point of malicious traffic to the network is essential in countering such denial of service attacks.

If the entry point of the malicious traffic is identified, it becomes possible to stop the incoming malicious traffic, thus reducing the impact of the attack. In newer attacks, however, attack hosts do not directly attack a server. A distributed reflection based denial of service attack follows a two step process. First, the attack machine does not directly send traffic to the target machine. It spoofs the source address of a legitimate packet that it sends to many servers. The servers, respond to the packet assuming it to be a legitimate request. Thus a single attack machine can create a flood of packets to any target machine through reflection from legitimate servers. Most such attacks employ some form of amplification technique. The amplification
effect is based on the fact that small trigger packets can generate much larger response packets in turn. A DDoS attack which abuses open recursive Domain Name System (DNS) name servers is known as the DNS amplification attack. This is a modification of the classic reflected attack which uses a weakness in the extensions to the DNS protocol to massively amplify the attack traffic at a particular host. For instance, classic DNS based amplification attacks typically have an amplification factor of around 8. A 60 byte DNS query can generate a response of up to 512 bytes. An extended version of the DNS protocol, EDNS, lets the request initiator advertise a much larger UDP buffer size. A EDNS query with a larger buffer can fetch responses of over 4000 bytes translating to an amplification factor of over 60.

As discussed in previous chapters, the last few years have seen a huge rise in botnet activity. Botnets running on a collection of ‘zombie’ machines effectively give massive untraceable distributed computing power to their owners. Botnets have been hugely successful in launching and sustaining distributed denial of service attacks against network resources. The Storm botnet for instance, automatically launches a denial of service attack against any machine trying to catch its nodes. Having such massive untraceable computing power at the disposal of attackers opens the web to significant danger of attacks.

Several methods have been proposed to counter DoS and DDoS attacks. Traditional ways to counter DoS threats have used ingress filtering and firewalls[21]. The accuracy of such filtering schemes improves dramatically by pushing the filters closer to the actual source of the attacks. However, these methods have been largely ineffective in solving DDoS for several reasons. It is usually difficult to shut down an attack using a push-back like mechanism in a timely fashion because IP addresses on attack traffic can easily be spoofed. Secondly, universal deployment and close cooperation between competing service providers, administrative domains and jurisdictions is necessary for such schemes to be useful. Also, these methods may reduce
the availability of the server they are designed to protect. Another approach is to perform extensive caching of web server content so that the server is not rendered unavailable during an attack. With most servers providing dynamic content these days, this method is not very useful.

We note that most current DDoS detection techniques suffer from having a limited view of the global threat scenario. In the case of a large-scale distributed threat such as nodes in a botnet, the behavior observed at any one sub-net or domain might not be large enough to raise an alarm. Indeed with botnet activity observed in a recent botnet attack [39] none of the reflectors was sending more than 144kbps for an attack of approximately 4Gbps magnitude. With botnet sizes running into millions, it is not unrealistic to assume that a service provider would not be able to realize that it is part of an attack simply because the traffic that it generates would show only partial information that would not be enough to raise an alarm. Each node only generates a limited amount of traffic directed at the target but collectively the volume is enormous, enough to overwhelm resources at the target domain. The Internet community can combine its collective resources to have a pro-active, prevention and damage containment policy complementing a reactive, curative approach towards such security incidents.

By aggregating information from different domains, the system can potentially trace large scale DDoS attacks and mitigate them nearer to the sources. We use the Foresight architecture [101] to build cooperative mechanisms that are able to detect and reduce the impact of DDoS attacks. Attacks can also be reduced by making it more difficult for an attacker to get access to machines from which a DDoS attack can be launched. The use of the Foresight architecture allows us to quickly identify and inform the offending service provider of its resource misuse while maintaining connectivity.
In this chapter, we present AMP (Attack Mitigation and Prevention), a robust service architecture for countering distributed denial of service attacks. This architecture is designed to work with web servers that serve static as well as dynamic content. AMP has a viable business model where it can be deployed incrementally as a service by domains. Our implementation of AMP targets flooding-based attacks as well as reflected attacks, coming from fixed or spoofed IP addresses. A flood attack is one in which a storm of attack packets is sent to the web server, starving it of legitimate traffic. Typically, the attack packets are of a particular kind, e.g., a storm of UDP packets or TCP SYN packets. In this case, it is relatively easy to detect that an attack has occurred, based on a change in the traffic profile. This remains an interesting class of attacks though because it is hard to differentiate between legitimate and offending traffic.

We tested the performance of AMP using a host of simulated attacks on our packet level simulator. Using the Foresight architecture we were able to quickly detect and identify the malicious nodes that formed the attack. Remedial action taken at the originating domain helped stop the attacks closest to the sources and thus very little legitimate traffic was hurt. While we limited our policing action to egress filters, the real life policing action could involve anything from taking the attacking machines off-line till the source of the problem was patched, to simply dropping target bound packets.

4.2 Design Rationale

We assume domains or AS’s as the basic design entity which means that AS’s are assumed to be under a single ownership, trust and administrative control. Here onwards, we use the terms domains and AS interchangeably. Client web servers connect to their respective domain, which provide them connectivity with the rest
of the Internet. The routers on the edge of an AS are called edge routers. All the traffic entering a domain passes through these routers (see figure 4.1).

The capability to detect malicious traffic in the network can be placed at various locations. One option is to place the detection capability in the core of the network. However, this is undesirable due to the large volumes of traffic involved. Per packet processing in the core of the network is not feasible. The other extreme option is to put all the detection and filtering in the webserver. This creates additional CPU load on a web server that is already likely to be loaded, especially if it serves dynamic content. Architecture C remedies this by moving the detection functionality to another machine within the sub-net. Another option, labelled architecture D, is to have a detection machine connected to the edge router within the domain network,
as depicted in Figure 4.1.

Architecture D has several advantages over the other options described. Firstly, in a DDoS attack, the malicious traffic might enter the a domain through multiple entry points. Filtering the attack traffic out at these entry points is desirable because it prevents the aggregation of malicious traffic at the egress edge router corresponding to the web server’s subnet. Placing the detector within the service provider’s network, as in architecture D, allows for cooperation between different detectors in the domains network to filter out malicious traffic at all entry points. Secondly, consider the scenario in which the web server has a traffic contract with the domain that limits the amount of traffic it receives. In this scenario, it is more attractive to place the detector before the traffic contract is enforced rather than after it. Thirdly, in flood attacks, it is possible for the web server’s entire subnet to be affected by attack traffic directed towards it. Filtering outside the subnet helps minimize the impact of the attack on other machines in the subnet. Finally, architecture D has the added benefit of placing the responsibility of upgrading and maintaining the system at the service provider’s end, without which, every web server would need to maintain its own detector. The disadvantage of architecture D is that placing the detector in the domain’s network might slow down the traffic going to the web server’s subnet. In summary, architecture D presents the most suitable option as it places functionality outside the core, minimizes load on the web server and allows for cooperation between edge routers in a domains network. This forms part of the configuration used by AMP.

AMP architecture essentially builds on top of the Foresight architecture, using the dynamically configured network components to perform traffic monitoring, filtering and detection of commonly known attacks. Alerts and forensics are shared across administrative boundaries using the Paranoid [102] group sharing architecture. This
allows for effective identification and isolation of misbehaving machines even under reflected or spoofed attacks.

Foresight promotes the concept of a trusted community collaborating with each other to achieve common goals. Domains take self-policing actions on egress routers based on the information they receive from other collaboration Agents. The information sharing architecture also allows each provider to view a global picture of the attacks on other domains. A domain might not be able to realize that it is part of an attack because the traffic that it generates might not be large enough to raise local alarms. However, an alert message passed on by the attacked domain can make a domain realize misuse of its resources. Remedial action taken at the providers also stops the attacks closest to the sources and thus very little legitimate traffic is hurt. The self corrective action can involve anything from taking the zombie machines off-line till the source of the problem is patched, to simply dropping target bound packets.

When a domain monitors another domain’s interaction with itself and its ‘friends’ at an edge router, it gains a rich set of information. Security policy and trust can be stated in terms of this accounting data. Priorities of service and traffic shaping and policing can be defined in terms of service received in the past from those domains. The overall Foresight architecture induces a desirable behavior from the participants by discouraging a domain from performing actions that are detrimental to other domains. Results through simulated experiments show a marked decrease in the amount of legitimate traffic being dropped due to packet filtering since filters can be pushed closer to the sources of attacks.

4.2.1 Solution Overview

The edge routers used in AMP need to be capable of dynamically installing packet classification filters based on packet headers. This allows a domain to install various
Figure 4.2: Distributed denial of service attack
traffic conditioning entities that allow it to dynamically change the way the traffic is being treated. This functionality has been proposed for the next generation of routers to provide, for example, differentiated services [4]. The AMP architecture consists of a Knowledge Manager, edge routers, detectors and a host-based sensor called Server-prof. Consider the setup show in architecture D in figure, 4.1. For notational convenience, from here onward, we denote the edge router by E, the detector by D, the Knowledge Manager as KM and the web server (that has subscribed for the service) by W.

The edge router samples the stream of packets heading for the webserver and forwards duplicates of the sampled packets to a “detector” machine which is assigned to that edge router. The detector collects these samples and tries to detect attacks on the subscribed web servers using the samples. The detection algorithm is described in Section 4.3.4. If the detector suspects an attack on a web server, it contacts the KM to verify if indeed there is an attack. The KM maintains the most recent performance data on the health of the server through its interaction with a server based sensor called Server-prof. It also maintains a bidirectional communication channel with all the sensors and enforcers. If the performance data shows significant performance degradation at the server the KM confirms an attack. If everything looks normal as far as performance is concerned, the KM assumes that there is none. If the KM is unable to get a response from the server, it assumes that the server is under attack. The server health data serves the purpose of an external “heartbeat” monitoring of the server’s operation.

A specialized sensor called Server-prof monitors the health of the web server itself and generates an alert if the performance degrades beyond a certain threshold. The health of the server is deducted from performance variables such as CPU utilization, memory, buffer and disk usage. The server health data is periodically logged onto the Knowledge Manager. The KM in turn uses this data to make judgements
about the health of a server. If the performance indicators cross a certain threshold, the KM assumes the server is under attack. This is a very simplistic use of the Server-prof sensor for a very specific functionality. A potentially more advanced role for Server-prof is explained in chapter 7.

When the KM decides that the server is under attack, it installs the appropriate filter on the edge router E for filtering the incoming traffic. The KM also informs its neighboring domains about the attack and requests for policing the traffic before it enters the attacked domain. Finally, if the sensors detect that the attack is no longer underway, the KM can alert the edge-routers and the neighbors to uninstall the filters. The router can then resume normal traffic processing.

In addition to sharing alerts and threat related data. Domains count on trusted partners to perform security related requests for each other. For example, domain A, under a denial of service attack can request peering domains to police suspicious traffic before it enters domain A. The response to such requests is observable at domain A and can be used to enhance or decrease a peering domain’s reputation.

This simple technique does not require universal deployment however the use of the Foresight collaboration architecture enables us to further fine tune the traffic filtering process. Once the KM is notified of an attack, it generates an attack signature and alerts the Collaboration Agent about the domains responsible for the misbehavior. The CA is responsible for disseminating the attack signature information within the trusted community. Each domain is responsible for misbehavior emanating from its users as well as its transit traffic. The immediate neighbors which are either part of an attack or are conduits for attack traffic can shape or police the attack specific traffic to ease the load on the target server and domain. In case the misbehaving traffic is local, the KM can instrument the appropriate enforcer to mitigate the threat. In the event that misbehavior is happening due to transit traffic, the Knowl-
edge Manager informs the Collaboration Agent of the source of the misbehavior and provides appropriate instrumentation at perimeter enforcers.

Once a domain has alerted the community about an attack on its servers and disseminated an attack signature, it becomes the responsibility of the neighboring domains to help out by filtering the traffic. Hop-by-hop warning (Figure 4.3) and policing can lead to the attacks being traced back to the sources of the offending traffic. The source of the offensive traffic however may or may not be a legitimate service. For instance, in reflected attacks, the real attack machines are not visible since the offending traffic either comes from zombie machines or a legitimate network service is being abused by spoofing IP addresses. We can isolate misbehaving domains and limit the effectiveness of an attack by observing the traffic patterns on local as well as transit traffic and applying the appropriate filtering.

AMP has a viable business model in which the service provider can offer this defense mechanism as a service to which web servers can subscribe. For each web server subscribing to this service, the service provider installs appropriate classifiers for identifying traffic for the web server at the respective immediate edge router, and assigns a detector machine to it. Service providers also have an incentive to cooperate since bilateral trust relationships between domains can lead to end-to-end trust relationships. This also helps detect zombie machines or mis-configured reflector machines and services in the network.

4.3 Design

The essential components of our application are deployed in three network entities: edge routers, detectors and subscribing web servers
Figure 4.3: Partial Filters enforced after alert disseminated
4.3.1 Edge Router

The edge routers used in our scheme have the added functionality of being able to dynamically configure the data plane. Our proposed routers differ from an ordinary router in two ways. Firstly, they filter and randomly sample incoming traffic to subscribing web servers. Second, they can reroute the sampled traffic to the detector machine D. The sampling module samples all incoming packets destined for the web server according to an exponential distribution on the inter packet interval. Exponential sampling is known to accurately approximate the real traffic profile [75]. Every sampled packet is duplicated, one copy is sent to the reroute module and the other to W.

4.3.2 Server-prof

Each participating web server installs a lightweight verification module that monitors system health statistics and responds to the KM requests, we refer to this as the Server-prof. Server-prof is a light weight sensor implemented on the server in order to raise a red flag in case of server performance degradation. This sensor is implemented as a daemon that looks for performance degradation symptoms such as high CPU utilization, resource exhaustion such as memory and buffers unavailable for legitimate traffic and packet queue backup leading to indiscriminate packet drop. Server-prof samples network usage, cpu, memory, disk usage and processes running on a server. HTTP services are not typically CPU-intensive, so the major slowdown in the performance indices is due to the flooding nature of the traffic which is arriving at the network interface as the traffic increases. The presence of a Server-prof red flag does not necessarily mean a denial of service attack, it however does prompt the Knowledge Manager to further investigate the server supplied traffic statistics to determine the presence of an attack. Server-prof has access to the server log-files and in the active data collection mode, it can provide information on services, processes.
and log files to the Knowledge Manager. Apart from keeping a forensics account of the log files etc at the Knowledge Manager, the Server-prof is ideally placed to help detect other significant server based attacks as discussed later.

Our Server-prof sensor polls the server data every 120 seconds so that the server is not overwhelmed by the data gathering activity. We measured the performance overhead of the host-based sensor Server-prof by stress testing our web server with and without the sensor installed. The overall impact on the server performance was negligible since as noted earlier, most of the performance data collection is already part of standard OS performance management packages. Since the communication between the KM and Sensor-prof is UDP based, there is little connection setup overhead for TCP. It shows that little performance overhead is incurred by the server monitoring. The penalty incurred is also acceptable given the fact that performance only starts being effected under extreme load conditions (rarely) and that Server-prof reduces the false positives rate of the edge-router detection mechanism significantly.

4.3.3 Detector Machine

The detector machine D performs three functions. It (1) detects an attack pattern in sampled packet data, (2) communicates with the KM to confirm the attack, (3) detects the end of an attack. We explain each of these functions in more detail later.

4.3.4 Detection Algorithm

The main idea behind the detection algorithm is that attack traffic can be differentiated from normal traffic. A related observation is that normal traffic follows predictable patterns and if we have an estimate of what normal traffic behavior is, we can determine if there is a potential attack situation. As noted earlier however, anomaly based detection techniques create their own set of false positive alerts.
We use a two step detection technique in order to reduce the total number of false positives.

A traffic anomaly occurs if the traffic pattern changes from the usual. This notion of attack however is weak since there is no known definition of normal traffic on the Internet. Traffic patterns vary by time of day as well as a lot of other reasons one of which may or may not be malicious network activity. At any rate, there is always the Internet background radiation that is present and does not necessarily translate to an active attack.

We based our detection scheme on traffic collected from live routers and servers. We used a combination of publicly available data, datasets generated at routers and web-servers under our control in order to arrive at a notion of ‘normal’ traffic behavior. We collected incoming traffic data at a medium sized router and webserver.
over a 2 week period. The dataset consists of over 50,000 requests and a total of over a 2 million packets and provides interesting insights into the underlying traffic patterns. The webserver in question was housing a commercial application and while the data showed considerable variation over time of day and the day of the week, there were significant patterns in the data that we exploited. We collected the number of packets received at the server over the given time period. We further broke down the data into per hour time slots and found out that the traffic statistics per hour, roughly followed the log-normal distribution. The lognormal and normal probability graphs for a peak traffic hour and an off-peak traffic hour are shown in

Figure 4.5: Server Traffic patterns (Peak hr), Lognormal Distribution
Figures 4.4, 4.5 and 4.6. The lognormal probability plot is a graphical technique for assessing whether or not the logarithm of a data set is approximately normally distributed. The points on this plot form a nearly linear pattern which indicates that the lognormal distribution is an excellent model for this data set. We collected the data for various different time slots and the trend was similar across different times of day. This formed the basis of our attack detection scheme at the edge-routers.

Our analysis of the dataset also suggested that the distribution of SYN arrival packets for non-attack traffic can also be modeled using the normal distribution. Our results were very similar to the results produced by other groups such as [67]. Their
Table 4.1: TCP traffic statistics: CAIDA

<table>
<thead>
<tr>
<th>Flags</th>
<th>% pkt</th>
<th>% byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
<td>2.57</td>
<td>0.20</td>
</tr>
<tr>
<td>RST</td>
<td>1.22</td>
<td>0.08</td>
</tr>
<tr>
<td>ACK</td>
<td>56.49</td>
<td>67.12</td>
</tr>
<tr>
<td>FIN</td>
<td>2.63</td>
<td>0.21</td>
</tr>
<tr>
<td>SYN-ACK</td>
<td>2.06</td>
<td>0.16</td>
</tr>
<tr>
<td>PUSH</td>
<td>24.10</td>
<td>27.40</td>
</tr>
<tr>
<td>FIN-PUSH</td>
<td>0.47</td>
<td>0.44</td>
</tr>
<tr>
<td>Sum</td>
<td>89.53</td>
<td>95.61</td>
</tr>
<tr>
<td>TCP</td>
<td>90.08</td>
<td>97.10</td>
</tr>
</tbody>
</table>

dataset contained over 1,900,000,000 packets at the border router of their domain and showed a normal distribution for the SYN arrival packets regardless of the time of day.

Another important source of traffic data that we utilized was the publicly available datasets collected by CAIDA. Table 4.1 shows TCP traffic statistics collected over a 2 hour window at the Seattle WA to San Jose CA SONET link of CAIDA’s backbone. We used the TCP performance statistics obtained from this data set in conjunction with our data to model normal traffic flow at the routers as well as the web server. With the normal behavior established through a sampling of the incoming traffic at the routers, any significant deviation from the normal was considered an attack.

While our data collection efforts were concentrated on TCP flows and packets, we note that we could similarly learn the traffic profile for various other transport layer protocols such as UDP and application layer protocols such as HTTP, FTP or SMTP etc. We showed in the previous chapter that SMTP traffic follows predictable patterns across different times of day. Once the network has learnt the normal traffic pattern, any deviations from the normal can suggest a possible attack. While any such anomaly detection scheme has its share of false positives, as we describe later, our scheme relies on a two-step detection process in order to increase accuracy and
reduce the number of overall false positives in the system.

Getting back to the detection algorithm, the detector receives sampled packets from the edge router and passes them off to a detection module (DM). The DM collects packets in a buffer of length $L$, as they come in, and runs an attack detection algorithm on this buffer. When the fraction of packets that match a particular packet type exceed a threshold value $T$, the detector signals an attack. If the total volume of traffic exceeds the average traffic size significantly, the detector signals an attack. Our attack detection scheme maintains the proportion $P$ of packets in its buffer that match the attack pattern. It also uses two threshold values, $U$ and $L$. The threshold values are derived from the observed network traffic during normal operation where $M$ specifies the mean number of packets and $L$ and $U$ signify two deviations from the mean. As shown in figure 4.7, if the current state is “no attack” and $P < U$, it moves to the “attack” state. If the current state is “attack” and $P < L$, it moves to the “no attack” state. We can largely eliminate router slowdown by delegating the detection task to a separate detector machine. False alarm rates are reduced by introducing the verification step that is explained in detail in the next section.
4.3.5 Verification

The verification step is performed by the KM in order to reduce the false alarm rate induced by the probabilistic nature of the detection algorithm. In this step, the KM tries to verify the presence of an attack by corroborating the evidence from another source. The other source in this case is the Server-prof sensor which is installed at the server.

In a possible scenario, the incoming traffic from all the edge routers combined can be large enough to swamp a server and yet be small enough at individual entry points to stay within the limits of the attack detection algorithm, or in an alternative scenario, the offending traffic can be coming from inside the network. The performance statistics collected by the Server-prof serve as a secondary line of defence in such a situation. If the server performance metrics go above a certain threshold, the server raises a red flag and alerts the KM prompting a finer grained checking on behalf of the server.

We setup a series of experiments against a publicly available web-server in order to measure its performance statistics against an increasing traffic load. Our results showed a direct correlation between the server performance statistics and the traffic reaching a server. The webserver that we used to derive web-server performance statistics was a Linux box running version 2.6.22.1–41.ftp with 1Gbyte Ram running Apache 2.2.4. We also made sure only the most necessary services were running on the machine while we were stress testing the server.

Server-prof works by polling system performance data at specified intervals. In order to establish the pattern of performance degradation under attack conditions, we set up experiments to measure the various performance statistics of our web server under an increasing request load. We simulated attack conditions by throwing load at the server at different level of concurrency while the server was randomly serving
a static data file of between 1 and $512K$ bytes. The choice of file size seems arbitrary however, according to recent studies[16], the average page size on the Internet is close to $312K$. The server performance data thus collected therefore provided a good estimate of the overall server performance under attack conditions. We note that while our stress testing involved static data content, with bigger file sizes and dynamic server content, the performance degradation would likely happen quicker and at lesser loads.

The request load was varied from 1 request per second to 50 requests per second directed at the server. The request load and response time statistics visible to the clients are summarized in Figure 4.8 and 4.9. As the traffic load increased beyond a certain threshold, the average response time visible to all the clients increased drastically.
Figure 4.9: Client statistics

The server side performance results during the same experiment are shown in Figure 4.10, 4.11 and 4.12. Figure 4.10 shows the CPU utilization over the course of the experiments. Beyond a certain threshold, the CPU utilization got to near 100%. This directly corresponded to the exact time in figure 4.8 when the throughput observed by the clients started decreasing steadily due to thrashing at the server and there was a sharp rise in the number of failed connections. Figure 4.11 plots the CPU utilization along with the number of page faults per second observed at the server. The number of page faults going above 200 also directly coincided with the server not being able to cope with the incoming traffic and near 100% CPU utilization. Two
other system statistics that imply resource consumption are interrupts and context switches.

The CPU performance statistics showed a marked degradation as the amount of traffic directed at the server started increasing. This behavior led to packet losses, increased response times and decreased throughput to the clients. The experiments clearly showed that as the number of packets reaching the server increased, the performance steadily degraded. This observation formed the basis of the our Server-prof red-flag alert.

Our experiments made the implicit assumption that the server was well provisioned to handle incoming traffic in the first place. The performance of the red-flag alert is dependent on how well provisioned the server is to handle average daily traffic. For a very ill-provisioned server for instance, the performance would degrade considerably despite the fact that HTTP is a typically low intensity protocol. We ran a separate set of experiments on a Linux box running version 2.4.18 – 14 with
128Mb Ram running Apache 2.0.40 as the web server. The results of CPU utilization suggested saturation at much lower workloads even in the absence of attack traffic. Assuming a server machine is capable of handling the average loads as well as bursty traffic gracefully, the server performance data is capable of reflecting the incoming traffic. If the incoming traffic exceeds the server packet handling capacity,
the performance data clearly degrades significantly.

Amp uses the Server-prof red-flag alert as part of a two step process to reduce the number of false positives in an anomaly based detection of denial of service attacks. The Knowledge Manager, upon receiving an alert from the edge-routers verifies the presence of an attack by checking the server performance statistics. Our results show that the two step detection scheme used in combination greatly increases the accuracy and rate of detection of the overall architecture. While both the detection steps have their own false positive rates, combining both the detection schemes in a two-step process increases the overall reliability of the scheme. Simple probability theory suggests theoretical verification of the same idea. We assume certain probabilities in order to make our argument.

Let the probability of an attack \( P(\text{Attack}) = .04 \).

The probability of false positives is \( P(\text{Detection} | \text{Attack}) = .1 \)

The probability of false negatives is \( P(\overline{\text{Detection}} | \text{Attack}) = .1 \).

Therefore the probability of an attack given a detection is give by the Bayes rule as \( P(\text{Attack} | \text{Detection}) = \)

\[
\frac{P(\text{Attack})P(\text{Detection} | \text{Attack})}{P(\text{Detection} | A)P(A) + P(\text{Detection} | A^c)P(A^c)} = .27
\]

This result is surprisingly low since while 4 out of 100 flows could belong to attack traffic (hence the .01 probability), 10 out of every 100 flows are false alarms. However, adding a second detection test to the scheme, which in our case is the red-flag alert at the server, we increase the overall chances of detection. Assuming the exact same false positive rates for the red-flag alert as well, \( [P(\text{Attack} | \text{Detection} \cup \text{Redflag}) = \)

\[
\frac{P(\text{Attack})P(\text{RedFlag} | \text{Attack})P(\text{Detection} | \text{Attack})}{P(\text{RedFlag} \cup \text{Detection})} = 0.79
\]

131
Despite the fact that the naive Bayes model assumes independence of the two detection tests, using two tests to verify an attack works much better than one might expect. Similar techniques are applied extensively to classify spam traffic based on word occurrences inside a document.

When the detector signals an attack, it sends a “verify attack” message to the KM. As described earlier, the KM keeps a track of the most recent server performance statistics for the server through the Server-prof interface. If the data at the KM is less than five minutes old, the KM can query the Server-prof to get fresh data. Each such request packet is not more than 50 bytes long. Thus, Server-prof is never be swamped by KM traffic. Server-prof replies to the KM with the most recent performance metrics for the server. If KM gets a reply within a specific timeout interval \( i \), it accepts Server-prof’s response. Otherwise, the KM assumes that server is under attack. In case the server performance seems ok, the detection cycle continues. However, if the attack is confirmed, KM moves the edge routers to the policing mode. The communication between server and the KM is done using UDP. One reason for using UDP instead of TCP is to eliminate the performance overhead of the TCP window mechanism. Also, this allows for sending multiple packets in a continuous stream and so chances of all of them getting dropped are lower than one TCP packet (or more based on TCP retransmissions).

4.3.6 Local Policing

Once an attack is identified, the KM informs the edge router to divert all server bound traffic to the detector. The KM also informs all the other detectors inside the service provider’s domain (near other edge routers) of the attack. At this point, all packets destined for the webserver are forwarded to the detector which must decide which of these it has to drop and which it will forward. It does this by following a probabilistic scheme. If the attack detection algorithm (described in figure 4.7) indicates that an
attack is still on, it randomly decides to drop the packet with probability $p$. Any traffic flow which exceeds its normal traffic profile as determined during normal server operation is dropped with the same probability $p$. Everything else is passed onto the server machine. The detector also has the dual role of collecting the packet headers and logging them off to a file so they can be picked up by the KM in order to create a signature for the attack.

4.3.7 Attack signature

Once an attack is identified, we cannot simply drop all target bound traffic since that would include legitimate traffic as well and doing such a thing would effectively have the same impact as a successful attack. The purpose of our attack signature however is to limit the effect of denial of service on legitimate traffic and to maintain operability. The KM is responsible for creating an attack signature that incorporates data available both at the server as well as the edge routers. The attack signature generation is complicated by two factors. Firstly, since the signature is generated on the fly, it is not 100% accurate. Secondly, IP address spoofing is trivial, hence the addresses on incoming traffic can be spoofed. However, the creation of a signature proves instrumental in enabling a server to maintain operations during an attack.

The primary component of the signature is the source IP addresses for the attack. There are three possibilities when it comes to deducing the attack source from the captured packets at the detector. Firstly, the source IP can be the attack machine itself. This situation is the easiest to handle. The traffic observed at the edge router is sampled in order to infer patterns from the traffic. The IP addresses with the highest traffic reaching the server is automatically selected to be part of the signature. A more likely scenario is legitimate traffic being reflected through legitimate servers that respond to spoofed network requests. Since the source IP address in this case is the IP address of a legitimate service, such packets are also easily distinguished from
normal traffic due to their frequency. In order to differentiate between legitimate and attack traffic, we apply a simple counting scheme to create an attack profile, all IP addresses sending more than 50 packets to the destination within a given time window are included in the signature.

We maintain average and deviation statistics at the detector regarding traffic size coming from an individual machine during a given time slot. Any IP address exceeding the threshold is included in the attack signature. The attack signature scheme is however limited by the fact that in the worst case scenario, almost all the packets could come from unique spoofed IP addresses. In that case, there would be no signature available since no IP address would cross the legitimate traffic threshold. In such a scenario, it is extremely difficult to distinguish an attack signature from the incoming traffic and a domain has to completely rely on its neighbors to perform egress filtering in order to maintain operation during an attack. The most tricky situation arises in the case of spoofed source IP addresses. A domain has no way to authenticate the source of an attack packet. If the attack traffic uses the same spoofed IP addresses for multiple attack packets, it is still possible to create a signature at the attacked domain. The signature in this situation would point to an innocent domain and could potentially result in denial of service for the domain which is wrongfully accused of sending the traffic.

4.3.8 SOS Alert and Collaboration

As noted in the previous section, if the attack is distributed enough and/or the IP addresses are randomly spoofed, a target domain cannot create an attack signature. In such a situation, an under attack domain relies on its neighboring domains to police traffic on its behalf in order to maintain operations during an attack. AMP proposes two profile based heuristic schemes to reduce the overall attack traffic and maintain operations in such a situation. Each cooperating domain in our scheme
Figure 4.13: Cumulative distribution of number of packets and domains at an egress router

maintains a ‘regular’ profile and size of its ingress and egress traffic. If no signature is available for an attack, the neighboring domains start policing on this traffic profile.

In order to enable such policing mechanism, two things are needed. Firstly, a peering domain has to have an idea of its ‘regular’ traffic profile at an egress router. This ‘regular’ profile is determined by sampling egress traffic during random time-slots on each of the egress-routers. Through this data collection, a domain knows its normal traffic profile to every other peering domain. The first heuristic is based on the observation that most egress traffic at a router actually follows a heavy tailed distribution. This means that the bulk of the traffic comes from a few domains and IP addresses
We observed egress router data for a medium sized domain over a two day period to verify this behavior pattern. The results, corresponding to a one hour time-slot from the dataset, are summarized in Figure 4.13. As the figure suggests, 20% of the domains accounted for 92% of the packets sent during the time window. This effectively means that a domain would be able to maintain high operability even if it were to allow traffic from only 20-30% of its traffic sources. This observation is essential in maintaining operation during an attack where nearly all the traffic is spoofed and no attack signature is available. Each domain chooses a random time-slot of half an hour to collect its ‘regular’ traffic set for its egress traffic. This traffic set contains the domains that are responsible for the bulk of the egress traffic coming through a domain. While the normal set may vary based on time of day and day of the week, we feel strongly that this relatively simple collection enables operability during most high powered attacks.

Once an attack is detected, the KM issues an immediate distress signal to its neighboring collaborating domains regarding the attack. In the absence of an attack signature, the neighbors start profiling their out-going traffic based on the ‘regular’ profile they have for the destination domain. This helps to provide immediate relief to the under attack domain. Having issued a distress alert, the domain under attack tries to create an attack profile from the data that it has collected at the detectors. If the attacked domain is able to create an attack traffic profile, it transfers the traffic signature to its neighbors.

Individual domains can choose to act on or ignore an SOS alert from a domain. Individual domains base this traffic policing behavior on the trust that they share with a particular domain. For instance, if a domain does not have a trust relationship with another domain, it can choose to ignore its policing request. The response to an SOS request is observable at the target domain. If a domain continues to receive attack traffic from another domain, it can update its trust notion for the offending
domain based on this evidence. Our cooperative policing scheme holds domains as being ultimately responsible for the misbehavior of traffic that exits their network. This includes transit traffic as well as traffic originating in the local domain.

The second heuristic approach adopts a similar profile based filtering. The only difference is that it adopts rate-limiting filters at ingress routers. Cooperating domains also ensure that local, target-bound traffic is not spoofed by applying appropriate egress filters. In the event of an attack, the attacked domain applies rate-based ingress filters at its routers and alerts the neighbors. Cooperating neighbors respond to the alert by applying profile based rate limiting filters at ingress routers. During this time period, the incoming traffic, both attack and legitimate suffers heavily. However, as a result of the ingress filters installed at cooperating domains, the total traffic flow at the attacked domain is reduced. Therefore, the local filters adopted by the attacked domain are timed out. This results in an increased connectivity for clients that belong to cooperating domain.

Having applied filters at the ingress routers, the domains pass the alerts upstream. A hop by hop reduction in attack traffic pushes the filters closer to the source of the misbehavior. It also serves to isolate misbehaving domains that do not apply the filters.

4.3.9 Peering domain operations

Egress filtering is the control of traffic leaving the network. A set of aggressive egress and ingress filtering rule by peering domains can help reduce the effect of denial of service traffic directed towards a victim domain.

The KM controls the flow of traffic through its network by applying appropriate filters at ingress and egress routers. The application of the filters is shown in the figure 4.14. Before any specific filters are installed, the KM needs to determine if the source of the offending traffic is local or transit traffic. We adopted a traffic
FIGURE 4.14: Policing of misbehaving traffic by the KM
marking scheme at the edge routers to determine this. Upon receiving an alert, the KM instruments all its edge routers to start marking target bound traffic with the router identification number in the packet header. This allows a domain to determine whether the source of the packet is local or transit traffic. Due to the fact that IP addresses can be easily spoofed, this packet marking scheme is quite useful in determining if a misbehavior is local. Apart from applying the normal filter, the edge routers or the traffic filtering entity co-located with the router randomly samples destination bound traffic and sends it to the KM. By observing the sampled traffic, the KM can determine if the target-bound misbehavior is local or external traffic.

In case the misbehavior is local, the problem translates to locating the source of the misbehavior and shutting it down. The KM applies Egress filters on the edge routers and sends an alert message to the administrator to check out the offending IP addresses. If the packet is not marked and the source of the packet is not local, it can easily be identified as a local machine pretending to be an external machine. If one of the local machines is being abused as a reflector machine, an external machine would be spoofing the target IP address as a source address and sending a request to the reflector machine. This can be checked by installing an ingress filter with the source address of the attacked machine and destination address which is present in the signature. In general, these filters should be part of good network administration principles. No machine should be allowed to spoof the IP address of another domain. If the domain has a well designed sensor architecture, any traffic anomalies can be reported and acted upon. Well placed sensors coupled with a programmable firewall can be used to further push the egress filtering to the individual sub-networks responsible for the misbehaving traffic rather than doing it at the edge of the network. The Foresight sensor architecture can be utilized for the purpose of individually identifying the machines.

If the misbehavior is transit, the KM applies the appropriate ingress filter at the
ingress router responsible and sends a distress message to the upstream KM. While our current approach for traffic filtering is software based, specialized traffic shapers can be optimized to efficiently handle this. Doing traffic shaping on a dedicated traffic shaping box also avoids burdening routers with other tasks, leaving the router free to focus on routing packets as fast as it can.

Each neighboring domain decides if and when to propagate the SOS alert further upstream. In our experiments, we propagate the SOS alerts further upstream in order to show the effects of pushing the filters closer to attack sources. This is essential in the case of reflector or zombie attacks where a domain that is sending traffic would not even know it is part of an attack unless informed otherwise. Egress filtering and hop-by-hop alert transmission also enables an organization to regulate misuse of its resources as well.

4.3.10 Termination

The attacked domain continues to monitor samples passed to it by the edge routers as well as the Server-prof. Once the appropriate filters have been installed locally as well as globally, the KM has to decide when and if the attack is over and normal operation can resume. When the detector moves to the “no attack” state, the KM performs the verification step by polling the Server-prof. In order to foil attacks that aim to exploit determinism in the behavior of the detector or to cause it to go into oscillations (by repeatedly switching the attack on and off) the detector only terminates filtering the traffic when it receives at least $K$ consecutive verifications that the attack is over. D waits a random period of time and then performs verification again. After $K$ consecutive verifications, it returns to the normal detection mode taking off the local filters. The absence of attack packets in the incoming traffic however does not mean that the attack has ended. Therefore, while it is ok for local filters to be removed, global filters however are not removed. Our scheme uses an arbitrary time-out period
of 30 minutes to reassess the policing filters applied on behalf of neighbors. If there are no packets in the sampled traffic being logged at the KM after the specified time period, the KM verifies the no attack scenario with the target domain. Upon receiving verification of no attack, the filters are taken down.

4.4 Prototype Implementation and Results

In this section we describe our evaluation methodology for the AMP architecture, simulation, experiments and results. The main idea behind the Foresight architecture is that security is a community responsibility and that aggregation and sharing of forensics data leads to better policing decisions about wide area attacks. In some instances the wide area effects that would possibly go undetected can be detected. We also show that behavioral performance analysis is a good scheme to detect network based attacks in order to complement advanced intrusion detection schemes. There is little or no performance overhead at the server since most of the performance degradation data that is collected by Server-prof is already collected as part of standard operating system statistics. Also, in the absence of a router based implementation of the Edge-router, software based implementations although much slower, can be used to create a fully functional solution. Specialized hardware based solutions however can significantly increase the performance of Edge router detection and filters.

While the architecture does suggest using data plane manipulation in the edge routers, we actually perform the ingress and egress filtering at a programmable firewall machine co-located with the router. The firewall has a secure bidirectional communication link with the KM. The relative difficulty level of implementing data plane manipulations in the routers forced us to go this route. Apart from being external and easy to install, we felt the added complexity of modifying and updating router software was orthogonal to our effort. The detector is implemented as a stand-alone process which is co-located on the same machine as the firewall, it passively
monitors all the packets that are directed to it by the firewall. A software based approach does have the disadvantage of being slow and hence vulnerable to packet losses. We do however observe that the same functionality can be performed in specialized hardware to greatly increase its performance. Server-prof runs as user-space process on the server machine.

Due to lack of a good experimental setup to test our system, we used a java based packet level simulator to test the performance of our scheme under various simulated attacks. Our simulator is designed to recreate traffic directed at the server. We simulate three different kinds of DDoS attacks, SYN floods, reflected attacks and attacks from spoofed IP addresses. The good traffic directed at the server was based on the real traffic trace collected at our web-server. We modelled the attack traffic based on the three attack types mentioned earlier. Our simulator runs at a packet level granularity and does not model individual TCP flows. Simulation as a verification scheme has its own set of advantages and disadvantages. The biggest disadvantage of simulation is that real security applications cannot be tested on it. We do however note that our scheme relies on verifiable individual components. For instance, the network firewall that forms the basis of our filtering scheme can be any commercially available firewall. The disadvantage however is that time passes in discrete steps rather than being continuous and simulated networks do not contain performance bottle-necks etc like bandwidth and latency constraints, routers firewalls and bridges.

4.4.1 Simulator setup

We used a randomly generated topology of 100 domains with a random sampling of malicious and non-malicious, cooperating and non-cooperating nodes interacting with our public-domain Apache web-server (Figure 4.15). Each domain in the simulated topology was connected to a subset of its neighbors signifying peer routing
relationships between domains. Every domain was modeled as having 50 clients machines that were used to send traffic to the webserver. Our simulator used a shortest path routing algorithm to direct traffic to the server. Normal server bound traffic was generated using traffic characteristics from the netflow and tcpdump datasets, malicious and denial of service traffic was generated using typical flooding based DOS attack traffic profiles. Denial of service attacks typically constitute extremely large volumes of data (a typical DNS based reflector attack size can be 10Gbps at
We used the traffic data collected from the peak hour slots to model normal traffic during our simulation runs. Our simulations assume an average of 100 packets per second with a standard deviation of 20 as the normal incoming traffic rate at the server. Normal traffic measured at our webserver during the data collection process constituted 80 KBytes per second. Assuming average packet size to be 800 bytes, once again derived from the traffic collected, this corresponds to an average of 100 packets per second. Attack traffic was modeled to have a varying rate from 2000 KBytes per second to 200 KBytes per second. While attack traffic can be significantly larger than this, we assert that attack detection at higher traffic rates would essentially have the same characteristics as lower rates.

Figure 4.16 shows the results from one of the set of experiments run on the simulator. We kept detailed traffic arrival statistics at the edge-router for the attacked domain. Total traffic statistics, traffic from an individual source, percentage of packet types arriving per time slot were kept at the edge-router in order to determine the presence of an attack. Attack traffic thresholds for total traffic, packets from an individual source and for different types of TCP packets were determined from the previously collected data sets. In short, the complete functionality of the first stage of
the detection algorithm was implemented at the edge-router. Since our experiments were run on a simulator, we did not model the host-based Server-prof. Instead, we used threshold measures derived from our earlier experiments to simulate the red-flag alert from Server-prof. If the traffic crossed a certain threshold, the server raised an alarm. Good traffic reaching our server followed the traffic profile of normal TCP traffic as derived by the CAIDA dataset. Also we programmed the 20-90 behavior of the egress traffic in the simulator. 90 of the traffic exiting a domain was coming from 20 percent of the IP addresses.

In the first set of experiments, we assumed hundred percent cooperation between the domains. Therefore, each domain calculated a ‘normal’ traffic profile for every one of its out-going routers. The first 500 seconds of the simulation were used to create a ‘normal’ traffic profile for each of the routers by the individual domains. The simulated denial of service attack was started 500 seconds into the simulation. The sharp spike in total traffic reaching the webserver at 500 seconds represents the start of our simulated SYN Flood denial of service attack. In this experiment, the simulated attack had two components, the first part of the attack came from fixed IP addresses like in the case of a reflected attack. This attack was easier to handle and as soon as the signature for the attack was generated and the filters enforced, that particular type of attack traffic dropped to zero. The second part of the attack consisted of traffic coming from randomly spoofed IP addresses. In this scenario, AMP was unable to create a signature since no source was sending a large enough traffic size and the neighbors eventually started applying their ‘normal’ filters to the out-going traffic. In this experiment, the normal traffic profile was not a particularly good profile since it was calculated over a five minute simulation period only. However, as the figure shows, the server was able to maintain operation even during an attack albeit at a lower throughput. A certain portion of the attack traffic was still able to escape the filters but not enough to overwhelm the server. This
experiment showed the working of the behavioral monitoring approach in creating an attack signature where one was available. It also showed the impact of adopting a ‘profile’ based filtering scheme at the ingress and egress routers. This approach as described earlier, is essential in maintaining operations during an attack for which no signature is available.

4.4.2 Performance Evaluation

The main idea behind our experiments was to show the feasibility of a behavioral monitoring approach and cooperation towards large scale attack detection and mitigation. We also wanted to show that if the domains are careful in the creation of the normal traffic profile, servers are able to maintain operation even during extremely large scale attacks. As a third phase, we wanted to show the feasibility of our scheme against large scale distributed reflector attacks that followed a normal traffic profile.

Essentially, we identified three ways to measure the success of AMP in our simulation experiments. Firstly, a) Success Rate: The fraction of all attacks that were caught b) False Alarm rate c) Percentage of good traffic that gets through during an attack. In order to measure the global benefit of cooperating with other domains, we introduced a fourth metric; the overall malicious traffic in the system.

4.4.3 False alarm rates

The two main components of AMP are the detection algorithm which leads to the generation of the attack signature, and the cooperative mechanism which leads to the push-back of filters. We showed earlier that the router and server bound traffic follows certain patterns and trends. Significant deviations from the norm signify attacks. This set of experiments was performed to ascertain the success of the attack detection scheme under simulated traffic loads. We simulated three different sets of attacks at varying attack rates. The first set of attacks, referred to as type 1, was a
SYN flood attack directed at the server from fixed IP addresses. This type of attack is not entirely theoretical. Real attacks of this nature can occur if the attacker uses a zombie machine to attack the server. Type 2 attacks referred to a class of attacks where the source IP attack was spoofed at random. This attack fails the signature generation scheme since no IP address stands out in the data collected by the KM. Type 3 attack was based on a distributed reflector attack. The attack came from a set of reflector machines that each sent well formed TCP packets to the server. This attack was the most interesting to catch since we created its traffic profile to conform to the normal traffic profile.

The results from the experiments were compiled over 100 attacks of each type using a randomly chosen subset of attack nodes. The total number of attacking nodes in each experiment was varied from 2 to 10 to simulate the effect of distributed attacks. The higher the number of nodes that were involved in the attack, the lower the amount of traffic that was coming from each individual machine. All the experiments were repeated with a high attack rate and a low attack rate, one that resembled the normal traffic patterns. The results from the experiments are summarized in table 4.2

Our detection scheme is sensitive to the total amount of attack traffic directed at the server. For high enough traffic loads, the overall performance of the detection scheme against simulated attacks was excellent. We do however note that against a real traffic workload, the performance of the profile based detection scheme would incur false positives. Sometimes the traffic simply follows an odd profile. Our simulated good traffic followed the average data statistics acquired using our data collection effort. Our simulations also did not have the normal internet back ground traffic that usually ends up at the routers. We do note however that while the detectors in the edge-routers would signal an attack, the scheme still relies on the server performance measurements to confirm an attack. We simulated this scenario by introducing a 10%
false alarm rate into the edge-routers, the overall accuracy of the scheme was not
effected due to the verification scheme. The false negatives introduced however were
interesting. Further examination of the simulations revealed two sources of error.
Firstly, since for higher number of attack machines, the traffic entering each attack
domain was divided among the various entry points, that led to a lower detection
rate at low attack levels. Secondly, since the simulator used a randomly generated
topology with a random distribution of attack clients, in some of the simulations, one
of the local clients was infected thereby bypassing the edge-routers. We addressed
this particular issue by limiting the attack set to external machines for the further
experiments.

Attack type 3 was the most interesting type of attack. If the attack traffic had a
typical profile like type 1 and 2, for example, SYN flood attacks, the number of false
positives at the edge-router were lower since the traffic stood out from our definition
of normal traffic. If the attack traffic had the same profile as the good traffic, the
number of false-positives or false-negatives were higher at low attack levels. We also
note here that while our experiments did show a false negative rate of close to 6%
for particular attack rates, although the attack was in progress, the total traffic was probably not enough to overwhelm the server.

### 4.4.4 Attack Verification

We conducted a series of experiments with a high traffic load in order to detect the performance of the two-step verification scheme as opposed to the simple edge-router based mechanism. In the first scenario, only the edge routers were considered for attack detection. The second set of simulations was performed with the detection scheme working with the verification mechanism. Table 4.3 shows the results from the experiments. While the detection rates were 100%, there was a high number of false positives when we simply used the router-based scheme. In real life situations, the number of false positives would be even higher due to the probabilistic nature of the average and normal traffic statistics calculated at the edge-router as well as the traffic sampling. In our simulation, we did not perform any traffic sampling, hence, the total number of false positives was lower. We note therefore that performing the verification step improves the accuracy of the attack detection. This however comes at the cost of detection latency.

### 4.4.5 Total Attack traffic in the system

We note that for certain types of attacks, cooperative alert sharing and propagation can help reduce the total amount of attack traffic in the network as well as identify mis-configured or zombie machines. It was however tricky to evaluate the effect of this using our simulator. However, we measured the sum of the total number of hops traveled for each attack flow as an estimate for the total attack traffic in the overall
system. Figure 4.17 estimates the effect of attack traffic for different cooperating percentages in the simulation. We do note that this is a function of the effectiveness of the filters adopted by the various cooperating domains. We assume here that all cooperating domains apply the appropriate egress filters.

### 4.4.6 Different attack types

We simulated three different attack types in our simulator. The first attack was labeled Type 1. This attack was modeled on the SYN flood attacks. Each packet was of a particular TCP type. This traffic was also distinctly different from other attacks since the source IP address was not spoofed. That allowed our scheme to create a valid signature, once the attack was detected. Once the attack signature is created and an alert issued to the neighbors, the overall attack traffic in the system as well as the total amount of attack traffic becomes a function of the total number of cooperation domains in the system. We conducted this experiment for a varying degree of cooperation between the domains. We only show results for two sets of experiments corresponding to 100% cooperation and 50% cooperation (see figure 4.18). The rest of the results are skipped for brevity. The attack detection for various different high attack rates was 100% and the 100% of legitimate traffic
directed at the web-server was recovered for all levels of cooperation. If an attack signature was generated successfully, the only cooperation needed by a domain is effectively that from its immediate neighbors. However, the total amount of attack traffic in the system as calculated by the total number of hop-counts traveled by the attack traffic gave the true effect of cooperation. For lower levels of cooperation, the amount of bandwidth wasted was much higher as showed by the number of hops traversed by the attack traffic.
Figure 4.19: Attack Type 3

The next set of experiments was performed with attack type 3. As described earlier, Type 3 attack was modeled in the simulator to represent reflected attacks. Type 3 attack traffic followed the normal traffic profile and was not distinguishable from it. One peculiar thing about reflected traffic is that it does come from fixed IP addresses, notwithstanding the fact that they are innocent IP addresses. In this respect reflected attacks are similar to type 1 attacks where signature generation is possible. If the signature generation is possible, the attack detection scheme works exactly like type 1 attacks and once appropriate filters are installed, 100 percent legitimate traffic is restored. The results from this set of experiments for varying degrees of cooperation are displayed in figure 4.20

The third and the most interesting type of attack was type 2. Type 2 attacks were simulated to come from a random IP address each time. This made signature generation impossible since no IP address was available. This also prevented IP based
filters from stopping attack traffic. This attack tested the normal filtering scheme. We ran each simulation for sometime in order to create a normal profile of traffic at each domain. We then introduced the Dos attack into the simulation (see figure 4.20). Being unable to create a signature, the attacked domain passed on a distress message twice prompting the neighboring domains to adopt profile based filtering. This allowed a significant portion of good traffic through to the server. While a proportion of the attack traffic was till getting through to the server, alert sharing between cooperating domains helped reduce the total amount of attack traffic getting through to the server. In the experiments that we ran, 100% cooperation between domains helped restore 92% of the normal traffic while eventually blocking all the attack traffic through egress filters at the appropriate domains. In a separate set
of experiments, with 50% cooperation between domains, the total amount of attack traffic getting through to the server was reduced to manageable proportions.

A separate series of experiments was performed to evaluate the rate-limiting heuristic scheme. The results from this set of experiments are summarized in figures 4.21. While there was an initial drop in the good traffic, the scheme slowly increased the total amount of good traffic reaching the server as the filters were pushed further back. An added benefit of the cooperative scheme was that clients belonging to cooperating domains were getting a better quality of service during an attack as compared to non-cooperating domains.

4.4.7 Massively distributed attacks

We described a scenario earlier where a domain A was under attack from a large number of distributed clients. While the total traffic reaching the server was very
high, each individual client was not sending enough attack traffic to raise local alerts. We conducted this final set of experiments for massively distributed attacks of type 1 and 2. By dividing the total attack traffic over a large number of attackers, we made sure that no one domain was sending too much traffic to the server. The only way to detect this misbehavior was through cooperation between domains. Figure 4.22 shows the results from our experiment with distributed attacks. As the graphs show, while the attack was Type-1, the domain was unable to create a signature and therefore resorted to normal filters. This ended up reducing the good traffic reaching the server by about 10% but eliminated the attack traffic completely. We note here that the total attack traffic in the system is however a function of the cooperation.

Figure 4.22: Distributed attack, no signature created
between domains. The results shown in the figure assumed complete cooperation and hence resulted in complete blocking of the malicious behavior through egress filtering.

4.5 Conclusions

In this chapter we have presented a framework for countering distributed denial of service attacks on servers that serve static as well as dynamic content. The framework relies on effective message passing between cooperating domains to enforce better traffic policing. An evaluation of the solution on our simulator produced results in line with our expectations. AMP can be deployed incrementally by domains as a service for clients. Peering domains can also enter into bilateral agreements to help enforce egress and ingress filtering on behalf of each other. Only those service providers who want to offer the service need to deploy it and similarly only those web servers that need to subscribe to the service need to install Server-prof. Since we separate the detector from the router, the scheme does not slow the router down.

The overall performance of the detection scheme was very high. The performance of the cooperative scheme in lowering the total attack traffic in the system was significant. The profile based filtering scheme was successful in restoring up to 90% of legitimate traffic even during extreme attack conditions.
5.1 Community/Trust Creation and Propagation

Foresight relies heavily on the creation of bilateral trust relationships between administrative peers. Trust relationships between entities on the Internet can be developed through one of two ways. The first is commonly referred to as the Public Key Infrastructure (PKI) and is composed of a tree of certifications with the root’s trustworthiness being self-evident to all subjects. Care must be taken to safeguard the credentials of the root of the trust. The alternative scheme uses a web-of-trust. Here an entity manually bootstraps the establishment of trust by using mechanisms external to the system to determine a set of other entities that are deemed to be trustworthy. Trust is assumed to be a transitive property and an entity that is trusted is allowed to certify other entities as trusted as well. The risk of compromising the integrity of the entire system is amortized over the integrity of a set of entities in the system. No single node can be compromised in a manner that is catastrophic for the system.
5.1.1 Foresight trusted community

We introduce a secondary notion of trust that is dependent on the interaction of a domain with another. Foresight’s reputation mechanism leverages upon observations of behavior and testimonies from peers to create a web of trust in order to enable collaborative achievement of individually and jointly set security goals. The idea is to build mechanisms into the overall architecture that induce desirable behavior from a majority of the participants.

When a domain monitors another domain’s interaction with itself and its ‘friends’, it gains a rich set of information. Individual security policy and trust can be stated in terms of this accounting data. Each domain also maintains its own publish/subscribe groups based on this trust value where it can share forensics data with the domains it trusts. Priorities of service to other domains are also defined in terms of service received in the past from those domains. The overall architecture induces a desirable behavior from the participants by discouraging a domain from performing actions that are be detrimental to other domains. The reputation mechanism gives an overall estimate of the level of participation of a domain in the system. The reputation mechanism also provides an efficient way to aggregate and track feedback from participants in the community. Logically, it is a collective memory of the system that rewards good and shuns malicious or careless behavior. Foresight employs an incentive-based model where domains avoid individualistic behavior in order to achieve a cooperative optimum goal between the group. The cooperative optimum goal in our situation is the global tracking and policing of newly emerging rapid malware.

5.1.2 Basic points of the scheme

**How is trust developed?** Trust is established in our system in the same way that casual trust relationships form in human society. Positive experiences build
the reputation of a peer. The total reputation score of a domain B at domain A is dependent upon:

- Direct interaction with B,
- Interaction with domains referred by B,
- Reputation of B as advertised by others.

Positive interactions lead to an enhancement of a domain’s reputation. In addition to sharing alerts and threat related data, domains can also count on trusted partners to perform security related requests for each other. As outlined in chapters 3 and chapter 4, the security request can range from filtering target bound traffic during a DDoS attack to performing mail-server SMTP authentication before forwarding email.

Every request made to a domain has an observable response through the rest of the system and any misbehavior on part of the domain, deliberate or otherwise, leads to a loss of reputation. For example, domain A, under a denial of service attack can request peering domains to police suspicious traffic before it enters domain A. For the purpose of our example, this traffic could either be originating from domain B or it could be transit traffic. Either way, the response to such requests is observable at domain A and can be used to enhance or decrease a peering domain’s reputation.

In another case, a domain receiving malicious email traffic from another domains can instruct it to stop such traffic at source. Our system holds each domain responsible for misbehavior that it originates or transits. Since interaction between domains is contingent upon a bilaterally good trust relationship, it is in the interest of peering domains to comply with A’s requests and vice versa. Interaction between participating domains can be of four types

- Security Requests
Figure 5.1 describes the main components of the reputation score computed by each domain for another. The overall reputation score of a domain is the summation of the individual components. The actual weights $w_1$, $w_2$ and $w_3$ assigned to each component of the reputation score represent the individual policy of each domain, for instance a domain might only be interested in the bi-lateral interaction with

- Sharing of non-repudiable transitive Trust certificates
- Sharing of Security Threat Alerts and Warnings
- Sharing of forensics data and logs
others and may choose to ignore the Community Reputation component altogether. Reputation scores are normalized to a scale of 10.

When a domain subscribes to security alerts from another domain, it chooses how often it wishes to interact with the neighbor. The situation is explained better in terms of the Import/Export matrix 5.2 maintained at each Collaboration Agent in the Foresight Architecture. This matrix, maintained by each domain, keeps track of how often the data is to be shared between various domains. The frequency values represent how often the transfer of information happens in each direction. We assume bi-lateral transfers of information for our design. In reality though, since domains often have asymmetric resources and hence expertise, the scheme can be expanded to allow one way interactions between publishers and subscribers of data as well. Apart from the periodic push/pull information exchange, each time an event of 'interest' occurs, the Collaboration Agent notifies all the recipients that have subscribed to that kind of event. Every domain interacts with its neighbors at a pre-specified frequency.
The reputation score decays at the same rate as the rate of the interaction defined in the sharing matrix. If there is no interaction during the time window specified, the reputation score automatically decreases by 1. An exchange of alerts during the specified window maintains the reputation score at the current value.

All request/response sequences are rated as positive or negative. A positive request/response sequence results in an increment to the direct recommendation score. Misbehaving traffic does not directly effect reputation scores since it is hard to ascertain the source of misbehavior. However, negative request/response sequence results in the loss of 50% of the reputation of a domain. We have deliberately kept this harsh in order to punish lack of cooperation and resource commitment from domains. The reputation score of a domain is non-negative and cannot drop below zero.

Based on the reputation score, we define trust as a binary value that defines a domain’s membership in another domain’s ‘Friends’ group. However, more complex definitions of groups interaction can be built upon the normalized reputation scores in order to cover a host of interactions between domains. Each domain can define an individual group to represent a different type of interaction with other domains. For instance, there can be a separate ‘email’ group to share email related data, and ‘Dos’ group to share denial of service data. Currently we represent the ‘friends’ group to represent all such interactions and restrict interaction between domains to this group only. A domain can only send security requests to domain outside its ‘Friends’ group.

**How is misbehavior punished?** Since interaction between domains is dependent upon the notion of trust, counter measures against misbehavior, selfishness and free-riding are essential to the performance of Foresight. A reputation scheme can be subverted by a patient attacker who carefully builds a positive profile and then abuses it. Foresight tackles this issue by weighting the learning algorithm to grant trust conservatively and take it away rapidly. We also observe that most peering relationships between domains on the Internet are long term hence less susceptible
to short term misbehavior. The scheme is particularly harsh on lack of response to security requests. A delayed or no-response to security requests results in the biggest reputation loss in the short-term and hence exclusion from the ‘Friends’ group. Misbehavior results in the loss of 50% of the reputation score of a domain.

**How is trust Propagated?** Each domain maintains a reputation score for all the domains it interacts with. Domains periodically exchange these scores with their neighbors. Foresight utilizes transitive certificates of trust from other domains, with such certificates being weighed according to the current notion of the certifiers trustworthiness. This transitive trust is only used to bootstrap the reputation in case a new domain is encountered. A neighbor demonstrates a history of being trustworthy gets a greater weight and has more influence indirectly on local trust perceptions but this maps to our intuitive notion of trust. When the trust is violated, the change against the recommending domain can be weighted heavily so as to discourage casual certification of other domain.

A reputation scheme based on sharing data with the peers can itself suffer from a denial of service attack due to the generation of a high number of genuine or false alarms from the community. We simply counter this by letting a domain decide how often it wishes to interact with any other domain. In order to further limit the interaction, each domain that wants to share data with others, sends a message to its neighbors. Any neighbor that is interested in the data can access it using the Paranoid group access mechanism. This way, there is no actual data transfer per alert. Any data transfer is initiated by the receiver. This simple mechanism helps regulate the interaction between domains and helps keep the traffic from escalating to denial of service proportions.
5.1.3 Access Control, Privacy And Security Concerns

Domains are traditionally reluctant to share security threat related information and forensics because it may disclose private information, local topological information, security policies and vulnerabilities. This data, if left unprotected can also be potentially utilized to observe weaknesses in the system etc. Foresight uses Paranoid [102] group semantics to address privacy concerns and to secure globally shared data. Each domain defines a ‘Friends’ group that it wants to share its forensics data with. Any domain interested in joining that group can issue a subscribe request to the domain’s Collaboration Agent.

Published data contents are locked via encryption and are unlocked only with the correct key. Access control thus transforms into a key management problem. Domains are implicitly authenticated by their ability to gain access to keys. Paranoid uses a novel approach using transform keys to address the key distribution and revocation problems. The access control protocol lets each domain define individual access groups according to their trust relationship with peers. Each trusted member has access to group accessible files without having a shared group secret. Such a scheme prevents a group member from adding members to the group by sharing the group secret. The system works because only trusted group members with the right capabilities have informational access to the shared data. We have successfully implemented this scheme in the design of a globally accessible secure file system [102]. Paranoid group semantics are described in detail in chapter 6

**Greed, Selfishness and Collusion** A major challenge in the management of such peering trust relationships is to control the resource commitments of peers to the collaboration. As with a real society, if the policies tend toward altruism on average, the system at large is likely to be more efficient, but is also more vulnerable to a policy constructed with greedy algorithms. The goal of a selfish domain within our
system would be to maximize its own utility over time. The use of a capability based access control model forces cooperation from participating domains since access to trusted groups is contingent upon continued good behavior. Also, continued response to security requests is also dependent on continued cooperation. Greed therefore, is a short-term ineffective policy. Simulations show that our simple trust scheme is robust enough to isolate misbehaving domains. As long as it is beneficial to share data, domains would voluntarily participate since the individually beneficial goal is also the socially optimal action.

A set of domains can attack the system by developing collusive rings to artificially trade or inflate reputations. However an attacker would be forced to create attacks that target the community as a whole. We use non-repudiable digitally signed receipts to track negative recommendations in our system. This discourages casual or malicious negative certifications. The scheme also helps identify domains that issue inconsistent recommendations.

5.2 Simulation

We simulated our simple reputation scheme using a java based multi-agent simulator that simulates the reputation exchange protocol as well as the periodic interactions between domains. The focus of our simulations was the ability of the scheme to identify and isolate misbehavior quickly. Towards this goal, we generated synthetic request logs along with malicious and actual recommendations in order to track the overall stability of the system against misbehaving nodes, collusion and free-riders.

Our experimental setup uses a constant periodic rate of exchange of reputations between cooperating domains which we refer to as the cycle. In reality though the rate of exchange can vary from domain to domain and individual policy. We also assume a constant reputation-decay rate of 1 per cycle. Every time a domain does
not interact with a peer within the data sharing cycle, its direct reputation for that partner drops by one. This helps to keep the reputation information fresh since older interactions are given less weight.

Each node shares its reputation matrix with its cooperating peers every cycle. Every positive request/response sequence results in an increment of 2 in the direct reputation score. We also assigned the values of .5 to $w_1$ and .2 and .3 to $w_2$ and $w_3$ respectively to signify the weight given to the direct reputation, community reputation and referrals. This assignment gives more weight to the direct interaction between domains while keeping the community aspect of the reputation as an significant factor. As discussed earlier, in reality though, each domain can choose to assign its own value to these weights in order to personalize its trusted group.

Our simulation setup consisted of a randomly generated topology varying from 10 to a 100 domains each node having an equal probability of interacting with another domain. We measured the performance of our scheme on how quickly and effectively rogue and non-cooperating elements were isolated. The average reputation of a rogue domain in the system was taken as a rough estimate of how the reputation information of a domain suffers as a result of its misbehavior. We conducted a series of experiments to determine the effect of cooperation between domains, the number of peers each domain shared its reputations with and the cycle time.

5.2.1 Effect of Cooperation

The first set of experiments was conducted to ascertain the effect of cooperation on the reputation dissemination scheme. Figure 5.3 shows the results from the set of experiments performed for identifying a rogue domain with varying degrees of cooperation between domains. In the first set of experiments, we simulated one misbehaving domain and monitored its reputation through the community. For this set of experiments, we assumed a constant cycle length of 50 time units for all the domains. A
higher degree of cooperation between nodes resulted in a quicker identification of a rogue domain. The graphs show a saw-toothed behavior which roughly corresponds to the data sharing cycle. Our simulation setup had the same data sharing cycle across domains. If the data sharing were to be staggered instead of simultaneous, it would result in smoother lines. We conducted the same experiment with varying population sizes with similar results, we only show results for a population size of 40 domains for brevity. A second set of experiments was performed with a group of 5 misbehaving nodes colluding with each other and misbehaving as a group. We calculated the average reputation of the group over varying degrees of cooperation. Increased cooperation clearly led to quicker identification of rogue elements in the system.

5.2.2 Cycle Length

The second set of simulations was performed to determine the effect of cycle length on reputation dissemination. The cycle is the length of time that a domain waits before sharing its information with neighbors. This set of experiments was run with a topology of 40 cooperating domains with each node having an average of 4 domains it shared its reputation matrix with each cycle. Figure 5.4 shows the effect of the cycle
length on the reputation score from the resulting simulations. The same experiment was performed for a group of misbehaving nodes. The results indicate that the lower the cycle length, the quicker the system identifies the misbehaving nodes.

We conducted the same set of experiments with a higher number of friends for a node and the results were very similar in nature and are skipped here for brevity. Clearly, at more realistic number of friends, the effect of the cycle length is clearer. The lower the cycle length, the quicker a rogue domain is identified. However a lower cycle length means a significantly higher level of data communication between domains. While the total number of exchanges per cycle doubles every time the cycle length is halved, the overall data exchanged is actually just a few kilobytes. In fact, for a 600 domain topology, the reputation matrix actually takes 8 kilobytes per domain. With an average number of friends of around ten, and only one exchange per cycle, we do not see this as a significant source of network traffic.

5.2.3 Effect of Fanout

Fan-out represents the number of cooperating domains each domain shares its reputation data with. While we observe that average reputation a rogue node in the system is a poor indicator to show the effect of fanout in the system, we conducted experiments to show the effect of an increasing fanout on the reputation score (see
Figure 5.5: Effect of Fan-out on the reputation score propagation

figure 5.5). As the graph shows, the average reputation is a not a good estimate of the effect of fanout on the reputation of a node.

Intuitively, the higher the fan-out, the quicker the reputation drops for a domain that is misbehaving since more cooperating nodes know about a misbehavior. However, average reputation of a misbehaving node is a bad estimate for the drop in reputation since the drop in reputation would be higher closer to the node and between domains that directly interact with it. Domains that do not interact with a domain directly would smooth the effect of the drop and the results shown in figure 5.5 reflect this trend. While the reputation steadily drops over the course of the experiments, a more significant drop happens due to the gradual decay of the past good behavior of the domain as observed by the saw-toothed nature of the graphs.

In order to clarify the role of fanout better, we decided to count the number of domains that consider a node as un-trusted in order to determine the effect of a higher fanout on misbehavior. The results are summarized on figure 5.7. For a higher level of sharing in the network, a higher number of nodes recognize a node as un-trusted quicker.

5.2.4 Malicious reporting of reputation

Since each domain calculates its local reputation based on the weights we assigned to $w_1$, $w_2$, and $w_3$, malicious reputation dissemination does not have a huge effect.
The weights also make the scheme relatively resilient to malicious nodes deliberately issuing bad reputation for others. For instance, in this simulation, we assume a weight of 50% for direct interaction with a domain. So while the neighbors are telling me that the domain is misbehaving, if my own interaction with a domain is positive, my overall reputation is skewed more towards positive reputation specially if the domain keeps having positive interaction with me.

We measured the reputation of a node under simulated attack from a significant portion of the network. The first set of experiments calculated the reputation of a node between its peering neighbors. The sharp drops in the reputation show in figure 5.8 represent attacks by 10 to 50 percent of the nodes colluding to falsely advertise bad reputation for this node. As the graph shows, since a large proportion of the reputation is based on continued direct good behavior, falsely advertised reputations can only effect the overall score up to the percentage of $w_3$ which in this case is 30 percent. The second set of experiments was conducted to measure the average reputation of a node under such an attack as well as the overall count of the nodes that consider a domain un-trusted. Figure 5.8 represents the results which clearly mark out the attacks. While the number of nodes that do not trust this domain increases sharply, it corresponds to the number of domains which are part of the attack. The average reputation of the domain whoever is fairly steady over the course of the simulation.
Figure 5.6: Fan-out 100 domains
Figure 5.7: Count of Nodes who consider the node as un-trusted

Figure 5.8: Neighbor Reputations Under Attack
6

Secure Distributed File System

6.1 Paranoid Group Semantics

Foresight architecture requires, secure information sharing across cooperating administrative domains without compromising on privacy or security of data. We use Paranoid group semantics in order to provide this functionality. This chapter introduces the Paranoid, access control system that allows domains to selectively, securely, and easily share information with others, even those they don’t know and don’t have prior trust relationships. It provides users with means to share data, without the fear of compromising the security of information they consider private or privileged. Each domain is able to grant selective access privileges to others outside their administrative domain without having to create accounts or grant outsiders any user privileges.

Each domain can define different access groups for its data. A domain with access privileges can access files and data regardless of whether they are trusted local users or outsiders. One of the key features of Paranoid access control is that each group member has cryptographic access to the group accessible files without possessing
a shared group secret. The Paranoid Group Management protocol was developed as part of user level secure file system. The overall goal of the Paranoid system is to facilitate global data sharing with enhanced security and privacy and minimal administrative overhead. Data is locked via encryption and can only be unlocked with the correct key. Thus, access control to the data transforms into a key management problem. Domains are implicitly authenticated by their ability to gain access to keys. Paranoid uses a novel approach using transform keys (detailed in Section 6.3) to address the key distribution and revocation problems. The transformation key approach eliminates the need for a shared group secret for data sharing.

6.2 Design

6.2.1 Encrypted Files

Data that is to be shared is locked and made inaccessible by encryption. We use a hybrid encryption system for this purpose. Data is encrypted with a symmetric cipher and symmetric keys are encrypted with a public key cipher. Foresight can use separate files to represent the different types of data that it is sharing. For instance, Email viruses data can be stored in a separate file and a separate file could represent SPAM signatures. Each file is encrypted with a different random key. Since public key ciphers are too slow, we use a symmetric cipher. The current system is implemented using DES [61] but any other symmetric cipher could be used, such as AES [62]. Each symmetric key is encrypted with the owner’s public key. Group access to a file is granted by encrypting the file’s symmetric key with the group’s public key. This information, along with the file digital signatures, version number and a time-stamp are stored in a header, together with the encrypted file’s contents.
Figure 6.1: Sharing a File
6.2.2 Access Groups and File Sharing

When a file owner A wants to share a file with person B, the owner can encrypt the file’s symmetric key with B’s public key. The encrypted key can be stored along with the encrypted file or sent directly to B. Theoretically, this is all that is needed for file sharing. However, if a group of people is sharing a set of files, a more efficient method is adopted. The owner of a set of files defines an access group for the files. Group members encrypt files with symmetric keys and encrypt the symmetric keys with the group public key. In this case, the owner is responsible for distributing the group’s private key to all group members. This scheme poses logistical problems since explicit key distribution is needed. One solution is to store group access information in a file which can be provided upon request. A group access information file holds group identities along with the group’s private key encrypted with the public keys of group members. In this scheme, when a user is trying to access a file, he uses his identity to retrieve the group’s encrypted private key from the group information file. He then decrypts it using his own private key. The group’s private key in turn allows him to decrypt the file’s symmetric key, granting him access to the file’s content. The group owner is responsible for group management tasks such as adding, deleting and updating entries.

This scheme is similar to the lock-boxes adopted by Cepheus [24] although they used a central group database to distribute keys. Their scheme suffers from an inherent weakness. Not only is the database a central point of vulnerability, but the scheme gives users more rights than necessary. Group membership should only enable access to shared files. With Cepheus, any group member can add new members to the group by disclosing the group secret (private key). Additionally, revoking access rights is difficult. It requires changing the group’s public and private keys and the
re-encryption of all symmetric keys. Coordinating key changes over the Internet is difficult.

The Paranoid system uses a novel scheme that does not requires the sharing of a group-specific secret. When a user (a group owner) creates a new access group, he creates a new public and private key pair for the group using the RSA public key cipher [80]. He publishes the group public key, that is the modulus $N$ and the public exponent $g$. All group members use the same modulus $N$ but each group member is assigned a different random exponent as a private key. Associated with each group member’s private key is a transform key known only to the group owner. When a group member requests access to a file, the group owner applies a member-specific transform key to the file’s encrypted symmetric key. The transformation changes the symmetric key’s encryption from an encryption with the group public key to an encryption that corresponds to the group member’s unique private key. The encrypted file together with the transformed encrypted symmetric key are sent to the member. Please note that the system does not use explicit authentication. The system relies on the fact that only the designated group member possesses the member-specific private key, and therefore only she can access the file content. Others may pretend to be group members, but they do not posses a valid private key and thus cannot access the file’s content. Details of how the transformation is computed are given in section 6.3. Paranoid uses XML group definition files created by the owner. Users define their own read and write access groups. Group definition files are digitally signed so that any tampering can be detected. Each access group has a public and private key pair that are used by the group owner to encrypt and decrypt symmetric keys. The group private key is kept a secret and is not shared with the group members. Additions and deletions from the group are done by the group owner.
Each access group has an XML group definition file listing member identities, transform keys, and access privileges. The group definition file is encrypted with the group's public key. Operations like adding or removing group members and changing the access rights for a file can only be performed by the group owner. Shared files are accessed via a file server agent. The file server enforces access rights, checks group member's access rights and applies key transformations. File headers, encrypted files, and group definition files are digitally signed. While the system does not provide special protection against malicious file deletion, any tampering with Paranoid files can be detected using the signatures.

In order to globally share files, a file server agent must be running on the group owner's machine. This process authenticates access requests on behalf of the owner, performs key transformations and sends requested files to group members. Modifying group access rights is done by adding, removing or modifying a member's entry in the group definition file. Symmetric keys of files that a revoked user has already accessed can be lazily re-encrypted - that is, the operation is done at a later stage when the file is next written to. Note that only the group owner can perform these operations.

Adding a member to a group requires adding a new member entry to the group file, generating a random private key, computing a transform key, and delivering to the member his private key. The Paranoid system assumes that private key information is communicated to members in response to subscribe requests as an out of band operation. These operations can only be done by the group owner.

**File Read**

A domain can request access to a file by sending its credentials to the appropriate collaboration Agents on a peering domain. Figure 6.2 describes the chain of events
triggered by a Collaboration Agent running on a remote machine requesting a Paranoid file. To keep the description simple we only cover a successful file open case. The file server verifies that the files exist and that the domain presenting the credentials is a member of the access group for this file. This verification is performed using the group definition file. The file server identifies which group the requester belongs to and applies the requester-specific key transformation on the file’s symmetric key. The encrypted file is sent back to the client agent along with the transformed symmetric key. File tampering can be caught by verifying the digital signatures. The requestor decrypts the transformed key and can now decrypt the file.

6.3 Key Transformation

The Paranoid file system uses a modified version of the RSA public key cipher [80]. Each access group uses a different modulus $N$, but all the members of a group use the same modulus. The modulus and the public exponent of the group key pair are published and the private exponent is only known to the group owner. Each group member is given a random exponent to use as his group private key. Associated with each group member is a transform key, known only to the group owner, that can transform a symmetric key encrypted with the group’s public key into the symmetric key encrypted by the “public half” corresponding to the member’s group private key. Thus each group member can encrypt a symmetric key for group use, but he can only decrypt a symmetric key after his specific transformation is applied to an encrypted key. The transformation step prevents a group member from granting group access rights to outsiders without revealing their group private key. Since this can be easily traced, it is expected to dissuade leakage. In contrast, systems that hand out the group private key to users effectively allow them to add new users by giving the key to others without any accountability. Further, Paranoid’s scheme lets the owner remove a user from a group without having to re-encrypt any keys or files.
This section describes the transformation in detail. When a user creates a new group he creates a standard RSA modulus $N$ where $N = pq$ where $p$ and $q$ are two large random prime numbers. The group has a public and private key pair, $g$ and $g^{-1}$, where:

$$g \times g^{-1} \equiv 1 \pmod{\Phi}$$

where

$$\Phi = (p - 1) \times (q - 1)$$

Every group member $M$, $M = 1, 2, 3, ...$ is given a large random private key $e_m$ that is relatively prime to $\Phi$. The group owner also computes the inverse of this key $e_m^{-1}$, such that:

$$e_m \times e_m^{-1} \equiv 1 \pmod{\Phi}$$

Additionally, a transform key $\tau_m$ is computed using the following formula:

$$\tau_m = g^{-1} \times e_m^{-1} \pmod{\Phi}$$

The inverse private key $e_m^{-1}$ is discarded and the transform key $\tau_m$ is stored in the group definition file together with the member identity $M$. The group definition file is encrypted and is kept on the owner machine where it is only accessible to the file server.

Assume that a symmetric key $K$ is encrypted with the RSA cipher using the public key $< N, g >$, that is:

$$E(g, K) = K^g \pmod{N}$$

When user $M$ asks to read a file encrypted with $K$, the file-server computes $E(\tau_m, E(g, K))$ and sends it to the group member $M$. The group member computes:

$$K = E(\tau_m, E(g, K))^{e_m} \pmod{N}$$
Proposition:

$$K = E(\tau_m, E(g, K))^{e_m} \pmod N$$

Proof: Since the set of integers relatively prime to $\Phi$ is a commutative group under multiplication modulo $\Phi$:

$$\tau_m \times g \times e_m \pmod {\Phi}$$

$$\equiv g^{-1} \times e^{-1}_m \times g \times e_m \pmod {\Phi} \equiv 1 \pmod {\Phi}$$

Therefore, using Euler’s Totient Theorem,

$$E(\tau_m, E(g, K))^{e_m} \pmod N$$

$$\equiv K^{g^{-1} \times e^{-1}_m \times g \times e_m} \pmod N = K$$

QED.

Please note: Applying the transform key $\tau_m$ to a signature $H$ generated by $M$ with his private key $e_m$ transforms the signature $H$ into a group signature generated with $g^{-1}$. That is:

$$E(e_m, H)^{\tau_m} \pmod N \equiv E(g^{-1}, H)$$

The proof is almost identical to the proof of the proposition above and is left to the reader. Also note that as long as the group owner keeps $\Phi$, $g^{-1}$, and the transform keys secret, he can use the same modulus $N$ for many different access groups.

6.3.1 Transformation Security

A primary question concerning the key transformation scheme is how secure is it? The answer is as follows. Since each group member’s private key is a large random number, knowledge of the group public key and a group-member’s private key does not give an attacker the ability to gain additional capabilities beyond impersonating the member. Any set of collaborating group members could not gain any additional capabilities they don’t already have. For example, if group access privileges were
taken away from a set of group members, they cannot regain group access by collaborating. Any group member getting hold of a symmetric key encrypted with the group public-key could not decrypt it without knowing the corresponding transform key.

However, the transform keys $\tau_m$ must be kept secret. Any person that knows both a group-member private-key $e_m$ and the corresponding transform key $\tau_m$ can decrypt any symmetric key encrypted with the group’s public key $g$. Thus she can access all the group files, bypassing any access controls. If she was also able to penetrate the server, then she could modify files, forge signatures and alter the group definition file, adding or subtracting members.

6.4 Implementation details and results

6.4.1 XML

Paranoid files are encrypted and stored in XML format. An XML header is prepended to the encrypted data. Binary data, such as encrypted keys, is stored in hexadecimal format for readability. A simplified schema of the XML file header is given below. The header contains the file access information along with the protected decryption keys. The header also contains a list of groups having read or write access rights. An encrypted symmetric key is stored with each group name. The file contains a digital signature of the XML header and the encrypted file.

6.4.2 Group Files

The XML group definition file is a list of group members and member’s transform keys. The skeleton of one is shown in Figure 6.4.
6.5 Performance

Paranoid incurs a large cost for encrypting, decrypting, signing and verifying files. The use of cryptographic operations in the critical path of file operations has the potential to create a significant adverse impact on overall performance. However, we argue that this overhead is acceptable in the context in which Paranoid is to be used since the file operations are dominated by the latency introduced by the network transfers. Also the actual file transfers do not happen that frequently. Below we provide measurements of the time it takes to open and close Paranoid files to illustrate the effect it has on performance.

The measurements were made using two 300 MHz Intel Pentium II machines connected through the network. The benchmark program invokes the open system call a number of times over a range of files of different sizes. The tables shows the mean results over 10 runs each for 5 file sizes between 1MB and 64MB. Table 6.5 shows a client and server located on the same machine with the client opening and closing a Paranoid file. Table 6.5 displays times for a client and server located on different machines. The first transfer time is the measurement when the file is retrieved from the server by the client. This includes the time to affect the transform...
key on the server.

6.6 Related Work

Several previous projects have proposed the use of encryption to lock data stored in files. The Cryptographic File System (CFS) [6, 7], created at AT&T Bell Laboratories, was one of the early realizations of such a scheme. However, CFS was designed as a local file system. Therefore, the only way a file could be shared was by explicitly distributing file keys to other users. CFS used symmetric keys for all protection. This meant that the keys were left unprotected in memory while in use. Such a scheme is vulnerable when an attacker gains access to the system since they then have access to the keys as well. The use of a public key scheme like that of Paranoid reduces this exposure. Further, the granularity for file accesses in CFS is per directory. Paranoid can be used to provide per file read or write access and per directory create permissions.

The Transparent Cryptographic File System [11] is similar to CFS but it moves the functionality from user space to kernel space for performance and ease of use. Cryptfs [100] uses a stackable file system infrastructure to provide similar functionality. TCFS, Cryptfs and [32] have the same weakness as CFS, which is that the symmetric keys are unprotected. This can only be resolved through the use of a public key cipher in the protocol.

Network of Attached Secure Disks [27] and Secure Network Attached Disks (SNAD) [23] store data remotely and operate at block level. Data is unprotected on the server in the former with data servers cooperating with a single group server for access control, making it a central point of failure. SNAD uses certificates for authentication. However, since access is at the block, neither system can provide the end-to-end security semantics that Paranoid can.
The Self-certifying File System (SFS) [50], from MIT, addressed the problem of mutually authenticating servers users. This was done in order to prevent an adversary from spoofing the server. SFS achieves this through path names which embed the public key. SFS-Read Only [25] extended SFS to address the problem of securely sharing read only data across the Internet.

Cepheus [24] focuses on the separation of storage and group server functionality. It uses session keys to protect communication between the server and clients. The storage server does not need to be aware of the access control operations which are handled by the group server. A shortcoming of the system is the fact that group members are given the private key of the group. Paranoid’s transform keys prevent a group member from granting his group access privileges to an outsider without revealing his own private key. In contrast, possession of Cepheus’ group private key effectively allows a user to add new users to the group without revealing who effected the delegation. By restricting access to the group key to only the group owner, such direct leakage of rights is not possible with Paranoid. The transform keys of Paranoid force a user to divulge their personal private key allowing the source of such leakage of rights to be uniquely identified.

Plutus [38] uses a client based key distribution scheme. It focuses on using file groups to reduce the number of keys exchanged between users. Plutus, from HP Labs, provides group sharing by explicitly sharing the secret with all the group users. This suffers from the same problems as Cepheus described above. [79] compares several related cryptographic file systems.

The Encrypting File System of Windows 2000 [53] uses symmetric keys to encrypt files. These are then encrypted with a public key cipher for rights management. Since they are stored on the host, rather than with a PSP, they are exposed in the event of a system compromise. Further, Paranoid’s transform keys extend the scheme to enable cryptographic group access control.
The Secure File system, developed at the University of Minnesota [35] uses a protocol similar to Paranoid. However, a key difference is that access control is arbitrated by a group server rather than the end user. This does not have the end-to-end security semantics guarantees of Paranoid. In the event that a security compromise is detected in Paranoid, only the currently active files are at risk. In the Secure File System scheme, there is no way to prevent the attacker from accessing all the remaining files that the group server is responsible for but are not currently being used, if the system is compromised.

The Trusted Computing Platform Alliance [90] is an alliance of industry leaders in hardware and software. It aims to build a trusted computing environment on top of trusted hardware. The IBM 4758 Cryptographic Co-processor [85] is a high security, programmable PCI board which can be used to provide data and cryptographic processing to implement TCPA functionality. It contains tamper detection sensors, circuitry of cryptographic operations, a microprocessor, memory, and a random number generator. It aims to provide security even in the face of a physical attack on the device. Its high cost and weak processing power has hampered widespread adoption. Palladium [54] provides lower assurance security than such a trusted co-processor but is cheap enough to be commercially feasible for commodity desktop systems. Paranoid performs privileged tasks on the PSP. Data is decrypted into the client’s volatile memory and assumed to be secure if stored there temporarily. The PSP’s functionality could instead be implemented using the IBM 4758 or Palladium.

6.7 Conclusions

In this section we presented the Paranoid group management system. Each user can define access groups and grant group access rights to peers outside their protection domains. A novel public key transformation scheme is used to facilitate low cost re-
vocation of access rights. Performance measurements show that the implementation overhead is low enough to make it practical.
Figure 6.2: File Read operation
Figure 6.3: XML headers for Paranoid Files

Figure 6.4: Paranoid Group Files
Multi-modal signatures for multi-modal worms

7.1 Behavioral Signature

One of the main goals of Foresight was to identify and retard the growth of zero-day malware. As we discussed earlier, there is a significant ‘submission to cure generation’ window of vulnerability during which malware can spread unabated causing havoc to the infrastructure. Foresight finds its niche during this time slot of uncertainty. Once a threat has been analyzed by the security community and a cure generated, the problem of handling an attack reduces to the problem of updating anti-virus software, patching the relevant systems or intrusion detection systems. Our scheme attempts to create a generalized footprint of a threat while it happening through a fusion of the forensics data available to the community. This ‘behavioral signature’ of an attack is essential in preventing damage to critical resources while a thorough analysis of a threat is done and the systems patched against it.

Our behavioral signature attempts to address some important issues related to conventional signature based schemes. Any attempt to track an emerging threat is as good as the signature available to identify the threat. If the signature available
is poor, it would lead to inaccurate conclusions and poor decision making in trying
to stop the threat. Given the dynamic nature of emerging threats, it is becoming
increasingly challenging to create a signature of a threat. Existing malware signature
generation schemes share one common flaw: they look for some static element which
is common to existing or known malware [19]. The attackers simply end up removing
or changing this element in order to make the attack immune to detection. The
emergence of polymorphic threats has increased the problem manyfold.

A secondary problem arises due to the multi-modal nature of emerging threats.
Attacks are no longer confined to a single propagation vector. Worms, bots, spam,
phishing, viruses are all becoming increasingly coordinated and interconnected. The
result is a new class of malware that exploits different weaknesses on different plat-
forms in order to propagate. A uni-dimensional static signature is no longer sufficient
to represent such an attack. Nimda [64] serves as an excellent example of a multi-
modal worm. Nimda utilizes file infection, mass mailing, Web worm as well as LAN
propagation on local shares to spread. The worm also behaves differently on different
machines depending on how the machine is initially infected.

7.1.1 Advanced Host-based sensor

We applied the Foresight design principles to two instances where we described two
host-based sensors. Net-man was part of the Mail-trap architecture and was used
to collect process level information regarding network traffic. We used net-man to
provide more context to our anomaly based approach. AMP on the other hand used
Server-prof, a host-based sensor that collected data on the health of a web-server.
While both of these sensors were used in a very basic capacity in the respective
contexts, they are ideally placed to collect much more advanced information about
unknown threats. Host-based sensors are successfully used to collect information
about services, processes, log-files, registry entries etc. While AMP only required
Server-prof to look for performance degradation symptoms such as high CPU utilization and resource exhaustion etc, we designed Server-prof to sample network usage, cpu, memory, disk usage and processes running on a server. For each process, the advanced Server-prof maintains a list of files that it has accessed during an arbitrary time slot. Such a host-based sensor can be used to provide early warnings and create much better signatures for multi-modal attacks. We explain our ‘behavioral signature’ scheme in the light of this advanced Server-prof.

By monitoring the network behavior of processes and file access patterns, we can track anomalies better and by aggregating this data at the KM, we can better understand an emerging threat. Server-prof was designed to work in the back-ground and log the data locally. If the KM observes anomalous network behavior, the data collected by the sensor can be retrieved and used to perform advanced analysis.

7.1.2 Data aggregation

Foresight proposes the use of a novel behavioral signature scheme that attempts to capture malicious behavior from different geographically and topologically dispersed vantage points in order to create a ‘better’ signature for the threat. An aggregation of worm behavior across cooperating domains also better represent the multi-modal and polymorphic nature of newer threats as compared to a uni-dimensional signature. Our simulator based results for Mail-trap suggested sharing forensics data can help in catching polymorphic email based worms. We extend the same idea to sharing information collected by a wide range of sensors such as our advanced host-based sensor. In the event of a massively polymorphic attack, each domain and hence each of its sensors would have a unique perspective of the threat. Each domain has a unique perspective and signature of an emerging threat from the sensors deployed in their network. By pooling its collective resources, cooperating clients and domains can create a better signature than one which is only based on local information. This
is helpful in two ways, firstly, the system is able to catch many different propagation
vectors of a threat, even those that are not visible at the current domain. Secondly,
this allows domains with lesser resources to benefit from the experiences of domains
with more comprehensive sensor architectures. Let us illustrate the situation with
an example.

We mentioned the analogy of the six blind men trying to observe an elephant
with their observations in chapter 1. A unified global picture of the elephant is the
closest approximation to an elephant as compared to the individual observations of
the blind men. This basically says that as we observe more and more information,
we can always predict another (unobserved) random variable at least as well as we
could when we had fewer observations. This is also intuitively true. The problem
has very basic proofs in information theory. After all, you could always ignore some
of the information, use just the remaining subset of info and get essentially the same
prediction as you would have with fewer observation. Therefore, you can only predict
better as you have more information available. Suppose X and Y are two random
variable which take on a finite set of values according to a probability distribution
p(x) and p(y). Then the entropy of this probability distribution is defined to be the
quantity $H(X)$ and $H(Y)$ respectively.

$$H(Y, X) = H(X) + H(Y|X)$$

we also know that

$$H(Y, X) \leq H(Y) + H(X)$$

with equality if and only if X and Y are independent.

$$H(Y, X) = H(Y|X) + H(X) \leq H(Y) + H(X)$$

which leads to

$$H(Y|X) \leq H(Y)$$

193
with equality if and only if $X$ and $Y$ are independent.

7.1.3 Behavioral Signature

Our signature scheme is a modification of the basic concepts introduced by the Common Language Project [34] in order to create a behavioral signature for malware. The Common Language Project describes an attack as a series of steps taken by an attacker to achieve an unauthorized result. Our scheme is an attempt at documenting attack information in a structured and re-usable form. The common language project started off as a taxonomy of a security incident. An Event (Figure 7.1) is defined as an action directed at a target with the intention to change the state of the target.

An attack is described as a series of steps taken by an attacker. A successful attack exploits a vulnerability in a tool to perform illegal actions on a specific target. The common language project presented a matrix of possible attack based on their
experiences. We show the completed matrix with a few modifications in order to represent some of the newer attacks (Figure 7.2).

An attack represents a path from the left most column to the right most column in the figure. This is not an exhaustive list of possible attack scenarios, however we feel it has the ability to capture most attacks while presenting the data in a human readable and understandable format. The first step in the sequence of events that constitute an attack is the set of tools that are utilized. An attack can take the form of a valid program installed on a computer, a remote machine instigating an attack, an email that triggers off the attack or a rogue malware program that initiate an attack. The data on the tool utilized to initiate an attack can be deduced from the various host and server based sensors installed inside a domain. The tools exploit a vulnerability in the configuration, design or implementation of the existing software to create a security event that can lead to unauthorized results.

We rely on the basic anomaly based detection premise that normal traffic and nor-
mal behavior follows patterns. If the host-based sensors are appropriately installed, a centralized KM can poll them to access their logged data in case anomalous behavior is detected. Foresight architecture is not limited by the specific anomaly detection schemes that we describe in this thesis. The data sharing mechanisms and the secure access control can work with any set of detectors built into the system.

### 7.1.4 Signature

Each domain would have a heterogeneous sensor architecture installed on its network. The specific architecture can range from the Mail-trap application, AMP, packet based sniffers, host based network monitors, registry monitors etc. Each sensor periodically logs its data at the Knowledge Manager. Even in the case of emails worms that show varying behavior, each domain that has the Mail-trap architecture installed has a unique perspective of the overall attack. Cooperation between different geographically and topologically dispersed vantage points that combines data and knowledge from different sensors with the aim of maximizing the useful information content gives a much closer picture of the overall attack. For example, in the case of Nimda, if the attack signature only consisted of the email component of the attack, it could completely miss out on the file system component or the web-server component. On the other hand, if domain A experienced the symptoms

**Figure 7.3**: Signature format
from the email attack and domain B experienced the symptoms from the web-server infection, they could cooperate to create a joint signature of the threat which would better represent the attack. Let us suppose there are four domains in our imaginary network. Two of the domains have Mail-trap installed on their network. Out of the other two, one domain has host-based sensors installed to monitor automatic registry modifications, the other domain has Gen-prof installed on its network. As we recall from chapter 2, Gen-prof is responsible to passively observe the network behavior of the machines on its sub-net. It records traffic flow data, number of connections initiated to and from a single client and sudden increases in traffic through specific ports or protocols. Now if there is a multi-modal, polymorphic worm in the system, the aggregate view of all the domains would be sufficient to cover network scanning vectors, registry modification vectors as well as email based vectors. If each domain is left on its own, each one would be unable to protect against a multi-modal attack. A multi-modal attack as such would be a set of paths from one end of the graph to the other, from ‘Tool’ to ‘Unauthorized results’.

We store the behavioral signature in an XML format shown in Figure 7.3. A behavioral signature is able to represent a highly morphing attack by the ability to represent malicious behavior patterns rather than strings. Efficient coordination between domains that leads to temporary behavior blocking also helps slow down epidemic spread of attacks. Another useful feature of this scheme is that it captures knowledge in a reusable and easy to read form so that a system administrator can make sense of the signatures and recognize suspect patterns.

Foresight also collects a detailed audit trail of the events leading to a threat rather than relying solely on evidence unintentionally left by an attack. We believe that the observable network behavior of a host coupled with an application level view can identify threats as well as give essential clues as to the causes with a high probability. An intersection of forensics data collected from multiple geographically
dispersed domains would be essential to determine the way a malware exploits the underlying host. Better forensics would be essential to trace back the threat, locate effected components, undo its effects and identify potential causes.

7.2 Analysis of the new signature

In this section we tested the performance of the new behavioral signature scheme in detecting and responding to stealthy, polymorphic, multi-modal attacks. We present experiments and simulation results to show that the resulting ‘behavioral signatures’ can better represent multi-modal threats compared to existing approaches.

7.2.1 SIMDA

We started off by creating a simulated version of a multi-modal attack which we named SIMDA. We simulated the behavior of our worm on our simulator using the behavior characteristics of a cross-section to some of the popular worms, hence the name Simda. We designed the worm to have 5 different propagation vectors with the worm choosing a different propagation vector on each domain depending on how that domain was first infected. The primary propagation vectors of Simda are

- **Email.** The worm has a mass-mailing component where the worm sends out bulk emails to a random subset of users at the rate of 8 emails per minute. We varied this value for various experiments with similar results.

- **File Infection.** The file infection component of the worm infects the local file system on the computer that it infects. The frequency of infection through this propagation vector is the lowest.

- **Web-server.** The webserver component infects all the files that can be accessed through the web. Anyone accessing the files would get infected by the worm.
• **DDoS.** The Worm opens up the machine to be used as part of a distributed DoS attack.

• **Scanner.** The worm installs programs on the machine which use the network connection to scan for vulnerable machines and infect them. The network scanning rate for the worm was fixed at three scans per minute. Typical worm outbreaks however have come with much higher scanning rates.

### 7.3 Prototype Implementations and Results

We tested the performance of our signature generation scheme using an artificially simulated environment with 100 domains and a 5000 individual clients. The simulation was setup in an enhanced version of the simulator that we used for Mail-trap. We modeled individual client behavior as well as varying propagation rates for our synthetic worm. We also modeled various different cooperation strategies which includes, the degree of cooperation and the frequency of interaction.

The first set of experiments shows the spread of the worm in our simulation. This set of experiments was performed in order to show that the resulting growth chart matches the exponential growth of existing worms. Figure 7.4 shows the spread of the worm across the network topology that we created. The spread of the worm as we noticed is much quicker than the standard ‘S’ curve spread shown by regular worms in the same time period. The worm is able to overwhelm the entire population within minutes. This result is in line with predicted and observed worm propagation trends for very fast moving worms. The spread of the multi-modal version of the worm is orders of magnitude quicker in spreading through the infrastructure simply because of its extra propagation vectors. It also provides a unique perspective of the worm to each of the domains that experience the worm. We simulated an anomaly based detection scheme in the simulator. Each client was given a unique usage
profile and anomalous network behavior was considered as malicious. We assumed zero-false positives in the simulation. We also programmed Simda to display only one propagation vector per infected client and even then, the worm only displayed a fraction of that particular propagation behavior. We modeled the variation in the signature created at each domain through this mechanism. Each infected client observed only a percentage of the behavior of the worm. Therefore, the signature of the worm generated by the domain only accounted for a percentage of the overall behavior. This can also be taken as the accuracy percentage of the signature.

We conducted a series of experiments to determine measure the performance of our behavior monitoring scheme where every domain was relying on local information alone. The idea was to compare the spread of a pathogen in this situation with the scenario where people were sharing the alerts. The results from the experiment are summarized in figure 7.5. We can see from the figure that for highly polymorphic worms (10% accuracy of signature), the total number of infections was close to 90%
of the total population. For 100% accurate signatures, the number of infections was a much more reasonable 25%. However, comparing this with the scenario where everyone was cooperating in the alert sharing and forensics gathering, our simulated results showed a marked increase in performance. the total number of infections was 6% even for highly polymorphic worms as opposed to 90%. If the accuracy of the signature was high, the total number of infections was less than 1%. The results on our simulator platform clearly showed that relying on local information was not a good idea in the case of a multi-modal worm. We would also argue that for higher population sizes, our simulation had a total population of 5000, the threats would be detected quicker, leading to even lower infection rates.

The second set of experiments was conducted to evaluate the effect of cooperation on the generation of a global signature of the threat. We classify the overall population as either ‘vulnerable’ or ‘immune’ to the Simda and track the number of vulnerable nodes in the system through varying degrees of cooperation between domains. We define an immune node as one that has witnessed a propagation vector either itself or through collaboration with another domain. The results here are significant in the sense that as the level of cooperation increases, the spread of the worm through the system slows down considerably specially since the number of immune nodes increases. The increase in the number of immune nodes also signifies a decrease in the vulnerable population.
The results depicted in the figures 7.6, 7.7, 7.8, 7.9 and 7.10 summarize the experiments conducted with varying degrees of cooperation between the domains and signature accuracy. For a 100% signature detection combined with full cooperation across domains, the total number of infected machines was well below 50. For lower levels of signature accuracy, the effect of the total number of infected machines was higher, however, increased level of cooperation greatly improved the effectiveness of the scheme.

The simulator highlights an interesting aspect of the multi-modal worm. While no protections in place, the worm was able to overwhelm the total population quickly. As the speed of propagation increases, the worm also becomes more noisy and hence easier to detect using behavior monitoring schemes. However, the system would be vulnerable against an extremely fast worm that overwhelmed the system before it was detected through local behavior monitoring. This is an active area of research.
that we need to explore. So while a uni-modal worm such as email took longer to propagate, in some ways, it was easier to hide the malicious email behavior than the behavior of the multi-modal worm. Even when the signature accuracy was low, the domains acquired the complete signature quicker due to the fact that with increased infections, the domains got multiple views of the worm. Aggregating the individual views across administrative boundaries helped reduce the overall infections quicker. It also helped individual domains acquire immunity about propagation vectors that they had not seen before.

**Figure 7.7:** Simda Spread, Signature accuracy 20%
**Figure 7.8**: Simda Spread, Signature accuracy 40%
Figure 7.9: Simda Spread, Signature accuracy 80%
Figure 7.10: Simda Spread, Signature accuracy 100%
The EWTF (Early Warning Task Force) was setup as a federal initiative to improve the sharing, integration and dissemination of information about cyber security threat, vulnerabilities, exploits and incidents at within a vetted trust community. The stated goals of the project are to improve warning and response to incidents, to increase coordination of response information, reduce vulnerabilities and enhance prevention and protection efforts. EWAN (Early Warning Alert Network) aims to provide channels of information for just such an exchange. The design of EWAN includes mechanisms to provide 1) Situational information on a daily basis. 2) The alert information channel and 3) An analysis communication channel for analysts to exchange and coordinate analysis with CERT and National Crisis Coordination Center. EWAN is designed as a many to many system with information flows back and forth between participants on a voluntary basis. Our system is designed to complement rather than compete with or replace existing Early Warning Systems.

ProofPoint’s Risk Management Software [74] uses statistical models and proprietary machine learning technology to check policy violations in outgoing traffic. The
system claims to be successful in countering viruses, spam and other email borne attacks as well as stopping corporate email policy violations.

CounterPane [15] correlates warnings by correlating events from earlier time zones and alerting IT teams hours before such threats hit the west coast for example. Devices on the network called sentries collect, sort, correlate, and analyze data from devices on the network such as firewalls, intrusion detection systems, routers, and servers. This data is continually forwarded to an analysts at counterPane’s headquarters. This data is also correlated against vendor databases for vulnerabilities and threats. This application provided by counter pane essentially has the same idea that cooperation can provide significant benefits.

DShield[88] is an attempt to collect data from all over the Internet and can be used to discover trends and prepare better firewall rules. Users report their logs etc to the centralized databases. ForeScout’s Global Early Warning System (GEWS)[29] aims to aggressively seek out threats before they have a chance to strike. GEWS is a subscriber based solution that provides users with real-time information about hostile network sources that pose a potential threat. The system watches for suspect reconnaissance activity and feeds bogus information to programs collecting system data. The system then traps programs utilizing this bogus information thus reducing false positives considerably.

WildList is an organization consisting of virus information professionals who report malicious programs to a common repository. However, the warnings generated by Wildlist would be very slow to counter a fast-moving threat.

Symantec’s Deep Sight Threat Management system[87] compiles data from symantec partner sites world-wide in order to get a more comprehensive view of internet security world-wide. The alert mechanism provides administrators an early warning for threats like the Blaster worm. Deep Sight has evolved from being a collector of
IDS data to cover different types of internet threats including data from firewalls and antivirus systems. Deep Sight is sold as an annual subscription service.

AVIEN and AVIEWS [2][36] are international on-line communities dedicated to a cooperative, grassroots information sharing effort to reduce the impact of malicious code (viruses, worms, Trojan Horses, Spy-ware) and other related vulnerabilities.

Viewed in their most basic context, Foresight’s sensor design and communication protocols provide the same functionality as the (SNMP) Simple Network Management Protocol. SNMP is a set of application layer protocols for monitoring network-attached devices and exchanging management information between them. SNMP works by sending messages, called protocol data units (PDUs), to different parts of a network. Agents, store data about SNMP managed devices in Management Information Bases (MIBs) and return this data to the SNMP requesters. Unfortunately however, SNMP was designed in an era where most computer users were benign and hence it was not designed as a secure protocol.

Worms and viruses have traditionally been modeled as biological viruses because of their self-replicating and propagation behavior. A detailed analysis of an epidemiological model of a worm is provided by ???. They also model the effects of dynamic quarantine on the propagation of a threat. Kephart [40] characterize viral infections in terms of birth-rates and death rates. Wang et al. [95] study the effects of partial immunization in throttling an epidemic and claim that even random immunization can retard the virus growth. They use mathematical modeling and a simulated test-bed to analyze viral propagation in networks. A key result of research in this area is that while viruses propagate through the network, relatively low levels of immunization can slow the infection significantly. The results obtained in our results suggest a similar trend when facing advanced worms. Even if the immunization is not able to stop the growth of the worm, partial immunization slows down the pathogen considerably.
There is little work done in evaluating the claims of schemes that claim to retard the growth and epidemic spread of viruses. The Swarm simulation package has been used by [65] in order to simulate interactive agents and worm mitigation strategies. use a mathematical model in which compromised sites warm ‘friends’ of the presence of a worm. In their scheme, edge routers share information with a small set of other edge routers. The edge routers can decide what to do with the particular information given to them. This scheme in its essence is closest to ours but their results on the SWARM simulator show that they are unable to retard fast moving threats. They use a simplified flat network topology in order to test their hypothesis. SWARM is a software package for multi-agent simulations of complex systems like population dynamics and simple social behavior in biological organisms.

DIDS [86] features distributed monitoring and data reduction with centralized data analysis. A centralized system suffers from traffic bottlenecks besides being a single point of failure.

IDES [37] and EMERALD [72] are intrusion detection systems coming from the SRI laboratory. IDES monitors different usage parameters for users, remote hosts and target systems. A profile of expected behavior and standard deviations is thus kept as numerical values. Each audit record is checked for anomalies at entry. The IDES project transformed into EMERALD. EMERALD operates at three different levels within an enterprise, service level, domain level and enterprise wide level. A monitor is dynamically employed throughout the system to monitor points of interest. Communication between monitors happens through a publish/subscribe mechanism. Emerald builds a hierarchy of monitors that can detect large-scale attacks against enterprise wide networks. The heart of monitors is an expert system that merges date from profilers, signature based engines and peering monitors to decide responses. MIDAS is a rule based expert system that uses domain-knowledge and symbolic reasoning to perform intrusion detection.
Netbait[14] is essentially a distributed query processing system over intrusion detection system data. It is a post-mortem analysis of the epidemic spread of well-known viruses and worms through the Internet. In principle, data from many geographically dispersed machines can be aggregated in order to have a global view of the infections. The system has a working prototype on the Plant-Lab test-bed. Our system in contrast to netbait would be able to identify and retard unknown threats in the real time.

The goal of Backtracker[44] is to try and identify automatically the sequence of steps that occurred in an intrusion. Starting with a suspicious point, backtracker identifies the files and processes that could have effected the detection point and displays the results in a dependency graph. Such a dependency graph allow others to avoid a similar scenario.

[41] introduce the concept of using internal sensors to perform ID. Internal sensors have the advantage of being able to obtain data at source with reasonable CPU and size overhead. Also they can look for specific attack conditions instead of reporting generic data for analysis. This means that the volume of data generated is orders of magnitude smaller. However their scheme requires having access to the source code of the operating system and its programs.

Moore et al. [58] have proposed black-listing of infected nodes and filtering of connections based on attack signatures. Zou et al [104] explore the possibility of temporary blocking of traffic on a host suspected of being infected. Their results claim to retard a worm propagation sufficiently so that it gives networks more time to fight against the threat. Commercial devices like [97] divide the network into many separate subnetworks and block traffic between them in an effort to contain worm behavior.

Various techniques have been proposed to help detect malicious email viruses. Virtual environments such as honeypots [33] have been used to observe the behavior
of suspicious emails on virtual hosts. Zou et al [103] propose a feedback email worm defense system to protect emails users in an enter prise network. Their system uses honeypots [33] to detect threatening emails received by email servers. An incoming mail has to pass through several detection procedures in order to determine whether it is a worm or not. Each email is assigned a detection score and an appropriate defensive action taken which varies from labeling of the email to aggressively deleting it from the server. They propose using a differential email service in order to prioritize the speedy delivery of emails based on how ‘dangerous’ they are.

Balzer [3] has proposed to monitor the run-time behavior of opened email attachments to ensure that these processes don’t do anything harmful. This technique can be used to detect and quarantine malicious activity. Under this technique, the system is divided up into four resources, the file-system, system registry, inter-host communication and process spawning. If the wrapper notices such activity happening, it can apply a stringent set of rules to ensure early warning of a problem.

Gupta [31] have used network traffic anomaly detection to detect the presence of email worms in the Internet. The approach detects an increase in the email traffic in comparison to a training period by monitoring email traffic between clients and servers in an organization. The MET [18] system assumes that an email worm has exactly the same MD5 sum. There technique uses a trusted third party to collect and disseminate activity noticed at various geographically dispersed mail servers. In a way this scheme is similar to commercial information dissemination services, the only difference being the automation of the mitigation steps at the mail-servers. The reliance on MD5 sums is an assumption that fails in the face of polymorphic threats.

Kuperman and Spafford [45] propose the use of library call interposition for intrusion detection. On a unix like operating system, this could be implemented using an
like library interposition. The interposition agent could log system calls to whatever granularity and could be used for threat specific intrusion detection. [41] introduce the concept of using internal sensors to perform Intrusion Detection. However their scheme requires having access to the source code of the operating system and its programs.
Conclusions and future work
This work addresses some of the unique problems faced when countering zero-day multi-modal pathogens. We presented cooperation between domains as a possible means of creating a better view of an unknown threat. We created a framework to facilitate this sharing while addressing the inherent security concerns involved.

The thesis of this dissertation is that sharing alerts and forensics information between cooperating domains is beneficial in countering newer pathogens. This dissertation proposes a framework to facilitate cooperation between domains in order to benefit from each other’s experiences. The idea is that each domain might have an incomplete, approximate or inaccurate picture of a developing threat. The framework also utilizes an incentive based cooperative policing scheme to reduce the effect of attack traffic on the overall system. Active cooperation between domains lets them create a better view of an emerging threat. This in turn enables them to slow down the epidemic spread through timely and appropriate immunization. The thesis proposes an interaction based reputation scheme to manage the trust issues. The risk of compromising the integrity of the entire system is amortized over the integrity of a set of entities in the system. No single node can be compromised in a manner that
is catastrophic for the system. We presented a security interaction based reputation scheme to drive our access control architecture. We simulated the performance of the scheme in a multi-agent environment. Our simulation results suggest that our scheme was able to detect non-cooperation from domains and was resilient to misbehavior. Reputation sharing enabled us to quickly identify rogue domains in the system. The scheme allowed us to build trust conservatively, misbehavior was punished swiftly so that a domain that carefully establishes trust in order to attack at a later stage could be denied.

While our framework allows sharing of all kinds of security related data, we evaluated the merits of our scheme in three different contexts.

10.0.1 Email based malware

We adopted an anomaly based network behavior monitoring approach to detect new pathogens. Having established malicious behavior through this scheme, we used the Foresight architecture to create a generalized footprint of the threat and share it with a trusted community. Performance evaluation of the scheme based on a real life trace-based simulator setup showed significant gains through cooperation. The scheme was successful in reducing the overall number of infections drastically. Our results showed a significant drop in the total attack traffic through the use of a cooperative policing mechanism. The domains were better able to protect against pathogens displaying polymorphic behavior. While the success of the signature generation scheme was dependent on the percentage of cooperating domains, the simulation results showed that Mail-trap was successful in slowing down fast moving worms even under partial cooperation. In such a situation, the vaccinated domains acted as fire-lines in slowing the growth of the pathogen. The anomaly based detection scheme was unable to tackle very slow moving or stealth worms that deliberately try to model normal or good traffic. Our evaluation methodology illustrated that the Foresight architecture
enables better signature generation of developing threats through aggregation of forensics from different vantage points.

10.0.2 Distributed denial of service attacks

We adopted an anomaly based detection methodology to mitigate DDoS attacks. We adopted a two-step detection scheme to reduce the overall number of false alarms generated by our scheme. We utilized the Foresight architecture to push filters closer to the sources of misbehavior. Trace-based simulation results showed that our scheme was successful in countering SYN-Flood attack as well as reflected attacks. Attacks with randomly spoofed source IP addresses were difficult to handle. We presented and analyzed two profile based cooperative policing schemes to reduce the total amount of attack traffic reaching the servers. Simulation results showed that after an initial drop in connectivity, the scheme was quickly able to recover the good traffic directed at the server. The success of the scheme was dependent on the degree of cooperation between domains. Simulated results suggested that cooperating domains were able to receive a better quality of service during attack conditions. This provided an incentive to cooperate in order to achieve the collective goal. The results presented by our scheme were limited by the use of a simulation environment for verification. While real life data collected from a single administrative system was used to parameterize the simulator, we would like to evaluate the scheme in a real life setting. We are currently working on a collaborative effort to collect and correlate data from multiple domains in order to create a better trace for the simulation setup.

10.0.3 Multi-modal worms

We proposed a multi-modal signature scheme to represent a multi-modal pathogen. We observed that multi-modal polymorphic worms display varying characteristics on different domains. Each domain thus has a unique perspective of a growing threat.
based on its local observations. We used a simulation setup to show that relying on local information alone is not good enough for such threats. We simulated a multi-modal worm and evaluated the performance of Foresight in handling such a threat. The scheme exploits the heterogeneity in the sensors adopted at each domain to create a better picture of a multi-modal, polymorphic threat. Our results in this context showed that domains were able to gain from the experience of cooperating neighbors to acquire immunity towards unknown and unseen propagation vectors.

10.0.4 Incentive to Cooperate

“Great rewards will come to those who can live together, learn together, work together, forge new ties that bind together”

We evaluated the performance of an interaction based reputation scheme to detection lack of cooperation and misbehavior among domains. This is a significant issue when it comes to sharing security related data. Lack of cooperation from domains can be isolated through the reputation scheme and rogue domains can be identified. The access control protocol can then be used to lock such domains out of the collective.

We show through our experimental results that there is a significant class of attacks where cooperation and reciprocity are more effective in achieving the collective goal of safer computing. Our experimental results in the context of Mail-trap and SIMDA suggest that cooperation leads to a better view of a polymorphic multimodal threat. Domains can acquire immunity towards unknown or unseen threats. Till now, this has largely been a manual process. Domains rely on security updates from security companies, intrusion detection systems as well as local observations of data by system administrators to detect misbehavior and to track it. Given the clean-up costs involved in dealing with such incidents after they have happened as
well as the demonstrable benefits provided by the Foresight architecture, we would argue there would be an incentive to cooperate.

Security and privacy concerns typically prohibit forensics sharing. We minimize the concerns by only sharing information that is already available. For instance, the email header information used in Mail-trap is already available at the mail servers that the message passes through.

Cooperative policing mechanisms in the context of a distributed denial of service attack have a built in incentive to cooperate. DDoS attacks typically result in an increase in the overall congestion in the networks. This effect on the overall network can be reduced by pushing filters as far back as possible. One of the main reasons such a defense is not in place is because of the pricing structures on the internet. Backbone traffic is charged at a per packet basis and thus there hasn’t been a push towards a cooperative policing mechanism. There is however incentive for domains to stop local misbehavior that ends up costing them. In the event of a massively distributed attack where the local misbehavior does not raise local alarms, the misbehavior is still visible at the attacked domain. This architecture enables participating domains to identify such misbehavior remotely and to pass the message along to the local Knowledge Manager.

Our simulation based results also showed that the domains that cooperated towards the collective goal of blocking local misbehavior and rate limiting attack traffic were able to get a better quality of traffic. This scheme actually has a viable business model where cooperating domains can pre-negotiate service level agreements whereby they get preferential treatment during attack conditions. Non-cooperating domains receive indiscriminate packet losses. Our scheme relies on a combination of components already present in most networks so there is very little infrastructure cost involved.
10.0.5 Future Work

This dissertation demonstrates that there is a need to securely and efficiently share security threat related data. However, we also discovered interesting areas of further exploration. There is a huge body of work in the worm fighting area. However, there is a shortage of verifiable verification infrastructure. Most research groups have resorted to writing their own simulation code. Each such effort comes with its own set of assumptions, client access patterns, network topology, worm characteristics etc. There is a need to have a common platform through which comparative studies can be performed on various competing directions of work.

We adopted various anomaly based detection schemes to detect misbehavior in the various instances of Foresight that we described. Anomaly based schemes have always suffered from false positives. They also fail to provide a context to the anomaly generated. While our scheme does detect anomalous behavior, it provides little context as to why that anomaly has happened. We kept very little functionality in our host based sensors. We think that a combination of host and network based approaches is best suited to detect misbehavior. Stealth worms would also create a problem with the current detection schemes. If a worm deliberately tries to resemble normal traffic patterns, the detection schemes would fail. This is still an open research question. Various different detection schemes are suited to different types of pathogens and alternative detection mechanisms need to be explored. We do contend however that whatever the actual detection scheme, Foresight can still be used to share the data to exploit the advantages associated with sharing forensics.

Published work has referred to Warhol worms or flash worms that would overwhelm the Internet within minutes. Our data sharing mechanism would fail in the face of an attack that would overwhelm the targets even before local alarms were raised. The quick detection of a worm is therefore significantly important to the
success of our data sharing mechanism. Once an alert has been generated, the propagation of the alert through the community can be done faster than the worm propagation rates. Achieving quicker detection rates for worms is an active area of research that needs to be explored further.
Bibliography


Biography

- Name: Muhammad Fareed Zaffar
- Date of Birth: 16th October 1977
- Place of Birth: Lahore, Pakistan
- Degrees received: Bsc, Computer Science
  Lahore University of Management Sciences, Pakistan