

Lightweight Change Detection and Response
Inspired by Biological Systems

By

VINOD BALACHANDRAN
B.Tech. (Anna University, India) 2011

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Prof. Sean Peisert, Chair

Prof. Karl Levitt

Prof. Matt Bishop

Committee in Charge

2013

Copyright © 2013 by
Vinod Balachandran
All rights reserved.

*To everyone who made this ...
a meaningful work.*

CONTENTS

List of Figures	v
List of Tables	vi
Abstract	vii
Acknowledgments	viii
1 Introduction	1
1.1 Problem Statement	2
1.2 Thesis Statement	2
1.3 Approach	2
1.4 Definitions	3
1.5 Organization of the Thesis	4
2 Background	5
2.1 Intrusion Detection	5
2.1.1 Challenges for network-based IDS	6
2.1.2 Hybrid coordinated approach	8
2.2 Response Systems	8
2.2.1 Intrusion Prevention vs. Intrusion Response Systems	8
2.2.2 Automated Response	9
3 Lightweight Monitoring and Intrusion Detection	10
3.1 Motivation	10
3.2 Desired Features	10
3.3 Existing models	12
3.3.1 Lightweight network IDS	12
3.3.2 Distributed monitoring systems	13
3.3.3 Agent based event monitors	14
3.3.4 Challenges	15

4	Automated Response	16
4.1	Motivation	16
4.2	Evolution of automatic response mechanisms	18
4.3	Taxonomy of existing response systems	18
4.4	Desired Features in Lightweight Distributed Response Systems	23
5	The Hive Mind: Lightweight Distributed Event Monitor	25
5.1	Inspiration from Biological Ant Foraging	26
5.2	Characteristics of the Hive Mind	26
5.3	Theory	27
5.3.1	Automated Response	30
6	Experiments and Results	32
6.1	Prerequisites	32
6.2	Experiments on the Hive Mind	33
6.2.1	Coverage	34
6.2.2	Detection time	37
6.2.3	Pheromone trail length	38
6.3	Scenario-Based Study	39
6.3.1	Data Exfiltration Scenario	39
6.3.2	Sensor tasks	40
6.3.3	Response	41
6.4	Hierarchical Bloom Filters	42
6.4.1	Bloom filters	42
6.4.2	Comparison to conventional methods	43
6.4.3	Potential Optimizations for hierarchical filters	45
7	Conclusion and Future work	46
7.1	Summary	46
7.2	Future Work	47

LIST OF FIGURES

1.1	The Hive Mind Venn Diagram	3
4.1	Automated response taxonomy	20
5.1	A typical segment of a Hive	29
5.2	An example of pheromone trail	30
6.1	Coverage by 1 ant	35
6.2	Coverage (in time) by 1 ant	35
6.3	Coverage by multiple ants	36
6.4	Coverage (in time) by multiple ants	36
6.5	Coverage with increase in Ant count	37
6.6	Detection time and Coverage time	38

LIST OF TABLES

4.1	Comparison of different response systems	21
6.1	Detection results for varying pheromone trail length	39
6.2	Comparison between Manual and Automated response	42
6.3	Comparison between conventional hashing and Bloom Filters	43
6.4	Comparison between conventional Traditional and Hierarchical Bloom Filters	45

ABSTRACT OF THE THESIS

Lightweight Change Detection and Response Inspired by Biological Systems

The state of computer security is complex. With computers taking multiple forms including such lightweight devices as smartphones and virtual machines and then connecting these devices to the open Internet, the task of securing devices become harder. To attempt to provide protection from threats it is a common practice to install Security Event Monitors. In this thesis, we present a lightweight host-based security event monitoring and response system called the Hive Mind that is designed to enable coordination among participating nodes for improved detection combined with reduced resource usage. We also present a model for automatic response in such lightweight systems. The Hive Mind is a host-based security event monitor (SEM), a system that monitors intermittently for potential threats and indirectly communicates the existence of a problem to other nodes using a stigmergic approach inspired from biological systems. When we apply the system on example scenarios, the results demonstrate that the Hive Mind system is consistent with the theory it is built on.

ACKNOWLEDGMENTS

This thesis is a result of generous support from many people throughout my life. I would like to thank my advisor and mentor Prof. Sean Peisert for sharing his valuable time over the last 2 years. He has always given me hope and support during difficult times. I appreciate his interest, support and guidance on all the important decisions in my Master's program and beyond. I would also like to thank my advisors Prof. Karl Levitt and Prof. Matt Bishop for accepting to be a part of my thesis committee and providing their guidance throughout the process. Special thanks to Prof. Levitt for taking personal interest in my career and supporting me in every possible way.

From the bottom of my heart, I thank my project colleague and lead software architect of Hive Mind, Steven Templeton, for spending countless hours providing lessons and advice on every single aspect of my research. He has never failed to extend a helping hand during challenging moments in my research. Special thanks to past and current members of the Hive Mind team who have given me a great platform to work on. I would also like to thank the DETERlab team for their wonderful testbed that made several of my large scale experiments possible. I would like to thank Steven again for his ideas and insights on Hierarchical Bloom Filters.

This research was supported by the National Science Foundation and the GENI Project Office under Grant Number CNS-0940805. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of any of the sponsors of this work.

Chapter 1

Introduction

The state of computer security is complicated. It is even hard to determine whether a system is in a “better” or “worse” state from one second to another. An FBI study [FBI] conducted a few years back stated that there was a decline in financial loss from computer security breaches and an increase in organizations’ interest in security audits. It is also important to point out that new security issues arise constantly and new computing platforms continue to be created. Devices that need to be protected from threats now come with a range of capabilities and therefore new threats. It is important to understand the system that needs protection in order to determine an effective solution. To ensure protection from several threats it is a common practice to install Security Event Monitors (SEM). A security event monitor surveys the network or the system it is deployed on for anomalous and/or potentially malicious activities based on predefined policies or statistical profiling and alerts the administrator when a policy violation or a statistical aberration is found. An Intrusion Detection System (IDS) is a type of security event monitor that looks for unauthorized intervention or presence in the deployed system. An IDS deployed on a host is referred to as host-based IDS and monitors system activity local to the host. An IDS deployed on a network’s access point is called a network-based IDS and monitors all the traffic entering and exiting the network. Both types of IDSes have disadvantages [KV02] that make their value increasingly questionable.

1.1 Problem Statement

With computers taking multiple forms including lightweight versions such as smartphones and virtual machines all connected to the open Internet, the task of securing devices become harder. We can no longer deploy processor-heavy host-based intrusion detection/prevention systems on such devices because those monitoring systems often require so many resources that the system cannot even perform its primary function. The alternative, network-based monitors, are not affected by the variations in device platforms or speed but suffer from the inability to process encrypted traffic. To overcome the disadvantages of both, we describe means for leveraging and extending a lightweight event monitor, the Hive Mind, that is tunable, resource-conscious and also retains many of the benefits of existing systems.

1.2 Thesis Statement

In this thesis, we present an application of the Hive Mind, a lightweight host-based security event monitoring and response system that is designed to enable coordination among participating nodes for improved detection and reduced resource usage. We also present a model for automatic response in such lightweight systems.

1.3 Approach

The Hive Mind is a general-purpose event monitor although we have developed it particularly with security and performance monitoring in mind. Figure 1.1 gives a clear representation of the Hive Mind's classification. Here, we use the Hive Mind as an host-based security event monitor (SEM), a system that monitors intermittently for potential threats and indirectly communicates the existence of a problem to other nodes (unlike traditional host based security event monitors) using a stigmergic approach inspired from biological systems. Our hypothesis is that the indirect coordination among participating nodes will improve the process of detection without overly reducing the productivity of the nodes.

Most intrusion detection systems respond to detecting an anomaly by logging or reporting. This might allow the problem to expand inside the network, compromise the device or steal more useful time from users before any action is taken to mitigate it. Therefore, we try to invoke automatic responses wherever possible and analyze the impact of those

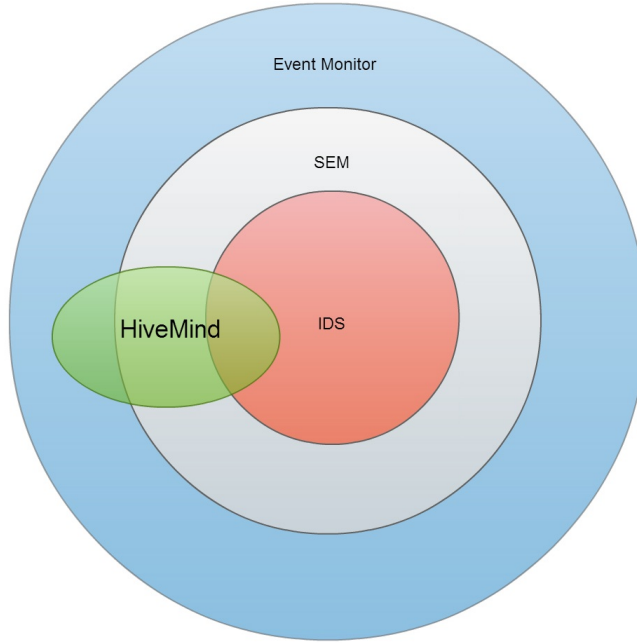


Figure 1.1. Venn Diagram representation of the Hive Mind's classification

responses on the usability of the system. At times, automatic responses can greatly degrade the usability of the system with an increase in the false positive rate of detection, so we entertain the possibility of dynamic and *optimistic* responses [PB13] in our design to reduce the impact as much as possible. We believe that a lightweight system can complement existing network intrusion detection systems while impacting performance less (including CPU cycles, space) than a complete host-based intrusion detection system deployed in every node.

1.4 Definitions

We define a *response* to be the action taken after detection of a potential threat. Responses are said to be *automated* if there is no human intervention between the detection and response phase. The term *autonomous response* is mainly used in cases where the response is taken by the same entity that is used for detection. In the Hive Mind, a *hive* is the collection of *nodes* participating in the monitoring process. The hive also includes the structure of the neighborhood network (set of neighboring entities to nodes chosen based on a application-specific metric) of each node thereby forming a map. The nodes are individual entities that can run software to monitor and protect themselves autonomously and also coordinate with other nodes through sending and receiving control messages from other nodes. We define

Ants as the virtual agents inspired from biological *ants* (real life social insects). *Ants* are the agents that travel between nodes in the hive to carry control information and execute *tasks*. Here, tasks are functions that are used to look for a smaller problem and try to report or rectify them when found.

1.5 Organization of the Thesis

This thesis is organized as follows: Chapter 2 discusses related work in the field of intrusion detection and response. Chapter 3 describes the need for lightweight monitoring, its desired features and existing models. Chapter 4 discusses automated responses and its taxonomy. Chapter 5 introduces the Hive Mind and its features and Chapter 6 presents the experiments run on the Hive Mind and their results. Finally, Chapter 7 presents our conclusions and suggestions for to future work.

Chapter 2

Background

The definition of security *events* is vague and so is the boundary between security event monitoring and intrusion detection. Most research in the field on intrusion detection also applies to the broader category of security event monitoring. We believe that security *events* only acquire meaningful definitions in the context of a particular environment. For example, transfer of data to an external IP address might be considered *normal* for a home user whereas the same can be considered a breach or more broadly a notable *event* for a company working with sensitive data. This same example qualifies as an *event* targeted by an IDS as well a threat monitored by a security event monitor. Keeping in mind the interchangeability in many cases of the terms, security event monitoring and intrusion detection system, we discuss related work in intrusion detection and automated response in this chapter.

2.1 Intrusion Detection

An *intrusion* is an attempt to compromise the confidentiality, integrity or availability of a system [Den87]. Intrusion detection systems can be considered to be a crude analogy to burglar alarms in real life. *Misuse-based* IDSs are designed to detect violations to predefined security policies. But things immediately get complicated with the introduction of “possibly malicious” behaviors which cannot be specified precisely ahead of time. An example would be developer in a firm doing large amount of file transfer in a short span of time. This could be a potential data exfiltration problem but might not be caught by access policies because he is allowed to transfer files. Statistical *anomaly detection* was introduced for this particular

reason where a profile of a user or a system is created and any deviations from the profile is reported. While both the type of systems are useful independently, a hybrid of both can reduce, but not eliminate, the individual disadvantages [HK88].

An important factor that defines the type of implementation an IDS adopts is the source of audit data [Axe00]. The two main sources are host-based logs that host-based IDSs work with and data packets flowing in a network that are tapped by network-based IDSs. The host logs can be kernel logs [OB01], application logs [PBKM07a] or device-related logs.

There are several issues with both host-based IDSs and network-based IDSs. They include:

- Heterogeneous operating systems making enumerating system-specific detection parameters for each system extremely time consuming
- Increased number of critical nodes in the network increasing the performance overhead
- Performance degradation in the host system due to additional activities for security such as logging
- Difficulty in recognizing network-wide attacks
- Hosts with insufficient computational capability to deploy a complete host-based IDS

In contrast, network-based intrusion detection systems can have a central system with a network tap to passively monitor traffic in the network [MHL94]. They do not affect the system performance and can detect network-wide attacks easily when installed at the network boundary. The implementation of network-based IDSs is very straightforward. Host-based IDSs in a network of mission-critical, performance oriented hosts have to be carefully chosen so they do not overly limit performance in each system.

2.1.1 Challenges for network-based IDS

With advances in network technology, network-based IDSs face some of the following issues.

Increased network traffic

Recent requirements and developments in computer applications have increased the network traffic exchanged between hosts to new heights. The network based IDS must cope with continuous processing and very high load.

Reduced Latency

Not only are more packets transmitted, but they are exchanged at faster rates now. This also poses a challenge to the IDS to collect and process data quickly in order to avoid delay and storage challenges.

The problems mentioned above are inevitable because of the evolution of computer networks and many can be overcome by “throwing more money” at them. Architecture and technology to process information faster can be used to mitigate the effects of many of these challenges. However, although the problems look trivial, sometimes they pose great challenges. Here, lightweight host-based monitors may be useful. Another challenge is that network based IDSs aren’t necessarily deployed ubiquitously. They are often only at the network boundary. Some organizations put them throughout the organization at other critical internal boundaries but that then introduces some of the same issues of coordinating network-wide attacks as host-based IDSs.

Encryption

The biggest threat for network-based IDS is the encryption of payloads in network packets [WS04] and it might even spark the question “*Is network-based IDS useful anymore?*” despite all its advantages. End-to-end encryption of traffic is increasingly becoming an integral part of several important applications like SSH (Secure SHell) and SSL (Secure Socket Layer) driven by VPNs and e-commerce. Because encryption by itself is a security mechanism and is being used increasingly, we cannot ignore this challenge.

The goal of encrypting payloads is to make the contents unreadable to everyone else other than the intended recipient or to be precise the one who holds the key to decrypt it. This makes sure that no “middle man” listens to the packet and steals/misuses the data. Encryption is crucial in cases like transmission of credit card details, health records and so on. Unfortunately, it also makes it impossible for the trusted network-based IDS to scan the payload content of the traffic. It is clear that encryption restricts man-in-the-middle

attacks but one still has to worry about the possibility that the source of the traffic might be malicious. It is typically impossible for the network IDS to see if the source is trying to compromise the receiver. The attacker might send well-crafted malicious command sequences and encrypt them making the network IDS oblivious to their presence.

2.1.2 Hybrid coordinated approach

Considering the challenges network-based IDSs face, it is clear that a hybrid approach would serve better [Axe00]. Especially with lightweight systems it is not feasible to run a complete host-based IDS on all entities or ignore the limitations of network-based IDS. However, it is useful to leverage the positives from both systems by trying to coordinate individual lightweight host-based monitors that run *on demand* and communicate problems and responses by directing resources. Although such an approach requires a strong yet fast communication protocol, it is a way to handle low power entities in a network.

2.2 Response Systems

2.2.1 Intrusion Prevention vs. Intrusion Response Systems

Intrusion prevention systems attempt to proactively stop certain type of intrusions from taking place. Unfortunately, such automated responses can often become infeasible due to imperfections in security systems and constraints like trade-off between security and user-friendliness or misused privileges. It is therefore important to focus more on response systems. A response is an evasive or corrective action taken against an intrusion or a potential breach. Some responses are automatic. For example, some systems will lock a user out of an account after a certain number of incorrect authentication attempts. Other responses are manual and are taken by security administrators after a suspicion that some kind of an intrusion is taken place. With increasingly complex intrusions and the rate in which they spread it is important to support the intrusion detection systems with a strong response system that does not itself rely on the IDS. Responses can range anywhere from notification and additional logging to blocking the user or the host [PB13]. Responses are classified on the basis of degree of automation, namely,

1. **Manual response** where administrator has a list of actions which he enforces, when

he is notified.

2. **Automatic response** where the system can trigger a response immediately after detecting a potential intrusion.

2.2.2 Automated Response

With the increasing speed and volume of attacks, notifications and manual responses are no longer effective and sufficient. Automation of responses become essential in present scenarios where the time required for attacker to cause damage is very minimal. According to Cohen [Coh99], the success rate of an intruder increases with an increase in the amount of time he is left undisturbed.

Automatic responses are classified, into *autonomous* or *cooperative*, based on the point of decision-making for a response and thereby an Intrusion Response System's ability to communicate between entities or lack thereof. Autonomous responses are independent and executed locally. The more flexible cooperative response can also respond locally with only the final strategy being determined globally. This offers better response speed and can potentially contain the volume of damage. The downside for this is the need of a strong communication and coordination mechanism that adds a significant overhead. Automatic response can be either be static or dynamic based on the response selection strategy. Several papers [LFM⁺02] [FWM⁺05] have also been on cost-sensitive response mapping where the impact of damages are analyzed and the best-suited response is taken. The data mining classification approach used by Lee et al. [LFM⁺02] shows from experiments that cost-sensitive response selection consistently lowers the overall cost where cost is defined by operational overhead and consequential costs. Foo et al. [FWM⁺05] conclude that improvement in automatic response selection strategy helps reduce the time taken to stop the attack. Automatic responses are categorized as static and adaptive based on the adjustability. An ideal intrusion response system should provide dynamic, adaptive, and cost-sensitive response [PB13]. These qualities help in devising a more appropriate response for a potential intrusion.

Chapter 3

Lightweight Monitoring and Intrusion Detection

3.1 Motivation

Security event monitoring systems and intrusion detection systems often demand significant amount of system resources to operate. With the increase in the use of low-power mobile devices, we require the monitoring system not to disrupt normal operations. Reduced use of resources has turned into a need rather than a luxury with the rise of such *lightweight* entities in the network. Components like mobile phones, control system sensors now share “networks” with other hosts but could not handle a complete IDS locally. Therefore, a need has emerged to design a lightweight system that is resource-conscious yet more efficient in comparison to its heavyweight counterparts.

3.2 Desired Features

The following list of characteristics defines a lightweight system according to Roesch et al. [R⁺99] and several others [HWH⁺03] [KGD08]. These features are,

- **Minimal disruption to the task:** The system should not negatively impact the actual tasks entities are supposed to carry out. Most lightweight devices operate in constrained environments and use most resources to perform a set of primary tasks. A lightweight intrusion detection system must not compete with the primary tasks for resources. It is ideal if the detection system is aware of the resources available at any

given instant, and can adjust accordingly.

- **Small system footprint:** The process running the lightweight IDS must aim at minimizing the number of CPU clock cycles it uses. Also, the size occupied in memory by the monitoring application should be very small. The application must strike the right balance while deciding the amount of log entries collected and stored in disk [Pei07, PBKM07b].
- **Distributed:** Lightweight systems must minimize resource use. Considering the amount and speed of information processed it is apparent that a centralized detection system should have processing capacity much higher than the participating entities. Also, the centralized detection system is a security, scalability and performance bottleneck. An alternative to this requires autonomous detection systems on participating entities. Each system needs coordination to leverage on the information other entities have gathered. This way, new threats can be identified at a faster pace compared to a completely non-coordinated approach. Therefore, it is important for monitoring and detection to be distributed to ensure balance and avoids a single point of failure.
- **Easily deployable on any system/network:** A lightweight monitoring system should be deployed with ease on any and all entities with minimal (ideally no) disruption to them. The entities here are usually host computers but can include other devices such as routers, control systems, etc.
- **Automatic response:** Since lightweight systems potentially have hundreds of hosts connected together manual response might not be scalable. Automatic response becomes necessary to limit the damage as early as possible. It is also important to avoid any additional communication overhead to implement a robust response system. Therefore, it is appropriate to include an automatic local response system.
- **Cross platform:** The system should be built using a technology that is portable and not restricted based on architecture or other factors.

3.3 Existing models

3.3.1 Lightweight network IDS

One of the early, successful and popular lightweight network intrusion detection system is Snort [R⁺99]. Snort can be considered structurally similar to tcpdump [R⁺99] [JLM⁺89] with added functionalities. Snort is a packet sniffer and logger that can be used as a lightweight network IDS. Snort does payload inspection on sniffed packets and looks for packets that match signatures of known attack patterns. Target patterns can be configured based on Snort rules. This simple “collect and process” technique makes Snort lightweight. Snort also provides the functionality to filter traffic based on commands allowing user to categorize traffic. The components of Snort (terminologies derived from [R⁺99] for consistency) can be broken into

- **Packet decoder:** Processes raw packets into packets organized around the layers of network protocol stack starting from data link layer up to application layer easy for the following components to use.
- **Detection Engine:** Detection rules are maintained as *rule chains* . Rule chains are two dimensional data structures. One dimension contains headers representing groups of signatures for a common source, destination and corresponding ports. The other dimension holds individual signatures. Every time a packet from the decoder comes into the detection engine, the rule chain is referred. Snort rules are often simple and require very little training. Basic Snort rules contain protocol, direction and port. The following rule mentions to record all traffic going to 10.1.1 /24 network for port 80(http). The *any* keywords are wild cards to indicate all traffic.

log tcp any any - > 10.1.1.0/24 80

- **Logging/Alert Subsystem:** This system logs packets into human readable format for fast analysis. Alerts can be sent to either syslog or displayed as a pop-ups. The following example rule from [R⁺99] shows how alerts are raised with human readable

messages defined by alert rule.

```
alert tcp any any -> 192.168.1.0/24 143 (content :  
    "|E8C0FFFFFF|/bin/sh";msg : "Buffer Overflow detected !";)
```

Snort is useful for several tasks including focused monitoring of critical nodes or services. Although Snort is lightweight it is not designed to monitor events happening on the hosts. Also, its design does not focus on capturing the global picture in case similar attacks occur at several hosts across the network.

3.3.2 Distributed monitoring systems

EMERALD

One of the notable works in distributed monitoring systems is EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances)[PN97] which was developed at SRI. It is a scalable tool designed to detect malicious activity across a large network. It uses a hierarchical approach covering misuse detection across various scales such as service analysis, domain-wide analysis and enterprise-wide analysis. Service Monitors that are distributed and autonomous are deployed to interact with the host as well as probe for additional information. Information can be correlated and disseminated to other service monitors thereby enabling coordination. EMERALD has a resolver to correlate the reports and implement a response based on it.

DIDS

The Distributed Intrusion Detection System developed at UC Davis [MHL94]. DIDS monitors a network using host monitors (one for each host), a LAN monitor and a DIDS director which acts as a center for information gathering, analysis and dissemination. The DIDS director is responsible for evaluation of individual reports which may be reports of potentially malicious events. It can ask each individual client for more details through a dedicated communication manager that reaches out to the hosts.

CIDF

The Common Intrusion Detection Framework [SCTS⁺98] is an effort to enable coordination between different intrusion detection systems as well as other complementary systems used

for event recording, logging etc. to communicate. The CIDF workgroup developed a language called Common Intrusion Specification Language defined to allow IDS systems to share intrusion data. CIDF was tested at several sites including UC Davis to collect intrusion data and send it IDS systems installed on other participants running unique systems on their premises. Although initial tests were successful [CID99], signatures need to be updated regularly and policies of competing commercial vendors might deter adaptation of this framework.

GrIDS

Graph-based Intrusion Detection System (GrIDS) [SCCC⁺] constructs activity graphs from data collected from hosts and network. GrIDS aims to detect large-scale coordinated attacks from the aggregated data collected. GrIDS allows administrators to define policies for individual hosts or a group. GrIDS reports violations of the stated policies. Initial tests were able to successfully detect a work attack [SCCC⁺].

3.3.3 Agent based event monitors

Although distributed detection is useful, all the above mentioned systems need to monitor activities in the hosts all the time running potentially complex logic. They are heavy and require significant effort to deploy and use. To ensure judicious use of resources most systems use distributed detection using mobile agents transported around the network between hosts. These agents typically are:

- fine-grained, carrying small logic,
- fast to transport, and
- updatable.

One of the early systems using distributed agent-based IDS is the AAFID (Autonomous Agent For Intrusion Detection) [CHSP00] developed at Purdue. It is developed to be cross-platform and flexible. The Computer Immunology project at the University of New Mexico [Hof99] also has small agents moving around looking for possible foreign agents (intrusions) and try resolving them. This gave rise to a design where set of system calls executed by the programs are observed and deviations from them are noted. This design does not focus on coordination of agents. The agents work individually. The JAM project at Columbia

University [SFL⁺00] uses distributed agents that implement learning algorithm to learn patterns of misuse. The authors used a financial institution example but do not discuss about responses in depth, likely due to the high sensitivity of information. Helmer et al. [HWHM98] have discussed lightweight agents which enables detection of correlated attacks on different hosts but does not consider response to the attacks.

3.3.4 Challenges

Coordination

The primary challenge with agent-based IDSs is to ensure effective coordination between participating agents with minimal use of resources. Inefficient coordination often causes repetition degrading productivity and tying up resources which are extremely critical in lightweight hosts. Although mobility of agents are dictated by the system's state at an instant, it is important to have an acceptable level of randomness to ensure all participating entities get "fair" coverage over time.

Agent "size"

Another important challenge is to understand the problem statement and the goals of agents involved. If an agent monitors for a problem that occurs rarely, it is wasteful to have the agent run its code frequently. Also, the nature of a problem determines the size of an agent's code and the number of agents needed to find evidence pointing to the existence of that particular problem. It is therefore important to build a flexible framework that allows users to define agents and tune its parameters based on the environment they work in.

Chapter 4

Automated Response

Intrusions have become sophisticated and widespread with advancements in network technologies and the reach of Internet. This suggests that it is not only important to detect an intrusion but also respond to thwart attacker's effort. Time taken to detect an intrusion and the time it takes to respond after the notification can be used by the attacker to deal further damage to the system. This emphasizes the need for responding as quickly as possible.

4.1 Motivation

Automatic responses reduce the delay between detection and response phases. A common example would be that, once a potential intrusion is detected, the automatic and immediate response can be to enable additional logging to gather more information. Later, an administrator can decide on additional responses with more precision. Completely automatic responses are not used due to high of rate of false positives in intrusion detection. Although this is true, several well known attacks can be thwarted immediately without the administrator "reinventing" the response. A typical example would be to remove permission from a particular executable that exactly matches the signature of known malware. Frequent attacks make life harder for an administrator to keep track of every attack manually although they were similar. Automatic responses are very effective in case of known software vulnerabilities or well-documented attacks.

On the flip side, there are practical concerns in implementing automatic responses

especially with anomaly based intrusion detection. Since these potential deviations from normal do not have written signatures, it is hard to determine the appropriate responses beforehand. Incremental responses are very useful in these cases. A typical example would be to choke the data transmission rate to the internet in the case of a huge deviation from the usual number of bytes transferred in a given time span. The bigger concern is that services, hosts and network links are often entangled in a complicated fashion and a response applied to one of these might affect others. The worst case would be a cascading effect on participating entities which may cause a self-inflicted Denial of Service (DOS) attack. Therefore, evaluation of how much damage an attack causes becomes important.

One must keep in mind that the cost of taking a response should never be more than the damage from an attack, or put the system in a position in which it is less secure than before the response was taken. There have been many methodologies proposed for evaluation of attacks including attack graphs [FWM⁺05] and resource dependency trees [BMRL03]. The speed of this evaluation will also be crucial in case of automatic responses. To determine the optimal response, the need for coordination arises in such interconnected systems. An example response would be to identify the source of an attack once an attack evidence is found in one of the affected systems. This requires the coordination of several systems (say sharing of log files) to trace back to the attacker and prevent further damage. A similar system called Intrusion Detection and Isolation Protocol (IDIP) [SDS00] was proposed for such coordinated efforts. Coordination is beneficial but also brings in a myriad of difficulties including insider threats, communication overhead and possible denial of service. Adding a certain amount of resilience to the response system, such as restoring a system back to normal in case of inappropriate response, can be useful.

Automatic responses take a new turn in case of distributed intrusion detection systems, especially ones that focus on keeping the detection as lightweight as possible. Since continuous monitoring and/or frequent communication are usually avoided to improve performance of the system, coordinated response becomes harder. Since there is no central orchestrator or resolver to put individual pieces together it poses an even larger challenge. Sharing “successful” response signatures can be useful in such cases.

4.2 Evolution of automatic response mechanisms

After recognizing that manual response is not sufficient to counter modern intrusions, software designers started constructing systems that used simple decision tables to determine a response for an identified attack. The decision tables use static mapping and are inflexible. Also with the increase in number of signatures and variations this system is not scalable. More complex mechanisms such as CSM (Cooperating Security Managers) [WFP96] and Emerald [PN97] use expert systems that mitigate limitations in static response mapping. These include severity and confidence metrics into the process. The severity metric rates how strong the response might be which in turn indicates potential negative effects in case legitimate users are affected. The confidence metrics include how certain the system is about the detection intrusion's validity and success rate of the response mechanism. These work well in case of independent actions but if there are actions against services that depend or affect other services, more effective means are needed to find the impact of the response. Methods like dependency-graphs [TK02] and attack graphs [SHJ⁺02] [NCR02] [FWM⁺05] try to model the impact of a response on the system or the network as a whole. Such metrics also allow the system to consider the risks involved in taking an action and the trade-off between actual damage and cost of response with time. Choosing various costs and weights for them is still a challenge and might depend on the nature of the system and the environment it is protecting.

4.3 Taxonomy of existing response systems

To build a useful response system it is necessary to analyze the behavior and features of existing systems. Below is a brief overview of existing taxonomies [SBW07b, SSEJJD12] on automated responses describing them in standard terminology. New classifications have been included and a few categories have been merged to present the overview concisely. Figure 4.1 summarizes the classifications.

Classification by adaptability

1. **Static/Non-adaptive responses:** Static responses are paired with the intrusion they respond to as long as the intrusion response system is deployed. They do not

depend on order of events. They are simple but suffer several disadvantages like lack of environment/state awareness, ignorance of future costs, and consequences of triggering them. Most early response systems, including EMERALD [PN97], were static and lacked adaptability.

2. **Adaptive responses:** The system with adaptive responses has the capacity to trigger the appropriate response based on several factors such as environment, response history, and confidence metrics. An adaptive response is often constructed after analyzing the current state of the system. An example would be to look for signs of other attacks and changing the response accordingly. Response models by Foo et. al [FWM⁺05] and Carver et. al [CHSP00] are examples of adaptive response strategies. Carver et al.'s model Adaptive Agent-based intrusion response system uses confidence metrics to fine tune future responses based on the previous response's success. In ADEPTS, Foo et. al's model, effectiveness of a metric is measured based on actions taken. ADEPTS is capable of automatic update based on effectiveness of a response in the past.

Classification by time of response

1. **Reactive/Delayed response:** Responses are delayed until the intrusion is confirmed by an existing attack signature, confidence metrics or other assurances. Most intrusion response system systems use this approach although it is not ideal for safety-critical systems. One popular solution is to suspend suspicious processes [SF00] [BMRL03]. This suspension can act as an intermediate step before a response strategy is determined.
2. **Proactive approach:** The proactive approach attempts to prevent an attack from happening thereby defending the system. Predicting attacks from available information is desirable but suffer from several limitations including potentially large false positive rates. Complex prediction schemes have been proposed to report multi-step attacks. Although this falls under intrusion prevention rather than response it is worth comparing. This approach as mentioned earlier suffers from the limitations faced by prevention systems. ADEPTS [FWM⁺05] uses attack graphs to model attacker's goal

and estimates possible spread of an intrusion. Then, an appropriate response is executed on nodes in the attack graph. Although this proactive approach of defending other nodes before an attack guarantees improved security, it might restrict usability in case of an overly cautious response.

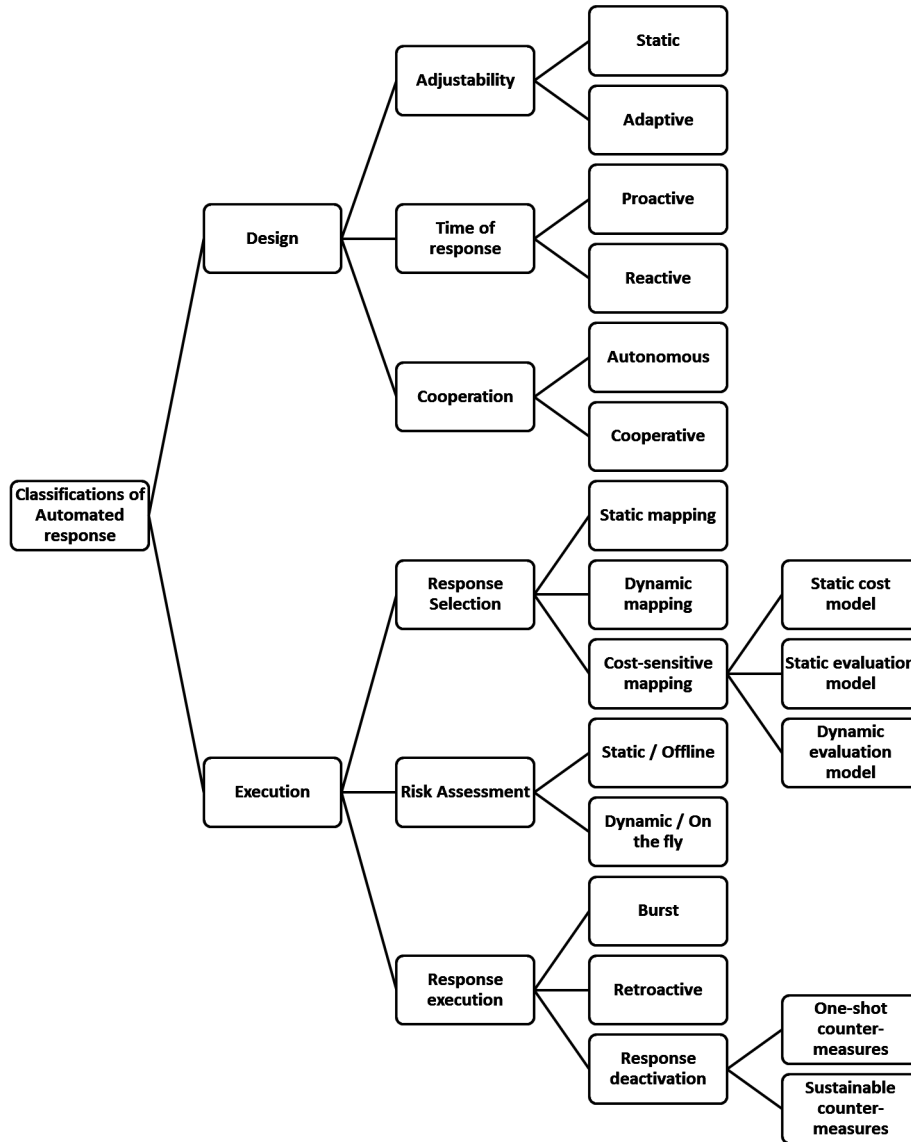


Figure 4.1. Taxonomy of automated response methods

Classification by cooperation

1. **Autonomous response:** Autonomous responses are taken by individual hosts in a network without any decision support from outside. The Hive Mind, our lightweight

Model and Type	Adaptability	Predictability	Cooperation	Response Selection	Response Evaluation
CSM [WFP96]: Host-based	Static	Reactive or Proactive	Yes	Dynamic Mapping	Static cost
EMERALD [PN97]: Network-based	Static	Reactive	Yes	Dynamic Mapping	Static cost
Adaptive Agent-based intrusion response system [RCJHP00]	Adaptive	Reactive	Yes	Dynamic Mapping	Static evaluated cost
SARA [LVHO ⁺ 01]: Host-based	Adaptive	Reactive	Yes	Cost-sensitive	Dynamic cost evaluation
Lee et. al's model [LFM ⁺ 02]: Host-based	Adaptive	Reactive	No	Cost-sensitive	Static evaluated cost
Specification-based intrusion response system [BMRL03]	Adaptive	Reactive	No	Cost-sensitive	Dynamic cost evaluation
ADEPTS [FWM ⁺ 05]: Host-based	Adaptive	Reactive	Yes	Cost-sensitive	Dynamic cost evaluation

Table 4.1. Comparison of different response systems

event monitoring and response system, currently supports local automated response strategies.

2. **Cooperative:** Systems that follow cooperative scheme are capable of responding locally but the strategy for a response is determined outside from a global resolver. This helps in getting the global picture of an intrusion, containing the volume of damage and possible increasing awareness about an intrusion. Several systems have implemented cooperative response strategies. The two main approaches followed are, a distributed response system with participating entities sharing decision-making process [SHS⁺01] [WFP96] and a global coordinator approach where a single orchestrator determines and disseminates the decisions [LVHO⁺01].

Classification by response selection

1. **Static mapping:** In this type of response selection, an alert response mapping is created beforehand. Although simple to build, this approach suffers from vulnerabilities

due to its predictability. If an intruder learns the response strategy it would be easy to work around it since the system doesn't change. Also, it is not state-aware and is not very scalable.

2. **Dynamic mapping:** The response is based on different factors like attack metrics including confidence, severity, frequency as well as system state, network state and internal policies. Response for the same attack might differ depending on the factors mention above. One drawback of the model is that is not sensitive to the cost of executing a response. Several systems use dynamic mapping techniques for response selection including CSM [WFP96] and EMERALD [PN97].
3. **Cost-sensitive mapping:** This method attempts to compare intrusion damage to cost of executing a response. There are risks in executing certain response like shutting any user down or removing a resource. Several methods have been proposed to evaluate the cost of a response but it becomes even more challenging in cases where interdependence between system is increasing. Some of the popular cost-sensitive mapping systems are Lee et. al's model [LFM⁺02], Foo et. al's ADEPTS [FWM⁺05] and Stakhanova et. al's cost-sensitive model [SBW07a]. The following categories help in understand the evaluation mechanism a model uses.
 - **Static cost model:** Response cost is considered static. This is not practical since the cost of shutting down a resource is not the same as the cost of adding additional logging capabilities.
 - **Static evaluation model:** In this approach, the cost is calculated based on the positive effects and negative impact ratio. The positive effects can be calculated based on performance metrics or their consequences to the CIA triad. Negative impacts can be calculated based on the availability and performance metrics.
 - **Dynamically evaluated cost model:** The cost is based on the state of the system. The cost of taking the same action might differ between states in this model. Typically, these models include real-time risk assessment.

Classification by risk assessment

1. **Static assessment:** This offline assessment uses static values assigned to resources in the network/system and uses several guidelines to assess the risk involved. This method cannot include environment states or dynamic values but is useful to decide on the initial assessment before an attack is seen in the system.
2. **Dynamic assessment:** This is an online real-time assessment that provides risk for individual hosts or the entire network. This is useful in improving the performance of the response taken. This model can evaluate from a set of responses in terms of which one will be suitable and least expensive for the current state. The evaluation is made possible by service/resource dependencies or attack graphs. Many of the recent systems that include cost-sensitive responses also include risk assessment implicitly.

Other classifications

Classification by response execution method

1. **Burst:** No risk assessment once response is triggered
2. **Retroactive:** This method uses a feedback mechanism where recent response history is used in fine tuning future responses.

Classification by response deactivation method

1. **One-shot countermeasures:** Defends against an instance of an intrusion
2. **Sustainable countermeasures:** Defends against all future instances of an intrusion and identifies attack paths to understand the entry of an attack and other key points on how to the attack penetrated and so on. This feature has rarely been implemented although it has been suggested as a useful tool.

4.4 Desired Features in Lightweight Distributed Response Systems

We have taken both the performance and effectiveness into consideration to propose a scheme for automatic response with significant coordination. This result is a lightweight distributed intrusion detection system, the Hive Mind. Although it is ideal for Lightweight intrusion

response system to have the most appealing feature from all the classifications of intrusion response systems, it is challenging to achieve some of them as described earlier.

- **Adaptability:** Lightweight intrusion response systems like the Hive Mind have mobile agents (“Ants”) carrying responses for a certain sensor. While it is easy to have static responses, it is much more useful for the system to have adaptive responses. Fortunately the ants wandering around the system can collect evidence of an intrusion seen before and can modify the response with parameters to enforce the required level of “harshness”.
- **Predictability:** It is often desirable to have proactive mechanisms to block attacks before those attacks cause damage. This is often infeasible in lightweight systems where continuous monitoring is not possible. Therefore the Hive Mind uses a reactive approach while attempting to contain the damage as early and as effectively as possible.
- **Cooperation:** The strategy for each response comes from the master node for the Hive Mind and the node manager executes it, although the response execution is not a feedback mechanisms like in other coordinated global response mechanisms. The Hive Mind does not fall into one single category of this taxonomy.
- **Response selection:** The Hive Mind has the potential to use dynamic mapping for response with parameters carrying severity level. To embed cost-sensitive response selection there is a need for risk assessment.
- **Response evaluation:** Evaluating the cost of a response cannot usually be done in real-time. This is at least partially due to computing power and also because local assessment is easy to implement, but dependencies are hard to resolve immediately.

Chapter 5

The Hive Mind: Lightweight Distributed Event Monitor

Security Event Monitoring has two contrasting approaches. One approach is to monitor all that is possible at all times and places. This approach aims to ensure accountability and availability of information to conclude what went wrong. But this approach also suffers from limitations due to operational and performance requirements such as disk space, processing overhead and network bandwidth. Another approach is to employ monitoring in a more judicious fashion. The ideal monitor that follows this approach will capture exactly the details needed to deduce an adverse action using the fewest resources (time and space) possible. Although determining exactly the optimal degree of monitoring to detect adverse events with minimal resources is difficult, systems running in restricted environment with limited computation power should lean towards this approach as any disruption to their performance is often unacceptable. The Hive Mind [TP13] [Tem13] , developed at the University of California, Davis Computer Security Lab takes the latter approach.

Biological systems naturally optimize their activities [BDT99]. A typical example is foraging process by animals and insects. Another striking feature is the autonomous and decentralized behavior of biological creatures like termites building their mounds. This motivates designing artificial systems capturing the qualities of biological systems. One such characteristic is the biological ant's behavior to direct itself and others back to the nest after foraging using pheromone (a temporary chemical signal that ants can sense) trails. This characteristic provides an effective resource direction property useful for any decentralized

lightweight event monitoring system. The Hive Mind system is inspired from the ant's foraging model.

5.1 Inspiration from Biological Ant Foraging

Foraging ants leave the nest dropping chemical markers called pheromone that can be sensed by other ants [BDT99]. When an ant finds food, it follows the trail of pheromone back to the nest. Other ants align with the trail once they encounter it, thereby increasing the intensity of pheromone. The pheromones evaporate and naturally the recently travelled trail has better “visibility” than the older ones. This property of direction is fundamental to the Hive Mind design. Another property is that the ant foraging is decentralized.

The ant foraging behavior can be considered analogous to the lightweight event monitoring albeit, without considering the central nest. In lightweight event monitoring the property of directing resources can be very useful ”in that the system can be designed in a way that seeks to optimize the detection mechanism to run the any detection code at exactly the “right” time. Although the ant foraging system is analogous to event monitoring, it is important to understand that the environments in which the systems work are completely different. Thus the design decisions deviate from biological model to accomodate the need of the virtual environment. To understand the justification behind these deviations from the biological model, it is necessary to understand the need and constraints of the system we intend to build.

5.2 Characteristics of the Hive Mind

In Chapter 3, we discussed desired features of lightweight monitors including decentralized operation, flexibility and efficient use of resources. The Hive Mind offers all the desired features mentioned above. Apart from those, the Hive Mind also exhibits the following characteristics as the result of its lightweight design.

- **Mobile Sensors:** Mobile sensors move between entities in the Hive Mind system to be executed. These sensors detect conditions they are programmed to look for and also respond locally based on the result of detection. The sensors are lightweight dictated by the need to conserve resources. The granularity of the sensor code is often decided

by how much an entity can handle without affecting the system's usual activities.

- **Non-determinism:** The time to detect a programmed activity is non-deterministic due to mobility of sensors between entities. Although estimates can be made on how frequent the detection code runs, it is not deterministic.
- **Delay in detection:** Since detection code is not run all the time in contrast to heavyweight monitoring systems it is possible to have delayed detection of events. This is the price the system pays for reduction in use of resources.
- **Resource coordination:** Apart from the benefits of being lightweight and fast, the Hive Mind has a more unique advantage. The Hive Mind offers a method for efficient resource coordination without central control. The resource can differ based on what environment the system is deployed on. In a network of hosts with restricted computation power, different detection sensors can be directed to a host looking for problems. These sensors could look for other potential threats.

5.3 Theory

The Hive Mind uses “cybernetic ” Ants [TP13] to monitor events across a group of independent entities often performing a designated function. Some examples for entities can be hosts in a network, devices in a control systems or mobile phones. The following components are integral parts of the Hive Mind. The terminology is adapted from Templeton's works [TP13, Tem13] for clarity and consistency.

- **Hive:** The set of systems that are monitored by the Hive Mind system. The entities can be diverse in hardware, operating systems and type of use among other things. The systems coordinate indirectly through mobile agents but do not communicate with each other through dedicated channels. In other words, systems that are being monitoring are oblivious of other systems except for their neighborhood.
- **Node Manager:** The process that runs on each entity that decides the actions to be taken on that node based on the messages it receives. Node Managers are the communication points for mobile agents to execute tasks.

- **Ants:** Ants are fundamentally messages passed between nodes. An Ant can be imagined as an agent that “carries” a designated function that is executed by the Node Manager it is received by. Going forward, we refer to biological ants as “ants” and their virtual counterparts as “Ants”.
- **Queen:** The Queen is the designated node where administrative tasks run. It is the point of collection of individual Ant’s telemetry [TP13]. The Hive Mind is decentralized and the Queen node can be restricted only for initiating and evaluating experiments.
- **Task Functions:** The functions that look for a programmed activity that may be or part of the evidence to an intrusion that occurred. They usually refer to a baseline configuration to detect changes to the system. One example of a task function might be to monitor for unexpected user accounts or processes.

Ants move through the hive for a specific number of hops where the number of hops is tunable either a priori or dynamically. Each Ant executes the task function they carry on the node they arrive on and then move to the next node based on the result of the task performed. Ants wander around using a “direction-biased random walk” strategy [TP13]. This means that an Ant tends to stick to a particular direction but can drift aside. The level of wandering is controlled by two parameters, namely, *drift* and *wander*. Drift indicates the probability of an Ant moving sideways, but heading in the same direction. Wander indicates the probability of an Ant shifting its direction. These parameters impact detection time as they determine the movement of Ants. Ants drop a marker called *pheromone* on each node. These pheromones are often bit flags that indicate that an issue was found in that node. Ants shift their direction, after a successful detection, in an attempt to visit undiscovered areas of the hive. A typical segment of hive and the types of ant movements are shown in Figure 5.1.

Pheromone impact

Pheromones create an emergent, albeit indirect and decentralized coordination amongst Ants. Once an Ant finds a problem, it drops pheromone markers on nodes it visits after. Other Ants follow the pheromone marker trail. This ensures that the neighborhood where a problem was found gets more attention. The pheromone marker “evaporates” over time

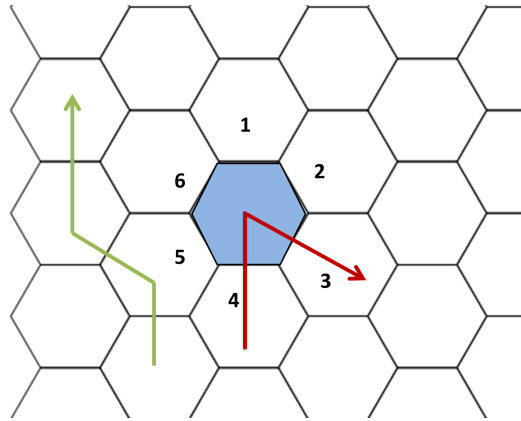


Figure 5.1. A typical segment of a hive showing a node (blue) and its neighbors. It also shows the two types of ant movement *Drift* (green) and *Wander* (red).

making sure that the same set of nodes do not get an overload of Ants. This redirection of Ants towards an effected node is highly beneficial for problems that have high likelihood of existence in the neighborhood. A graphical representation of detection, dropping mode and other ants following the trail is shown in Figure 5.2.

Lifetime of Ant

The Hive Mind system provides a parameter to control the number of hops an Ant can make before getting destroyed by the node manager on the node it visits last. During its lifetime an Ant can either perform tasks or drop markers. An Ant can switch between these two tasks as well. The length of marker trail an Ant drops is also tunable. An ant performing a task typically follows these steps.

1. Collect information regarding a task
2. Check for evidences of its assigned problem
3. Execute automatic response if problem is found
4. Switch to dropping mode
5. Leave the node

Ants that are in dropping mode drop the marker and leave the node. Once an Ant drops the assigned number of pheromones, it switches to detection mode and continues to

hunt for problems. These Ants can also be tuned to look for different problems at the end of their dropping mode.

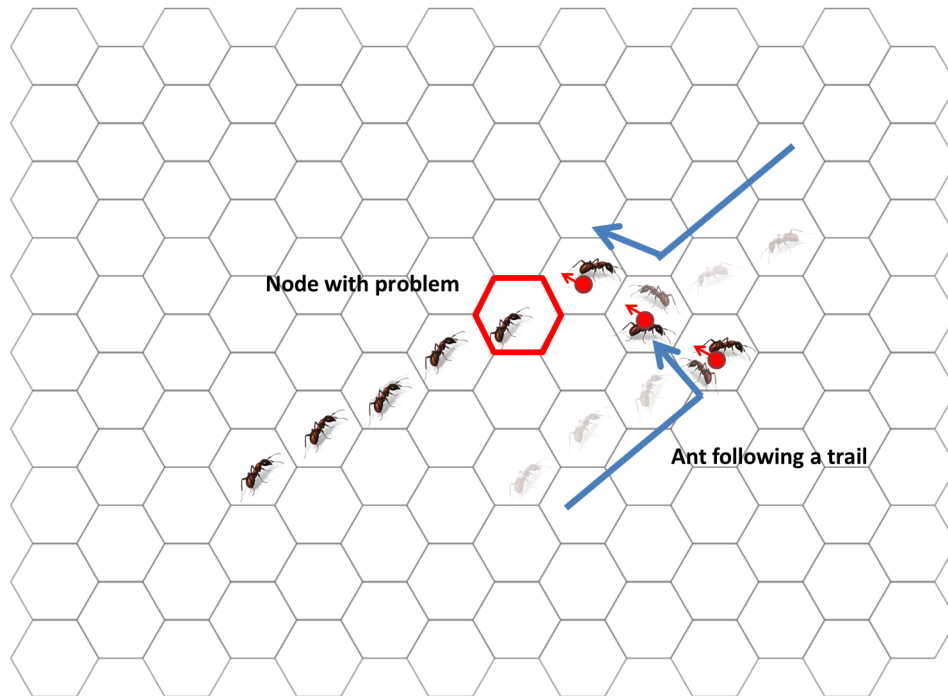


Figure 5.2. A typical Ant detecting a problem and dropping pheromone (red circles). Other ants (shown in gray) follow the pheromone trail (red arrows) back to its origin once they sense pheromone.

5.3.1 Automated Response

Simple response

Each task function an Ant performs has an automatic response function associated with it. This response function typically contains code to mitigate or eradicate the problem the task was created to identify. For example, let us assume a user is discovered having superuser permission contrary to a predefined specification. If an Ant carrying the task to identify this problem reaches that node, it detects the problem and executes the response, which may be to strip the permissions from the user or even remove the user account.

Decision tables

The response can also be made somewhat flexible with the help of decision tables inside the task. The response can be chosen based on the number of evidence collected. It can be designed in such a way that increase in number of evidence, increases the harshness of a

response. This can be implemented in the current Hive Mind version without any change to the source code. Additional functionality can be added to include “weight” to each type of evidence and have the decision taken based on accumulated weight.

Dynamic response

Dynamic response involves the node initiating a response to varying degrees depending on problem detection and the environment. Dynamic response gives the decision-making control to the local node based on the information gathered by the Ant and task functions. Although this can add overhead to the hosts, it only requires nodes to act when the task function finds a problem. The node can create a suitable response strategy and execute it. This also does not require the Ant to stay in the same node thereby avoiding stagnation. This type of response is not present yet in the current Hive Mind prototype, however.

Evaluation of response

The current Hive Mind version does not include response evaluation. That is, an Ant executes a response and then leaves the node. However, it might be useful to store the response history so that other Ants drawn towards the node can detect if the problem has reappeared and can execute an alternative refined response. This will be useful especially for dynamic response strategies. Evaluation of response becomes harder when system activities are interconnected. For example, if a web server process was killed in an affected node, it can affect other users connected to the server. This increases the risk of affecting usability of the system and in the worst case, “destabilizing” the entire system. Unfortunately, such a global response evaluation defeats the purpose of a fast lightweight system. Response and dependency evaluation might take significant amount of time and can stagnate an Ant.

Chapter 6

Experiments and Results

In this chapter, we present several experiments and their results. Along with those, we also present a test scenario to discuss the potential practical utility of the Hive Mind. Although the scenario is simplified for clarity, it is a meaningful, non-trivial case to discuss the types of scenarios the Hive Mind is capable of addressing.

6.1 Prerequisites

To conduct experiments using the Hive Mind, it is necessary to have a distributed system that satisfies the following prerequisites. A network of nodes (preferably 100s of nodes) where every node is connected to a central administrator called Queen. The Queen is required to initiate the monitoring setup and collect statistics on each nodes activities.

For our testing, we use Deterlab [Ben11], a virtual platform for cyber-security testing used by a variety of academic researchers. As with many testbeds, physical resources are limited so researchers are encouraged to use virtual hosts to scale the experiments up. We argue that using virtual node is more appropriate for testing the Hive Mind prototype since they reflect a lightweight system closely when configured properly. Deterlab offers *containers* to install several virtual hosts (around 100) per physical node.

Deterlab containers

Deterlab containers are physical hosts that can hold many virtual nodes. The container brings up the configured number of virtual hosts through an installation script. The density of virtual nodes per container is termed the *packaging factor*. The containers appear as sin-

gular units in the Deterlab interface but individual virtual nodes can be accessed like regular nodes. After a required number of nodes are connected to the Queen, the installation of Node Manager is required on every node in the hive. Since a queen can communicate with all the nodes, we install the required software on individual nodes without logging into them manually. At this stage we also need to setup an “rsyslog” log server in Queen and individual nodes, to connect and send information for statistical analysis offline.

Now that the “hive” is ready, we can inject one or more Ants that carry messages (usually codes for tasks to be carried out). The Ants can be “teleported” anywhere from the Queen to start after which they conform to the the Hive Mind design of moving autonomously. Facilities to inject “target problems” are also available. Ant group behavior and cooperation can also be tested.

Size of the system

We have two hives, one with 256 nodes and the other with 1024 nodes. The 256-node hive uses 6 containers (physical nodes) and the 1024-node hive uses 21. Each container holds, approximately, 50 virtual nodes. The container machines have the following features: [PC2]

- One Intel(R) Xeon(R) CPU X3210 quad core processor running at 2.13 Ghz
- 4GB of RAM
- One 250Gb SATA Disk Drive
- One Dual port PCI-X Intel Gigabit Ethernet card

Each virtual machine has a linux-based Ubuntu operating system installed on them with a shared NFS (Network File System) volume mounted on each node, to copy data in and out of the node through the Deterlab user account.

6.2 Experiments on the Hive Mind

This section includes experiments to understand the behavior of Ants in the Hive Mind. In particular, we perform tests to determine the optimal number of Ants of a given type to cover the entire hive, focusing on number of steps and time taken. We also perform tests to

understand whether the pheromone trail length has significant impact in the total detection time of preset problems.

6.2.1 Coverage

Here, the goal is to test the time taken as well as the total number of moves it takes to visit every single node in the Hive atleast once. We inject Ants at random locations in the hive and measure how long it takes for them to travel through the hive covering all nodes. These tests also help understand the average number of revisits Ants make to a node before visiting every node. We increase the number of Ants to measure the change in coverage time and total number of steps. Coverage might help us infer the amount of wasted resources of resources with the introduction or reduction of the number of Ants of the same type in the system. The number of Ants is an important factor since it is directly correlated with the resource use. Several experiments were run to demonstrate coverage and the most significant tests and results are shared below.

6.2.1.1 Theory

To compute the average number of steps it takes to cover the entire hive, we need to compute the summation of the mean trials to visit a new node. It is represented as,

$$\sum_{n=1}^H H/n, \text{ where } H \text{ is the hive size}$$

The average number of steps for 256-node hive is 1568 steps and for 1024-node hive is 7689 steps. We now describe an experiment to demonstrate this theory on a real system.

6.2.1.2 Experiment with One Ant

Here, a single Ant is injected at random and is tracked to see when it covers the entire hive. The hops taken by an Ant to explore newer nodes increases linearly for a short while and then increasingly it gets harder to spot new nodes. The plot for number of steps against the number of nodes found is shown in Figure 6.1. It is apparent that the number of steps taken tends to match with the theoretical average. The number of moves does not appear to change with the increase in the number of Ants. This is because the total of number of moves made by all the Ants together is a function of the hive size. It is analogous to the problem of picking H unique items from a group of H items.

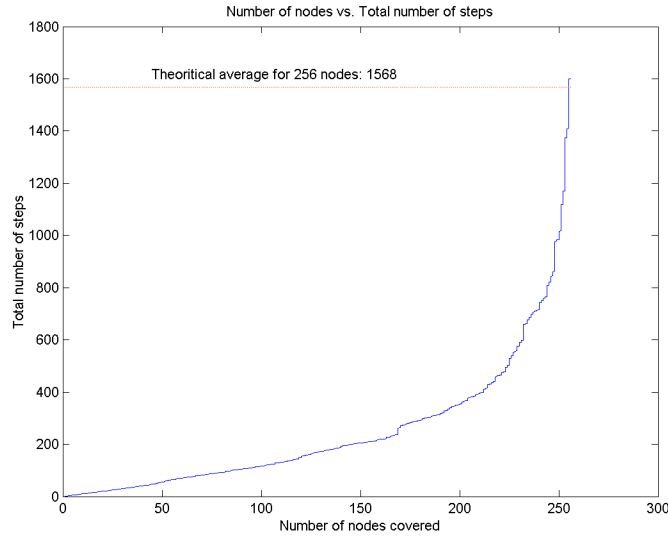


Figure 6.1. Number of nodes uncovered vs. number of steps taken for 1 ant in a 256-node hive

Also, the time taken to cover all the nodes has a similar graph with the last few nodes taking up a lot of time to be covered as shown in Figure 6.2.

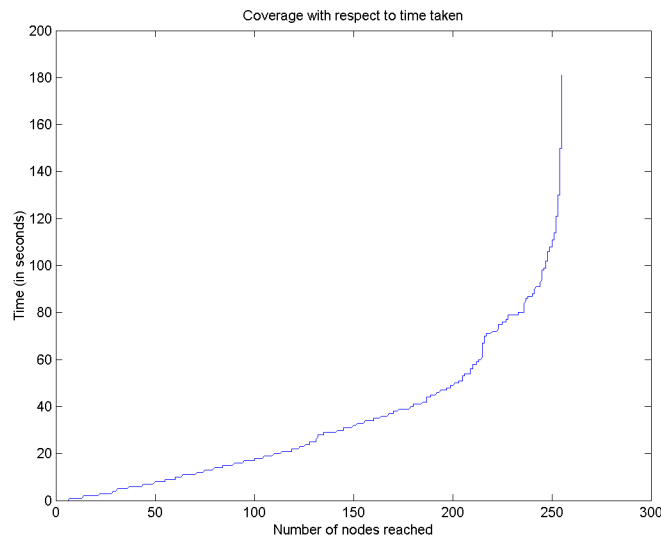


Figure 6.2. Number of nodes uncovered vs. Time taken for 1 ant in a 256-node hive

6.2.1.3 Experiment with multiple Ants

The purpose of this experiment is to test the rate of change in coverage with respect to the increase in the number of Ants injected. It is seen that the graph is similar to the one Ant experiment's graph but the number of visits is much higher. This is due to repetitive coverage

of same node by multiple Ants. However, the most interesting result is the reduction in time the inclusion of multiple Ants causes. A graph showing comparisons between several runs of experiments with 1 Ant and 5 Ants are shown in Figures 6.3 and 6.4

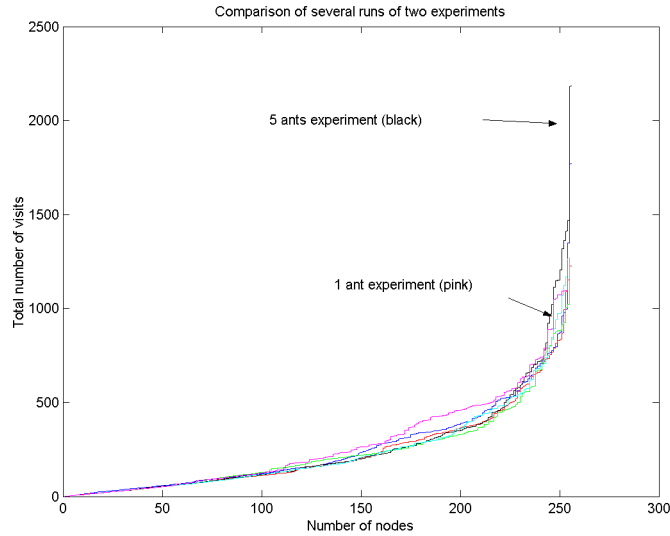


Figure 6.3. Number of nodes uncovered vs. Number of steps taken for a 256-node hive

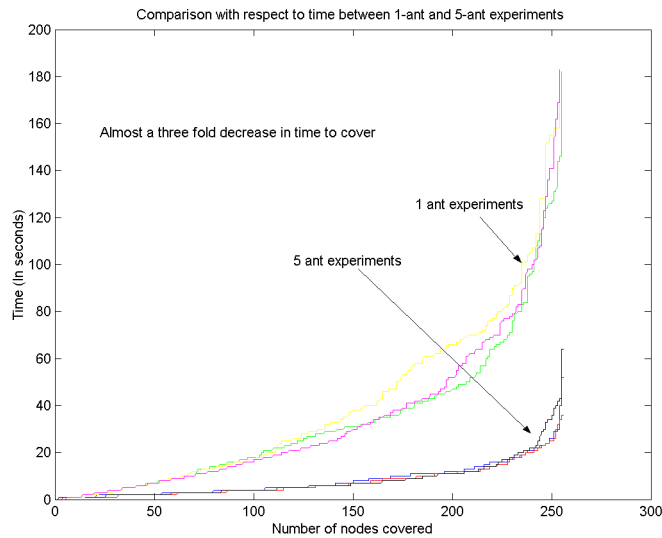


Figure 6.4. Number of nodes uncovered vs. Time taken for a 256-node hive

6.2.1.4 Increase in Ant count

We tried to measure the change in coverage steps with increasing number of Ants injected into the system. The results showed that the number of Ants did not affect the change in coverage steps significantly. There is no consistent decrease in coverage steps as one might

expect. In fact, the number of steps to cover the hive tended to stay near the theoretical average. This can be seen in Figure 6.5.

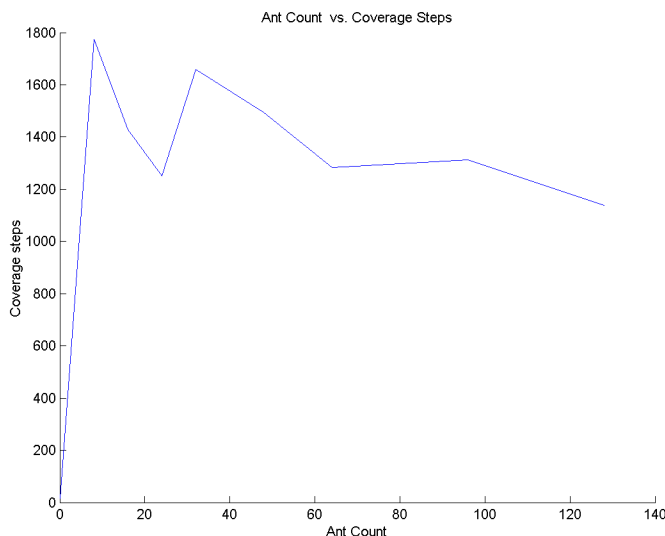


Figure 6.5. Number of Ants vs. Coverage steps in a 256-node hive

6.2.2 Detection time

Detection time is another parameter to measure the Hive Mind’s performance. Total detection time is the time required to find all the problems that exist in the system. We measure it by installing known problems on the nodes and injecting Ants into the hive. The Ants detect and respond to a problem and create a pheromone trail for a fixed length before restarting the detection phase.

Experiment

We installed 10 identical target issues at random in the 256-node hive. We increased the number of Ants gradually from 16 ants to 128 ants and tracked the difference in detection time. Although the decrease in time taken to detect all the problems was not as smooth as we expected, as seen in Figure 6.6. This figure also shows a comparison between the coverage time and detection time. It is obvious that detection time is always lesser than or equal to the coverage time. It is, however, noteworthy that in some cases the detection time approaches the coverage time. This fluctuation might be due to the fact that target issues were set randomly. It is possible that, in some experiments, Ants might have been injected

closer to the problem nodes compared to others. It is clear that there is a decreasing trend in time taken apparent from the plot.

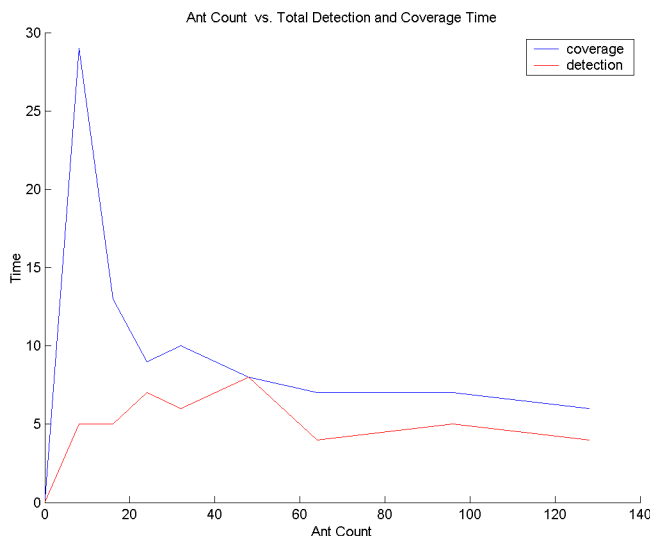


Figure 6.6. Number of Ants vs. Time taken for detection and coverage in a 256-node hive with 10 pre-installed target issues

6.2.3 Pheromone trail length

The main aim in the following set of experiments is to determine an optimal trail length for which Ants should drop pheromone markers once a problem is found. This can be tested by configuring the trail length parameter in the Hive Mind configuration file. We also expect significant changes to detection time when the pheromone trail length is varied. To make sure the markers are dropped we installed target issues at random on 10 different nodes.

Zero trail length

In this case, Ants start looking for problem again, right after they found one. This makes sure other Ants are “not influenced” and drawn towards a single neighborhood. Although this case does not help Ants coordinate, it is useful for testing the impact of coordination and lack thereof.

Experiment

We perform two different experiments with the trail length of pheromone markers set to 0 and then to an arbitrary fixed value of 32. We analyzed the change in detection time over multiple runs and monitored for significant differences. We found that the detection times

Table 6.1. Detection results for varying pheromone trail length

Trail length	Detection time (in seconds)
0	6
32	5

are highly random due to the random nature of existence of problems in the hive. So, we tried to configure known problems in a small neighborhood in the hive, in an attempt to increase the impact of coordinated resource direction. The results were encouraging, with a decrease in detection time when the pheromone trail length was changed to 32 from 0.

6.3 Scenario-Based Study

Testing the correctness and efficiency of the Hive Mind framework requires useful scenarios in order to provide meaningful results. Ideally, The scenarios ought to have multiple sensors looking for evidences of a bigger problem. The following properties, if possessed by the scenarios, bring out best from the model.

1. High probability of same problem being found in the neighborhood of any affected host
2. High probability that an affected host has several different problems

Since the Ants draw attention to a node where a problem has been found, the purpose of Ants carrying new sensors is to find other problems, too. Also, on their way towards a particular affected host, Ants that follow a pheromone trail check for problems in the neighborhood. Although this is usually ideal, there are exceptions. For example, some attacks can target specific systems for which the neighborhood property does not hold. Even then, there is a good chance that there are several issues with the same host (the attacker might have performed several attacks after taking partial or complete control of the system). Thus, the design by which one Ant entices other Ants to the neighborhood can be argued as being not wasteful.

6.3.1 Data Exfiltration Scenario

In a system where continuous monitoring is expensive and/or infeasible, we need to resort to figuring out evidence of the problem. Some of the attacks might not be identified by

single piece of evidence but resolving a collection of evidence. Also, if the entire logic resides in a single sensor, it might become too heavy and expensive. This justifies the need for distributed evidence/problem identification.

To use a concrete scenario we looked at the problem of data exfiltration in an organization. Let us take the case of a user account downloading files from an FTP server to his hard drive and then uploading all of them into his personal cloud. Now he erases the data after uploading. Some sensors that would help identifying this problem would a sensor that computes the difference in free space during certain intervals. This might work when executed between the time periods when data is downloaded and data is erased. So we cannot depend on only this method. We can have sensors that calculate the amount of bytes transferred out/in which might have been logged. This again cannot be a standalone identifier but gives a good indication. The intruder or malicious insider could have deleted the logs, but if there is a way to keep track of checksums for log files so no deletion allowed we can conclude that there was an exfiltration attempt with a degree of confidence.

This particular scenario matches the second property mentioned above where the same host has several problems. But, if this is a malicious insider it is hard to argue that it will be found in the entire neighborhood. There is also a chance that this might be the work of an intruder who has used a particular software vulnerability to gain access and steal critical data. If so, there is a good chance that the software is used by many users across a division of the organization. This is a strong case to satisfy the first property.

6.3.2 Sensor tasks

6.3.2.1 Free disk sensor

This sensor task retrieves the number of free blocks in the file system and ensures that the value is below a specified threshold. The threshold can be passed as a parameter to the Ant carrying the sensor function. It can also be stored in the node as a baseline. If a node's file system has free memory below the threshold, the task function sends an e-mail to the administrator as the default response.

6.3.2.2 File existence check

From our study and discussions of the selection of Task functions for the exfiltration scenario, we arrived at a useful by-product. One of the task functions required a lightweight method to search for a file's existence. We derived an extension of Bloom filters [Blo70] that we term as Hierarchical Bloom Filters. We believe this is a significant result and have dedicated the entire section 6.4 describing its use and benefits.

6.3.2.3 Directory change detection

For data exfiltration detection, it is useful to know if any changes have been made to directories inside the file system. We can use hierarchical data structure to hold information about changes inside directories. To pinpoint the changes, we require a more complicated data structure and a significant amount of space. So, we choose to perform change detection test, on demand, whenever the simpler test returns positive.

6.3.2.4 File change detection

To carry out file change detection, we store the MD5 signatures of the files and compare them with the baseline to detect changes. Since there can be many files, and therefore, many signatures to store, we resort to this sensor only for configuration files and other important files that are specified not to change unless authorized to.

These sensors correspond to various task functions the Ant can carry in the Hive Mind framework. The sensor list mentioned above is not comprehensive and custom sensors can always be added.

6.3.3 Response

Due to lack of time available for this thesis, we could not test the impact of responses on the Hive Mind's performance. In future, we would like to run thorough experiments on automated responses and their impact on the coverage, detection time and the stability of the system. We would also like to run tests on responses with online response evaluation and risk assessment.

Experiment

Although we do not present complete analysis of automated responses, we ran a comparison test to ensure the benefit of automated response. We ran a target issue detection experiment

Table 6.2. Comparison between Manual and Automated response

Response type	Average Detection time (in seconds)
Manual response	7
Automated response	5

with 10 issues on the 256-node hive. We ran two different types of test, one with no automated response and the other with automated response activated. The change in detection time is calculated and tabulated below. Manual responses only include the time to discover the problems. It does not include the administrator’s reaction time. It is apparent from the table that, automatic response improves detection time. The reason for this improvement is the fact that, with no response, the issue on the node is pending to be resolved by an administrator. This causes wandering Ants to continue to be drawn into the nodes.

6.4 Hierarchical Bloom Filters

To ensure the existence or absence of a file, we can use a recursive directory search algorithm to percolate down from root and hit the file entry. However this approach is slow. We can think of an approach to maintain a file entry database with indexing for faster lookup but that has a space overhead which might force us to access the disk more often than we would wish to.

To address this problem we can use a Bloom filter [Blo70]. A Bloom filter is a simple data structure which intends to reduce the amount of space required to contain information by allowing a very small fraction of errors. More specifically, a Bloom filter is a bitmap which contains hash-coded (the technique is independent of the hashing algorithm used) information.

6.4.1 Bloom filters

Initially all the bits in the bitmap are set to 0. The data to be contained is subject to a function (F) determining bit locations in the bitmap that are set to 1. For example, a string denoting a file entry can be subjected to any function that returns b bit locations. The bit locations returned are set to 1 in the bitmap. Subsequent entries use the updated bitmap to set their bits. It is obvious at this point that there can be collisions from this approach.

Table 6.3. Comparison between conventional hashing and Bloom Filters

Feature	Conventional Hashing	Bloom Filter
Error	None	Yes due to collisions
Space	N times sizeof(hash value)	Size of the bitmap (Independent of N)
Time	O(1)	O(1)

The collision rate is reduced if the b is large.

To query if a piece of information is contained in the Bloom filter we apply the function F and check the b bit locations it returns. If all b locations have their bits set to 1 then the information might exist or it is a collision. But if any of the locations is not set we can be certain that the information is not present. This technique is very helpful if most queries are not elements of the set that the Bloom filter contains. The total error is computed as a formula of three variables as mentioned below.

$$E = (A - N)/(T - N)$$

The variables here are,

- N = Number of distinct elements contained in the filter
- T = Total number of distinct elements that can be present in the given bitmap
- A = Number of accepted queries (positives)

6.4.2 Comparison to conventional methods

The size of the bitmap we use is usually dependent on the hashing algorithm we use for the filter. The larger the size of the bitmap is, higher is the number of elements it can hold with fewer collisions. Since a Bloom filter introduces some false positives the time to confirm that an element actually exists might be higher and may require $O(N)$ depending on the process of lookup.

This analysis drives us to create a Bloom filter that can fit into the main memory and prevent significant amounts of disk accesses. We can include the file names into the filter and check for the existence of a file or lack thereof. From a security or asset management perspective there can be certain certain files that should not be present in the memory. This

method allows us to quickly (given creation of the Bloom filter is a one-time cost) lookup for a certain element. If the filter returns positive we can further look into using other methods. In a normal environment we do not expect every system to be malicious and therefore most queries looking for “unallowed” file should return negative thereby exploiting the advantages of Bloom filters.

Although the above described method is faster than conventional method, there is scope for improvement in this particular problem. The cost of looking for a file that the Bloom filter returned as positive is very high. Instead of making this steep jump, we propose a hierarchy of Bloom filters for each subdirectory in a directory and merge the individual filters into a bigger Bloom filter. It is immediately clear that each directory has its own Bloom filter containing the file names inside the directory and also the Bloom filters of its subdirectories recursively.

Querying is done on several filters to narrow down the location of the filter. First the topmost directory’s Bloom filter is queried. If the file does not exist then we return negative. If it exists we check the filters of the subdirectories to see which one returns positive and then we dwell down into the tree of filters. Bloom filter lookups are faster than disk access and therefore this method is expected to consume fewer resources.

Two challenges arise here.

1. To merge two Bloom filters they individual filters should be the same size as that of the larger one. The two filters are simply merged with logical “OR”.
2. To maintain the Bloom filter tree we need a lookup table or a dictionary with the directory path as key and the filter as the value.

It is important to analyse the overhead these two challenges bring in. Considering that some file systems might have several thousand directories we need to figure out optimizations that help avoid disk accesses. Also ideally we would want the filters to be smaller in size at lower levels and bigger at the top level. Since implementation does not allow this we must carefully choose the size of the individual filters which is a balance between space and error rate.

It is important to note that even though there is only one lookup operation for Bloom

Table 6.4. Comparison between conventional Traditional and Hierarchical Bloom Filters

Feature	Traditional Bloom Filter	Hierarchical Bloom Filter
Error	Yes	Yes but possibly lower
Space	Size of the bitmap	Size of the bitmap * directory count
Time	$O(1)$	$O(\log D)$ to $O(D)$

filters, in our hierarchical extension we need need more time to check for the file when traversing the tree of Bloom filters.

6.4.3 Potential Optimizations for hierarchical filters

1. A variety of potential optimizations exist. One is to break down the root filter to R initial filters thereby giving opportunity to reduce the size of the individual filter from M to potentially M/R . The reduction in size might be substantial in the case of several thousand directories. However, the downside is the increase in the price for lookup. Lookup time increases from $O(1)$ to $O(R)$ for every query.
2. Another optimization involves limiting the construction of Bloom filters to few top levels only. After narrowing down to a certain limit we can start to look into the directory itself. This reduces the uncertainty and thereby the error rate in general. The driving factor for the optimization is that lower level directories might have very few files under them for which constructing several Bloom filters might be unnecessary.

Choosing the right hash function for the Bloom filter has been the biggest problem so far. We have tried,

1. CRC-32
2. MD5
3. SHA-224
4. SHA-256

CRC32 computation is 2-4 faster and takes up a quarter of the space MD5 requires. CRC is light but suffers from more collisions. Other hash functions take comparable time to compute but SHA offers fewer collisions.

Chapter 7

Conclusion and Future work

7.1 Summary

This thesis presents a lightweight host-based security event monitoring and response system that enables indirect coordination among resources. The system we have presented, the Hive Mind, is inspired by biological ants' foraging model. The system satisfies most of the important features of lightweight security event monitors. The Hive Mind is flexible and distributed, yet coordinated and resource-conscious. The Hive Mind also offers resource redirection, a unique advantage to its merit. From experimentation, we gathered that the system offers important advantages on problems that tend to exist in a localized neighborhood and ones that exist together. Such problems are common and non-trivial in the world of security.

The Hive Mind model also offers a useful platform for automated response research and implementation. Since nodes are distributed, manual response might not be scalable and automated response simplifies work greatly. The Hive Mind model inherently offers local automated response. The response can further be extended to decision tables that offer more flexibility. The performance of the Hive Mind in general has been consistent with the theory it was built on. This was demonstrated successfully through experiments. To conclude, the Hive Mind has enormous potential and, with advanced automated response strategies, it can serve as an effective security event monitor for lightweight systems that mark the future where it currently seems computer security is headed.

7.2 Future Work

Although, the Hive Mind has transformed into a stable system over the past few years, potential for expansion is significant. Further analysis on the Hive Mind's performance is required, however. In order to find optimal configurations, it is clear that a wide variety of tests are needed under highly varied host and network scenarios. Also, more tests on responses using the existing test beds are needed. More sophisticated response strategies could also likely strengthen the system. Studies on global response and awareness about network-wide attacks are required to improve the efficiency of the system. Finally, clearly, finding applications for the Hive Mind in real world systems to test its actual performance is very important.

REFERENCES

- [Axe00] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Technical report, 2000.
- [BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*, volume 4. Oxford university press New York, 1999.
- [Ben11] Terry Benzel. The science of cyber security experimentation: the deter project. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 137–148. ACM, 2011.
- [Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [BMRL03] Ivan Balepin, Sergei Maltsev, Jeff Rowe, and Karl Levitt. Using specification-based intrusion detection for automated response. In *Recent Advances in Intrusion Detection*, pages 136–154. Springer, 2003.
- [CHSP00] C Carver, JM Hill, John R Surdu, and Udo W Pooch. A methodology for using intelligent agents to provide automated intrusion response. In *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, NY*, pages 110–116, 2000.
- [CID99] The common intrusion detection framework. <http://gost.isi.edu/cidf/demo>, 1999.
- [Coh99] Fred Cohen. Simulating cyber attacks, defences, and consequences. *Computers & Security*, 18(6):479–518, 1999.
- [Den87] Dorothy E Denning. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, (2):222–232, 1987.
- [FBI] What’s the current state of computer network security? http://www.fbi.gov/news/stories/2005/july/ccyber_072505. Accessed: 2013-09-01.
- [FWM⁺05] Bingrui Foo, Y-S Wu, Y-C Mao, Saurabh Bagchi, and Eugene Spafford. Adeptis: adaptive intrusion response using attack graphs in an e-commerce environment. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 508–517. IEEE, 2005.
- [HK88] L Halme and B Kahn. Building a security monitor with adaptive user work profiles. In *Proceedings of the 11th National Computer Security Conference*, pages 17–20, 1988.
- [Hof99] Steven Andrew Hofmeyr. An immunological model of distributed detection and its application to computer security. 1999.

- [HWH⁺03] Guy Helmer, Johnny SK Wong, Vasant Honavar, Les Miller, and Yanxin Wang. Lightweight agents for intrusion detection. *Journal of Systems and Software*, 67(2):109–122, 2003.
- [HWHM98] Guy G Helmer, Johnny SK Wong, Vasant Honavar, and Les Miller. Intelligent agents for intrusion detection. In *Information Technology Conference, 1998. IEEE*, pages 121–124. IEEE, 1998.
- [JLM⁺89] Van Jacobson, Craig Leres, Steven McCanne, et al. Tcpcdump, 1989.
- [KGD08] Ioannis Krontiris, Thanassis Giannetsos, and Tassos Dimitriou. Lidea: a distributed lightweight intrusion detection architecture for sensor networks. In *Proceedings of the 4th international conference on Security and privacy in communication networks*, page 20. ACM, 2008.
- [KV02] Richard A Kemmerer and Giovanni Vigna. Intrusion detection: a brief history and overview. *Computer*, 35(4):27–30, 2002.
- [LFM⁺02] Wenke Lee, Wei Fan, Matthew Miller, Salvatore J Stolfo, and Erez Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1):5–22, 2002.
- [LVHO⁺01] Scott M Lewandowski, Daniel J Van Hook, Gerald C O’Leary, Joshua W Haines, and Lee M Rossey. Sara: Survivable autonomic response architecture. In *DARPA Information Survivability Conference & Exposition II, 2001. DIS-CEX’01. Proceedings*, volume 1, pages 77–88. IEEE, 2001.
- [MHL94] Biswanath Mukherjee, L Todd Heberlein, and Karl N Levitt. Network intrusion detection. *Network, IEEE*, 8(3):26–41, 1994.
- [NCR02] Peng Ning, Yun Cui, and Douglas S Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 245–254. ACM, 2002.
- [OB01] William Osser and Sun BluePrints. Auditing in the solaris 8 operating environment. *Sun BluePrints OnLine, February*, 2001.
- [PB13] Sean Peisert and Matt Bishop. Dynamic, Flexible, and Optimistic Access Control. Technical Report CSE-2013-76, University of California at Davis, March 2013.
- [PBKM07a] Sean Peisert, Matt Bishop, Sidney Karin, and Keith Marzullo. Analysis of Computer Intrusions Using Sequences of Function Calls. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 4(2):137–150, April–June 2007.
- [PBKM07b] Sean Peisert, Matt Bishop, Sidney Karin, and Keith Marzullo. Toward Models for Forensic Analysis. In *Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE)*, pages 3–15, Seattle, WA, April 2007.

- [PC2] Deterlab pc2133 class machines. <https://trac.deterlab.net/wiki/pc2133>. Accessed: 2013-09-01.
- [Pei07] Sean Philip Peisert. *A Model of Forensic Analysis Using Goal-Oriented Logging*. PhD thesis, Department of Computer Science and Engineering, University of California, San Diego, March 2007.
- [PN97] Phillip A Porras and Peter G Neumann. Emerald: Event monitoring enabling response to anomalous live disturbances. In *Proceedings of the 20th national information systems security conference*, pages 353–365, 1997.
- [R⁺99] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.
- [RCJHP00] Daniel J Ragsdale, Curtis A Carver Jr, Jeffrey W Humphries, and Udo W Pooch. Adaptation techniques for intrusion detection and intrusion response systems. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2344–2349. IEEE, 2000.
- [SBW07a] Natalia Stakhanova, Samik Basu, and Johnny Wong. A cost-sensitive model for preemptive intrusion response systems. In *AINA*, volume 7, pages 428–435, 2007.
- [SBW07b] Natalia Stakhanova, Samik Basu, and Johnny Wong. A taxonomy of intrusion response systems. *International Journal of Information and Computer Security*, 1(1):169–184, 2007.
- [SCCC⁺] Stuart Staniford-Chen, Steven Cheung, Richard Crawford, Mark Dilger, Jeremy Frank, James Hoagland, Karl Levitt, Christopher Wee, Raymond Yip, and Dan Zerkle. Grids-a graph based intrusion detection system for large networks.
- [SCTS⁺98] Stuart Staniford-Chen, Brian Tung, Dan Schnackenberg, et al. The common intrusion detection framework (cidf). In *Proceedings of the information survivability workshop*, 1998.
- [SDS00] Dan Schnackenberg, Kelly Djahandari, and Dan Sterne. Infrastructure for intrusion detection and response. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 2, pages 3–11. IEEE, 2000.
- [SF00] Anil Somayaji and Stephanie Forrest. Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symposium*, volume 70, 2000.
- [SFL⁺00] Salvatore J Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip K Chan. Cost-based modeling for fraud and intrusion detection: Results from the jam project. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 2, pages 130–144. IEEE, 2000.

- [SHJ⁺02] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.
- [SHS⁺01] D Schnackengerg, Harley Holliday, Randall Smith, Kelly Djahandari, and Dan Sterne. Cooperative intrusion traceback and response architecture (citra). In *DARPA Information Survivability Conference & Exposition II, 2001. DIS-CEX'01. Proceedings*, volume 1, pages 56–68. IEEE, 2001.
- [SSEJJD12] Alireza Shameli-Sendi, Naser Ezzati-Jivan, Masoume Jabbarifar, and Michel Dagenais. Intrusion response systems: survey and taxonomy. *SIGMOD Rec*, 12:1–14, 2012.
- [Tem13] Steven J Templeton. *Ph.D. Dissertation (in progress)*. PhD thesis, University of California, Davis, 2013.
- [TK02] Thomas Toth and Christopher Kruegel. Evaluating the impact of automated intrusion response mechanisms. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 301–310. IEEE, 2002.
- [TP13] Steven J Templeton and Sean P Peisert. The Hive Mind: Applying a Distributed Security Sensor Network to GENI - GENI Spiral 2 Final Project Report, Unpublished. 2013.
- [WFP96] Gregory B White, Eric A Fisch, and Udo W Pooch. Cooperating security managers: A peer-based intrusion detection system. *Network, IEEE*, 10(1):20–23, 1996.
- [WS04] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

Lightweight Change Detection and Response
Inspired by Biological Systems

Abstract

The state of computer security is complex. With computers taking multiple forms including such lightweight devices as smartphones and virtual machines and then connecting these devices to the open Internet, the task of securing devices becomes harder. To attempt to provide protection from threats it is a common practice to install Security Event Monitors. In this thesis, we present a lightweight host-based security event monitoring and response system called the Hive Mind that is designed to enable coordination among participating nodes for improved detection combined with reduced resource usage. Since lightweight systems potentially have hundreds of hosts connected together manual response to an intrusion is not be scalable. Automatic response becomes necessary to limit the damage as early as possible. We present taxonomy for automatic response and also discuss a model for automatic response in lightweight monitors. The Hive Mind is a host-based security event monitor (SEM), a system that monitors intermittently for potential threats and indirectly communicates the existence of a problem to other nodes using a stigmergic approach inspired from biological systems. Our hypothesis is that the indirect coordination among participating nodes will improve the process of detection without overly reducing the productivity of the nodes. The response model uses local response execution, but aims to achieve global awareness through indirect coordination of nodes. When we apply the system on example scenarios, the results demonstrate that the Hive Mind system offers improved coverage and reduced issue detection time compared to a system with no coordination.