MAIDS for VoIP: A Mobile Agents-based Intrusion Detection System for Voice over Internet Protocol

by

Christian Chita

DEC, Collège de Maisonneuve, 1999
B.Sc., The University of British Columbia, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

May 2008

Abstract

Compared to traditional (PSTN) voice networks, a Voice over Internet Protocol network is a convergence of a signaling network and a data network using Internet Protocol (IP). The use of shared media by VoIP systems opens the door to some uncertainty as to the source of a call. While in the traditional voice networks one has to tap into a specific circuit to eavesdrop, in an IP network any equipment connected to the target LAN can identify, store and playback the VoIP packets that traverse that LAN. Unlike traditional voice networks which have only "dumb" end nodes (i.e. simple telephone receivers), VoIP must, by its very nature, deploy intelligent end point devices such as computers and/or IP phones, which are connected to open public networks. An unprotected, unauthenticated IP network makes VoIP susceptible to hostile use, such as call hijacking, connection tear down, denial of service, or sending computer viruses over the network.

In this thesis, we perform a series of attacks against a commercial VoIP application, and prove that they succeed with nothing more than a couple of identity tokens captured from the network traffic as prerequisites. We then leverage the mobile agent-based framework introduced by APHIDS to design an Intrusion Detection System implementing a gradual attack-response procedure, destined to inform and protect the End-Users of the Application Under Test when specific, internet telephony attacks do occur, and ultimately to block the capability of the attack perpetrator to induce further damage.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I am profoundly indebt towards Dr. Karon MacLean, who guided my first steps in academic research.

I would like to extend my most sincere gratitude towards my supervisor, Dr. Son T. Vuong, for his invaluable advice, gentle supervision, and critical guidance and support in completing this research.

A kind note goes to Dr. Charles Krasic, for accepting to become the second reader of my thesis.

I would also like to thank the wonderful professors I have found here in the Computer Science department, and who influenced my path during my studies.

A special "thank you" goes to my colleague students Ken, Shahed, and Kapil for the countless tips, advice, and late-night debug sessions.

I am sending a warm "thank you" towards our administrative staff, notably Valerie, Monica, Joyce, Holly, Lara, and Hermie.

This research was supported in part by the Directorate of Telecom Engineering and Certification of the Department of Industry Canada.

Chapter 1
1. Introduction

The widespread access to the World Wide Web, combined with the growing need for better, less expensive, and less hardware requiring – interpersonal communication has given Voice over Internet Protocol a very strong momentum in recent years. With the advent of voice communication over the internet however, one must ask: are we opening the door to the known internet threats into the Voice over Internet Protocol world? Or will Voice over Internet Protocol (VoIP) become the target of unique, specific attacks? Will both scenarios become true? None?

This thesis describes a series of VoIP-specific attacks, as exhibited via a Commercial Off The Shelf (COTS) VoIP application deployed in a laboratory controlled environment, and a mobile-agent based Intrusion Detection System (IDS) designed to warn and protect legit Users of the COTS VoIP application when an intrusion is detected, as well as to identify, and ultimately deny access to intruders perpetrating specific, VoIP attacks against the COTS VoIP application.

1.1. Motivation

This section will describe the motivation behind our thesis work, by presenting points arguing in favor of the need for internet security in general, and VoIP security in particular.

Over the last decade, the computing experience has changed for both business and private users, mainly due to the advent of powerful, yet affordable computing machines. In addition, a significant amount of data has migrated from shelved hard copies to the World Wide Web (WWW), engendering a frenzy of Internet activity all over the world. Moreover, interpersonal communication equally migrated in a significant proportion over the WWW. As a consequence, always on, high speed internet connections have become common in the western world households, and a plethora of service providers has flourished to answer the new, Internet-related needs of the population. One such need is Internet Telephony. Unfortunately, it is an emerging technology and has a number of technological and evolutionary issues. The technological issues are mainly because the Internet was not designed for real time traffic such as voice and video. However, the benefits of using IP as a generic platform for both data and real time applications are compelling enough to encourage resolution of these issues.

In VoIP networks, the vulnerability issues become more prominent. Compared to traditional voice networks, a Voice over Internet Protocol network is a convergence of a signaling network and a data network using Internet Protocol (IP). The use of shared media by VoIP systems opens the door to some uncertainty as to the source of a call. While in the traditional voice networks one has to tap into a specific circuit to eavesdrop, in an IP network any equipment connected to the target LAN can identify, store and playback the VoIP packets that traverse that LAN. An unprotected, unauthenticated IP network makes VoIP susceptible to hostile use, such as prank calls, sending computer viruses or flooding the network. Finally, unlike traditional voice networks which have only "dumb" end nodes (i.e. simple telephone receivers), VoIP must, by its very nature, deploy intelligent end point devices such as computers and/or IP phones, which are connected to open public networks. The public, always-on exposure of the latter devices

will affect the security of the LAN they belong to, by undoubtedly increasing the number of vulnerabilities that LAN will be exposed to.

The following examples illustrate the effect of certain vulnerabilities on VoIP services.

1. The *eavesdropping* may cause decreased quality of service, as VoIP must obey strict real-time packet-delivery requirements. We infer that repeated dropping of random payload amounts during audio connections or voice conferences will induce personal or enterprise customers to find alternative solutions to VoIP. In addition, the VoIP carrier will also reduce its revenue, thus negatively impacting further development of the technology. Furthermore, an attacker can sniff the packets to monitor or record a conversation, and thus gain access to confidential information.
2. The *theft of service* mostly happens when an attacker steals the user name and password of a legal user. The attacker could perform the theft either by doing a replay action at an IP phone, or by data-mining the cache when he gains access to a legal user's browser at the end of his/her call. Since there are no unified standards for VoIP protocol security, vendors have implemented the different protocols according to "in house" knowledge of the current threats. As a case in point, certain vulnerabilities could happen at the terminals during the call setup procedure. For example, during the SIP "register" requests, although authentication will be required in some cases, there are a number of ways to extract the username and password from a Pingtel Sip-based IP phone [1]. Moreover, a malicious party will be able to extract the registration information or call information during the IP phone boot-up or call placing operations.
3. The (*distributed*) *denial of service* can cause the server or router (media gateway controller, media gateway, or soft switch) to shutdown the service. Since encrypted packets may bypass the firewall or intrusion detection

systems, the malicious message may reach the destination port and cause the break down of the service.

We argue that security vulnerabilities lead to loss of confidentiality, integrity, or availability of the information resources, and consequently, affect wide deployment of VoIP technology.

In this thesis, we attempt to provide a solution to some of the problems prone to plague the expansion of VoIP usage. Namely, we propose a Mobile Agent Based Intrusion Detection System which we design to inhabit a specific commercial implementation of the VoIP technology, and destined to protect its Users from specific intrusions. We expose the nature and extent of the attacks the VoIP Application Under Test (AUT) is prone to by performing such attacks in a controlled, laboratory environment, and showing that they succeed. We then deploy our IDS with the declared intent of protecting the VoIP application's Users from these intrusions, and demonstrate the increased level of protection our system offers without interfering with the inhabited application, nor hindering its usage by the legit parties.

1.2. Thesis Contributions

The main contributions materialized in this thesis are as follows:

1. Successful performance of specific internet telephony attacks against a commercial VoIP application.
2. Successful implementation and deployment of a mobile agent-based IDS destined to monitor the VoIP commercial application under test.
3. Design of a heuristic algorithm implementing a gradual attack-response procedure, destined to inform and protect the Users of the AUT when specific, internet telephony attacks do occur, and ultimately to block the capability of the attack perpetrator to induce further attacks.
4. A prototype implementation[1] of our mobile agent-based IDS for VoIP solution, which we name *MAIDS for VoIP* (A Mobile Agent-based Intrusion Detection System for Voice over Internet Protocol).

[1] The prototype was successfully presented at Industry Canada's Ottawa headquarters during a University Demo Day organized May 10, 2005

1.3. Thesis Outline

This thesis is structured as follows:

Chapter 2 is dedicated to the presentation of our work's background. In particular, we detail the Intrusion Detection technology, its metaphors, and advantages and disadvantages. Then, we proceed to introduce the VoIP technology, its two main protocols, and underline its increased potential for malicious intrusion. Last, we introduce the Mobile Agent technology, its concepts, as well as advantages and disadvantages.

In Chapter 3, we briefly present the Components off the Shelf used in our project; namely, the VoIP component, the mobile agent platform, and the firewall component.

Chapter 4 represents the core of our thesis; it is the chapter where we present and detail our MAIDS for VoIP system. In particular, in Section 4.1 we present the genesis of our project, and we follow in Section 4.2 with its mission statement. Next, in Section 4.3, we outline the vulnerabilities the IUT is prone to, as exposed by us under controlled laboratory conditions. We proceed next by explaining our system's algorithm and state machine in Section 4.4, we detail its implementation in Section 4.5, and we present an experimental evaluation in Section 4.6. We close, in Section 4.7, by outlining MAIDS for VoIP's computational cost.

Finally, Chapter 5 presents the conclusion of our work, whereas Chapter 6 closes with our proposed future work.

Chapter 2
2. Background

2.1. Intrusion Detection Systems

An Intrusion Detection System (IDS) is a software (or combination of software and hardware) product designed to automate the monitoring of events occurring either on a specific computer system, or on a network of such systems. Ultimately, its goal is to detect malicious actions performed on a computer system, or several networked systems. Two aspects of an IDS's functionality are important to mention: first, is that an IDS is designed to (at least) detect actions that are not flagged by a regular firewall. Second, is that IDS's have branching topology insertion points: some are designed to use network traffic as input; thus, they are predicated upon the notion of computer networks. Others are designed to act locally at a specific host, regardless of whether the host in question is part of a corporate network or not. Yet others evolved as a hybrid of the two former approaches. Overlapping these three categories, IDS's operate according to an anomaly or a misuse detection operating metaphor.

Since our main contribution is in the form of an IDS for a VoIP application, we would dedicate this section to introducing and detailing the various IDS technologies, presenting both their advantages and drawbacks, and extracting their most empowering features.

2.1.1. Timeline

It is agreed in several forums ([2], [3]) that the first document establishing the importance of monitoring system logs, the gaps in auditing created by access to data by individuals with elevated privileges, the relevance of security audit trails, and ultimately the birth of Intrusion Detection System was James Anderson's 1980 paper [4], which was actually a study he wrote while under contract for a governmental organization. Among the most important contributions made in this paper, we enumerate: the formal definition

of threats, types of users and of hostile penetrations, and the structure of a surveillance system. Then, a classification of attacker types, and the background work and theoretical foundations of actually implementing an IDS on the system he was hired to monitor.

Just three short years later, SRI[2] International designed an IDES (Intrusion Detection Expert System), utilizing statistical anomaly detection and a rule-based signature analysis procedure [5]. It was perfected (by the same organization) in the 1990s to become NIDES (Next-Generation IDES). As a major novelty we note the concept of intelligent user profiles: the system not only utilizes user profiles, but also manages them in a way allowing the system to learn new, or modified behaviors of a legit user of the inhabited system.

Further theoretical groundwork in the modern evolution of IDS's was published in 1987 by Dorothy Denning [6]. In her paper, the author presents the theoretical model of a real-time intrusion detection expert system; once implemented, the system is completely independent of the operating system platform, types of vulnerabilities the monitored system is prone to, or whether the intruder is an external attacker or an insider. As such, the model becomes a framework for intelligent IDS's, and its main strengths are the leveraging of known user behavior in a mathematical model to infer intrusions.

We conclude this subsection by highlighting pioneering work in the area of network-based IDS's: the work by Heberlein et al [7]: it is the first system to use network traffic as audit source. In their paper, the authors present the formal underpinnings of various network attacks, and use security audits to implement a historical data, user profile-based IDS. The system uses a hierarchical model in the data analysis process: we note a *packet*, a *thread*, and a *connection* layer(s), whose goal is to gradually combine the low level captured traffic into abstracted-out logical streams. At the end of the processing chain, a specific security state of the monitoring system emerges. Last, we note that the prototype IDS developed by the authors was successful in detecting a significant number of network attacks, when compared to the rate of success of sole system administrator monitoring[3].

---

[2] The initials stand for Stanford Research Institute, but the organization formally separated from Stanford University in 1970, and adopted the current name in 1977

[3] The observation was made while the system was monitoring UC Davis's network.

2.1.2. IDS Operating Metaphor

We can easily view the theoretical foundation of the intrusion detection (as applied to computer systems) as a specific instance of the signal detection problem. The overall problem is exposed in various published works on Statistics [8], [9], and the interested reader is encouraged to browse these references for a detailed analysis of this topic. As to its application in the Engineering field, the reader is referred to the various academic papers published on this topic; we provide two for reference: [10], [11]. For the purpose of this thesis, we will refer to the particular application of detecting signal from noise. An outline of its gist is as follows:

- Let the null hypothesis H_0 be that the signal of interest[4] is absent from the data to be analyzed
- Let the alternative hypothesis H_1 be that the signal of interest[5] is present in the data to be analyzed
- A hypothesis test is developed, over the entire data space, to infer which hypothesis stands
- If H_0 is erroneously rejected, then a type I error occurs [12]. In IDS terms, this would be a false alarm: the system raises an alarm when in fact, there were no signs of intrusion in the analyzed data.
- If H_0 is erroneously deemed to stand, then a type II error occurs [12]. In IDS terms, we arrive at the false dismissal situation: the system does not raise an alarm, when in fact, the signs of intrusion were present in the analyzed data.

Thus, as McHugh et al explain [13], if a given IDS focuses on the presence or absence of the signal in the data, then it is classified as *misuse-based.* Conversely, if an IDS focuses on the analysis of the noise (rather than the signal), then it is classified as *anomaly based.* And we can easily infer that a *hybrid* operating metaphor is also possible.

[4] i.e. a sign of intrusion
[5] Ibid.

2.1.3. IDS Sensing Metaphor

McHugh et al also distinguish IDS's based on the phenomenology that they sense:

- *network-based* systems analyze network packets. Usually "listening" on a network segment, this type of device is designed to protect several hosts.
- *host-based* systems analyze information available at an individual host. Carefully inspecting specific audit data, this type of system is designed to protect a single host (typically a critical system such as a web server, etc).

In this space as well, we can infer that some IDS's can be designed to function based on a *hybrid* approach.

2.1.4. Network Based Intrusion Detection Systems

A Network IDS (NIDS) predicates its actions on the concept of network. That is, its data collection sensors are placed at various points along the physical links between the hosts (or network segments) of the monitored system. Its main components are a series of sensors (or hosts), which are to be placed at key points within the monitored network. Different configurations thus result, but the main advantage remains the ability of the sensors to become ubiquitous, yet "stealth"-like when it comes to being detected.

By monitoring (mainly) incoming traffic, a basic NIDS is able to detect major attacks such as denial of services (DOS), port scans, buffer overflow and various probing attacks. In its more sophisticated incarnations, NIDS's are geared towards detecting shell code payloads – segments of machine code whose execution will result in granting an attacker administrative access to a target host. In the following table, we briefly outline the advantages and disadvantages of this type of IDS [14]:

NIDS Advantages	NIDS Disadvantages
• Highly adaptable to cross-platform environments	• High rate of failure once a certain traffic level has being reached
• Little impact on monitored network	• Severely limited capability on switch-based networks
• Easy to harden against attack against self	• Cannot analyze encrypted traffic
• Easy to upgrade	• No access to specific hosts to pursue investigation; this results in little feedback with respect to extend of damage of a given attack
• Afford central management	• Difficulty in dealing with fragmented packets

Table 1: Advantages and Disadvantages of NIDS

As Table 1 shows, NIDS have in their favor the ability to monitor multiple hosts, while disclosing little, if any, about their presence. However, as McHugh et al [13] point out, keeping state for an extended period of time on a high throughput network is one major challenge. A second, arises when we consider the fact that more and more of today's networks are switch-divided into smaller segments. And even when the (main) switch features a monitoring port, it cannot or simply does not (by design) mirror all traffic traversing the switch [14].

One option for dealing with the first drawback is to use a "slicer" NIDS-component [16] to split the very fast (say Gigabit, or some form of optical link) traffic into more real-time-manageable streams. The challenge here however is to keep the overall context at the assembly station. To deal with the second drawback, the NIDS can be configured with a large number of sensors, which can be strategically placed such as to mitigate the switch-based network partitioning. Managing the resulting sensor-array becomes then a self-inflicted problem, and the same stands for the resulting reduced, or increased processing complexity to maintain, equivalent capability to detect specific routing and spoofing based attacks [17].

Lastly, we discuss the topic of monitoring sensor placement, and we begin by pointing out two drawbacks of the up-scale, stealth-like[6] monitoring sensors [20]: first and most significant is lack of reflex responses[7], and second is inability to monitor traffic in both directions. At the other end of sensor placement complexity, we have hub-based sensors. They are much favorite for their low-cost and drop-in installation[8], but are collisions prone, and most importantly, constitute a single point of failure.

As main examples of NIDS, we highlight the following open source[9] systems.

We begin with the SNORT system [18]; a highly popular, open source NIDS capable of performing a comprehensive series of protocol analysis activities. SNORT[10] is a real-time, signature-based[11] traffic analysis tool. It has become almost the de facto standard within the NIDS community, to the point where full books are solely dedicated to it [19], [20]. Snort can perform – in real time – protocol analysis and also pattern matching. Important to note is that the product features a highly customizable rule based language [21]. Lastly, we mention its ability to accept – by design – a wide variety of specific plug-ins. Among its attack-detecting abilities, we enumerate: CGI attacks, SMP probes, buffer overflows, and various port scan attempts. Upon such detections, SNORT can be configured to initiate a variety of SysAdmin[12] notifications: mailing, (syslog) logging, UNIX sockets, and finally various messaging options in the Windows environment via Samba's *smbclient* [13].

We continue the open source examples of NIDS with SHADOW [22]. Entirely based on other available open source libraries, SHADOW has, at its core, and in a highly simplified rendering, a Perl[13] engine processing output from the *tcpdump* and *libcap*[14] libraries. As with any NIDS, there exist a wide variety of possible configurations for the number and placement of monitoring sensors. Of note here is that sensors are by definition un-trusted, even by the IDS itself. They run only a handful of software

---

[6] Devices connected to the network but which to not have an Internet Protocol address.
[7] Ability to take a predetermined action in response to a rule being triggered.
[8] Do not require special installation procedures.
[9] Managed under the Open Source Initiative (OSI); for details see [15].
[10] We note here for completeness that there exist a SNORT-based commercial application: Sourcefire Intrusion Management System.
[11] This specific IDS operating metaphor was introduced in 2.1.2, and will be detailed in 2.1.8.
[12] Security Office; common position in corporate environments
[13] Perl programming language created by Larry Wall
[14] (UNIX) Libraries developed at Lawrence Berkeley Laboratory

components, and communicate exclusively with the analyzing engine via SSH[15], thus accomplishing the design goal of hardening the IDS against attacks targeting it. Finally, its reporting capability is accomplished via an HTML[16] report page driven by a series of CGI[17] scripts. Like SNORT, SHADOW is a very popular IDS for the UNIX platform and its various clones.

We conclude the open source-based NIDS enumeration with Bro [23]. The Bro IDS functions according to a three-layer, closed feedback loop metaphor: each layer processes data provided by its (lower) predecessor, does its own processing, and forwards the results to the next layer up. It can, in turn, generate events that will alter the processing operations of its immediate predecessor. The bottommost layer, *libcap*, "listens" to the raw network traffic and does a limited amount of initial processing via the *tcpdump* library. The middle layer, the *event engine*, separates legal packets[18] form the rest, and instantiates specialized handlers for each connection. The uppermost layer, the *policy script interpreter*, generates highly specialized event handlers, which are written in Bro's own scripting language. This provides the added ability to extend, in a plug-in-like manner, Bro's capabilities by writing additional event handlers to suit specific needs. The Bro IDS also sets a major precedent by featuring built-in capabilities to handle specific attacks against self: *overload*, *crash*, and *subterfuge* attacks. Briefly, an overload attack is a specific incarnation of the known DOS[19] attack, a crash attack is one in which the attacker attempts to find a flaw in the IDS's implementation, and a subterfuge attack is a highly sophisticated attack-avenue which attempts to mask malicious traffic as benign. Bro handles the first attack via built-in thresholds, the second via an inner timer guarding the per-packet process time and which upon firing forces a shift in the system's state, and finally a complex decision-tree logic and software implementation attempts to mitigate the third[20].

---

[15] Secure Shell – a network protocol affording a secure communication channel between two remote computer hosts.
[16] Hyper Text Markup Language – current de facto standard for markup language.
[17] Common Gateway Interface – standard protocol for interfacing external applications with a local web server.
[18] A TCP/IP packet can have a malformed header (i.e. not conform to standard)
[19] Denial of Service
[20] We infer its similarity with chess-playing programs, where the program writer takes into account an as-comprehensive-as-possible set of possible next moves

2.1.5. Host Based Intrusion Detection Systems

As opposed to a NIDS, a host based IDS (HIDS) is installed at specific hosts, and its sensors monitor core files or specific objects; generally, they are compared to fingerprints from a secure database. Alternatively, the HIDS creates such a database upon installation on the target host, either by recording the attributes and access lists associated with core files, either by using a checksum technique (MD5[21] is a common standard) to fingerprint files of interest. One main challenge in designing such systems resides in the fact that a large proportion of the artifacts of interest are dynamic in nature, and as such, unsuitable for checksum techniques.

In the following table, we briefly outline the advantages and disadvantages of this type of IDS [14]:

HIDS Advantages	HIDS Disadvantages
• Negligible network overhead	• Usually operating platform dependent
• Can detect attacks invisible at network level	• Overload of local resources: CPU, memory footprint, and often local storage of logs and data
• Not affected by switch-based subdivisions of a given network	• Need to be deployed at every host to be monitored; also needs to be managed separately at each host
• Lower cost of deployment and ownership (compared to NIDS)	• Cannot tell whether a given attack is also targeting other hosts on the same network
• Capable of detecting various Trojan and BackDoor intrusions as an artifact of analyzing operating system core logs	• By comparison with NIDS, easy to attack and even disable

Table 2: Advantages and Disadvantages of HIDS

[21] Invented in 1991 by MIT Professor Ronald Rivest to replace MD4

We distinguish thus as one of their main advantages the fact that by design, such systems can detect attacks who do not generate externally visible behaviors [13]. Yet another resides in their ability to audit a host with unsurpassed precision: a thorough mapping between logged-in users and running processes can be obtained; a feast outside of the NIDS's reaches. HIDS's have two main data sources [14]: operating system audit trails and system logs. The former provide a much detailed trail of ongoing activities, and they are also better protected. The latter however are much less cryptic, and also have a significantly smaller footprint. HIDS's have two main disadvantages: first, they tap into the monitored host's processing power, and all depending upon the host's hardware design and HIDS's complexity, performance can suffer dramatically. Second, HIDS's cannot conceal their presence as NIDS's do, and thus can be targeted by very specific attacks, and ultimately disabled[22].

On the topic of monitoring sensors, we point out pioneering work by Kerschbaum et al [25] which suggest embedding the sensors into the operating system kernel – as opposed to the traditional, stand alone running process technique. The main advantages behind this novel approach are native resistance to tampering and not overloading the monitored host's resources with one, or more extra processes which have to run whether there is an intrusion in progress or not. Perhaps the most important advantage resides in this type of sensor's ability to greatly reduce false positives: the sensor is coded to trigger on a specific attack pattern (no difference here as compared to traditional ones), but it is embedded in the specific UNIX[23] kernel layer responsible for handling the specific system calls the attack targets. Not negligible is also the fact that such a sensor does not rely (nor needs to) on log data or other audit trail. The authors however also carefully point out that by their very nature, such sensors can become serious performance bottlenecks if they are poorly implemented, or even bring the entire system to a halt.

As main examples of HIDS, we highlight the following commercial or open source[24] systems.

We begin with NADIR [27], originally developed at the Los Alamos National Laboratory for internal use. As such, it is not surprising that one of its main design goals

[22] Specific, kernel level rootkits are known to accomplish such actions [24].
[23] The authors used a UNIX clone for their implementation: OpenBSD [26].
[24] Managed under the Open Source Initiative (OSI); for details see [15].

was protection against internal security policies violations, mostly by an internal external attacker. From the onset, we mention that since NADIR was designed with very specific security goals in mind, its designers and operators had a very precise idea as to what sort of intrusions they want the system to handle, which events should be logged, and what actions should be taken for each identified intrusion. This stands to a much lesser extent in the case of typical corporate environments, where the local culture does not have to comply with a highly structured and rigid discipline. We equally draw attention to NADIR for its introduction of user profiles: they contain static, granular information about a user's behavior, such as log-ins, terminals used, all predicated on a specific time line. Furthermore, the profiles are refreshed on weekly basis. On the other hand, a master security profile was built taken as input the set of internal security policies, other audit trails, and interviews with security experts. The master profile is then partitioned into a set of expert rules, and the single user profiles are mapped against it. As Hochgerg et all point out, though the system was very effective in detective policy violations, it also had a high rate of false positives – which however were not a problem given the particular environment where it was deployed. We infer that it remains thus unclear how the system would perform, and how it will be tolerated in a regular organization environment.

We close our survey of HIDS's with IDIOT [28]. Its main governing principle revolves around the use of colored Petri nets[25] to perform state transitions leading to intrusion detection. From an architectural point of view, IDIOT features three distinct layers: information, signature, and matching. The information layer's role is to preprocess the audit data, in an effort to increase the platform indecency of the system itself. The signature layer provides a text-based description of the intrusions the system can detect, and the matching engine provides the processing power for a match between a unit produced by the signature layer into one produced by the information layer. To enable the functioning of the Petri nets automaton, the user must insert a significant number of code segments in the kernel, and hook a majority of system calls with custom wrappers; we infer this as the major drawback of IDIOT. Once that is accomplished however, the automaton functions flawlessly: for every pattern of attack, a token is placed in each starting state. Then, when the appropriate guarding condition becomes true, a transition

[25] See [29] for an introduction

occurs, moving the token to the next state, until the final, alarm-generating one is reached. One of the system's highlights is its ability to spawn a matching engine from a given information layer unit, and attach it to a running instance of the parent system (via the user interface). Yet another stems from the additional work the authors have done to classify intrusions and to capture them in the signature layer's units. In all, the system appears to have very solid theoretical foundations, which increase the confidence in its performance. We infer as its main deployment difficulty the fact that on every target platform, the operating system must first be instrumented with the code instructions affording the functioning of the information layer.

2.1.6. Hybrid Intrusion Detection Systems

Hybrid – from a sensing phenomenology perspective – IDS's (HYIDS) combine the data gathering methods characterizing the two formerly mentioned IDS types to create a highly specialized intrusion detection system. More specifically, the IDS becomes highly focused in auditing and monitoring a particular application residing on a host (its host-based arm) whilst placing its information-gathering sensors at various points along the network the monitored host is active on. This type of IDS is thus operating in protocol enforcing mode: for example, a very common implementation would be the protection of an HTTP server, whose protocol is well known and followed by the World Wide Web oriented applications.

As main examples of HYIDS, we highlight the following commercial or open source[26] systems.

We begin with RealSecure [30]. RealSecure is a commercial application, targeting corporate environments. As such, it provides for complex auditing trails, affording the gathering of proof required to formally accuse someone (say an employee) of breaching the company's access and usage policies. From a low level perspective, RealSecure feature a two layer architecture: the bottom layer is composed of *sensors*, which in turn is composed of *network*, *server*, and *operating-system* sensors. The *workgroup manager* composes the upper layer. From a high level perspective however, RealSecure presents a

[26] Managed under the Open Source Initiative (OSI); for details see [15]

rather three layer architecture [13]: a *network-based* engine, a *host-based* engine, and an *administrator module*. The two engines complement each other, in an attempt to mitigate the shortcomings of NIDS's[27]: it is rather trivial for a logged in, legal user, to undertake hostile actions undetected by the network-based sensors. In addition, the outcome of an attack cannot be inferred based on network-based sensing phenomenology. It is to be noted however that the extra features come with a price: the authors explain that the host-based sensors consume between 2% and 4% of the host processor for a single user system, and though they mention that the impact will be greater on a multi user system, they do not provide specific data. Lastly, we found interesting that the implementers of the system rely on accruing the weight placed on host-based sensors to mitigate for the "blindness" of the network-based ones when forced to work on a switched network.

Our last example in this category is DIDS [31]. The system leverages its hybrid architecture to mitigate for hosts which are not monitored by its host-based sensors: their actions are scrutinized via the network sensors. Of note, is that DIDS incorporates two other IDS's in its making: NSM [7], which we introduced in Section 2.1.1, and Haystack[28]. These two stand-alone IDS's proxy for the two obvious architectural layers of DIDS: the *host monitor* and the *LAN monitor*. A *DIDS director* composes the third layer, and its role, as its name implies, is to process the flow of data generated by the sensors part of the two sensing layers. The host monitor is the more elaborate sensing device among the two: it actually incorporates signature matching processing power. In addition, it features an events processor which decides which of the observed events[29] is to be forwarded to the DIDS director. A similar director-forwarding function is performed by the Haystack-based element, only this time for session profiles the latter builds. The LAN monitor (an instance of NSM), accomplishes its director-forwarding task at the network event level. Finally, the DIDS director has a communication and a decision-taking arms.

[27] Tables 1 (Section 2.1.4) and 2 (Section 2.1.5) provide a summary
[28] For an overview, please see [32].
[29] Log-in attempts, etc

2.1.7. Misuse Detection Operating Metaphor

As explained in Section 2.1.2 (IDS Operating Metaphor), a *misuse-based* system underpins its operating metaphor on a very specific task: while analyzing the audit data, find the event (or superset of events) matching a known "criminal fingerprint". Obviously, the system must carry with it a database of offending fingerprints at all times. The most common type of such fingerprints is a text-form pattern capturing, say, a specific section of a particular network packet [13]. Such patterns are commonly known as *signatures*[30] [14]. A next, medium complexity level of signature is one encompassing a family of attacks [14]. Meaning, a common packet structure core belonging to more than just one attack can be identified and extracted, such that one signature will detect the entire family. This type of automated reasoning is known as *forward-chaining* reasoning [33]: the system deduces a known, multi-step scenario has occurred based on a series of asserted-facts and rules one-to-one mapping. In its most intelligent form, misuse-based IDS's can build and maintain a complex state machine or neural network mapping sensor data to abstract attack representations [13]. One such system is STAT [34]: a novel model for representing intrusions, based on the analysis of a state machine where the final state is the compromised-host state. However, as McHugh et al point out [13], the state machine technique often has the drawback of requiring training on labeled sensor data, but has the significant advantage of rendering the system capable of detecting previously unseen attacks as long as they are abstract equivalents of known patterns.

In the following table, we briefly outline the advantages and disadvantages of this type of IDS [14], [16]:

[30] We will be using this term thereafter.

Misuse Detection Advantages	Misuse Detection Disadvantages
• Highly effective sensor technology	• Can only detect attacks it know about
• Fast attack pattern recognition	• Continuous signature updates are required
• Independent of operator experience level	• Except for state-based systems, variants of known attacks can be detected
• Very low rate of false positives	• Signature development requires highly skilled personnel
• Possibility to fingerprint normal (i.e. legal) behavior as well	• State-based systems have prohibitive development costs and logistics

Table 3: Advantages and Disadvantages of Misuse Detection

As mentioned in Table 3, the main drawback of this type of intrusion detection resides in the rigid mapping between a given signature and the forensic data it applies to. This implies that the slightest variation in a given attack – for which a signature exists – will pass unnoticed. It is precisely this shortcoming that state-based systems are trying to address, albeit there exists a non-trivial price to pay in increased operational complexity. We close the introduction of the misuse intrusion detection operating metaphor by pointing out that IDS's operating under this aura can be built around any of the sensing architectures we already presented[31].

We begin our mini-survey of misuse detection implementations with ASAX [35]. The system revolves around two cornerstone computing arms: a first, named NADF by Habra et al, has the role of converting the raw audit trail data into a canonical format. Next, a second enables performing all required data processing in one pass, via the novel RUSSEL language used by the authors to express queries on audit trails. One of the main design reasons behind NADF was, as its name implies (Normalized Audit File Format), to increase the portability of the system such that the IDS can run on virtually any platform[32], whereas maximizing efficiency was a second. Thus, to enable ASAX to run

[31] Sections 2.1.4, 2.1.5, 2.1.6.

[32] A co-sponsor of the project is a private corporation (Siemens Nixdorf Software S.A.) marketing its proprietary flavor of operating systems: BS2000 and SINIX [35].

on a given platform, a required implementation step would be to write a *format adaptor* translating the native operating system's audit trail data into NADF. The format adaptors also produce a set of auxiliary files preserving the mapping the two data formats. Finally, the authors make the case for the analysis being performed via the specially designed RUSSEL language. The authors draw on evidence from similar, distinctive language-based implemented systems that the data-analyzing power of such systems increases dramatically. RUSSEL in particular, is a declarative rule-based language specifically written to process NADF's digest of inhabited system native audit data. In the authors' own words, "RUSSEL could be viewed as a procedural language including a particular *predefined control structure* which is suitable to make reasoning about sequences of records" [35]. Among other things, the language affords a regeneration of rules where for each processed record, the application of current rules database generates new mappings which the system stores for future use. The authors recognize however that RUSSEL is assuredly too cryptic for the average potential user, and thus envisage the creation of a higher level abstraction (dubbed RUSSEL2) and empowering SysAdmin's with the ability to better converse with the system. Lastly, we will mention that no formal evaluation of ASAX has been perform by Habra et al, though the paper features enough detailed examples to leave no doubt as to its potential.

We continue our mini-presentation of IDS's utilizing a policy-based approach with USTAT [36], [37][33]. Among the first things we note is that USTAT is that it is based on the original STAT design [34], which we briefly introduced at the beginning of this section. What USTAT is trying to address, is two of the main shortcomings of audit-trail based IDS's: the fact that only a highly skilled technician can write rules that correctly map an intrusion to the raw audit data, and the fact that a new rule is required for even the slightest attack-variation. USTAT[34] thusly bases its foundations upon the concept of state transition: a system starts in a known safe state, and, via transitions composed of various hostile actions arrives in a final, compromised state. USTAT was implemented on a SunOS 4.1.1 featuring the C2-BSM (Basic Security Module) add-on package. Architecturally speaking, USTAT is predicated upon the following components:

[33] Our presentation is based exclusively on [36].
[34] For brevity, we will not differentiate between contributions made by STAT [34] and USTAT [36].

a *preprocessor*. Then, a *knowledge-base*, with the following subcomponents: a *fact-base* (comprising a *fact-base initializer* and a *fact-base updater*), and a *rule-base* (comprising a *state description table* and a *signature action table*). Next, an *inference engine*, and finally a *decision engine*[35]. Briefly, the role of the preprocessor is to parse, filter, and process the audit data for the inference engine's consumption; it accomplishes a mapping of the raw audit data to a set of *actions* known to the system. The first component of the knowledge-base, the fact-base, consists of sets of files with the identified characteristic of being vulnerable to specific attacks; it is created by the fact-base initializer, and maintained by the fact-base updater. The second component, the rule-base, is basically a state-transitions repository: it defines artifacts accurately describing an attack based upon audit data and the fact-base as inputs. The inference engine constitutes the brain of the IDS: it can be abstracted to a conglomerate of algorithms fetching and executing the rules in the database at every inference step, without caching any matches for future use. However, partial matches encountered during an in-progress examination are stored in the fact-base such that it can be used throughout, until the system reaches a conclusion. Lastly, the decision engine represents the vehicle used to communicate with the SysAdmin. One of its implemented features is a *decision-table*, where specific messages are inserted for each successful transition to a new state. Otherwise, the decision engine boosts the same set of SysAdmin-interaction actions as the majority of IDS's: various warnings, notifications, but also suggestions as to how human intervention could prevent the system to reach the next state, and recover from the current one. The work by Ilgun et al stands out also due to the fact that the authors proceeded to formally evaluate the system for both completeness and performance. Performance-wise, and not at all surprising for a host-based implementation[36], the system had a very small footprint under a light load, but degraded to 13% processor usage under a more intensive one. We characterize USTAT however as one of the most complete works we surveyed for our thesis.

We conclude by introducing the work of Chari et al: the BlueBoX system [39]. In designing their system, the authors use the method of system call introspection to create a

[35] Which incidentally constitute the backbone of a typical expert system [38].
[36] We have introduced this shortcoming in Section 2.1.5.

blue print of the core capabilities of the inhabited kernel[37]. The blue print is then used to control access to system resources, via a set of policies expressed in a textual language. The authors do acknowledge as the main drawback to the overall approach the fact that intensive, time-consuming, and highly pertinent expertise is required to instrument the native kernel with the calls the IDS will trigger upon. To mitigate this, they suggest using a *ptrace*-based kernel monitoring infrastructure, such that only the processes of interest are monitored and altered as demanded by the IDS's logic. As for the rules themselves, they are defined as policies governing which system resources a given process may access or not. In a nutshell, the system uses the concept of a *white list*, whereas what is hard coded in the policies is what is specifically *allowed*. From an architectural standpoint, BlueBox features a two layer structure: in the user space resides a rules *parser*, converting human-specified text rules into binary files. In the kernel[38] space, at system call level, resides a policy *enforcer*. Its role is to intercept core system calls, and cross-reference their parameters against the white list. We consider as a major contribution of this paper the fact that the authors took extra steps to minimize the impact on the kernel. In particular, the enforcer is written as an independent module, except for "about 10 lines of assembly and 20 lines of C code" [39]. As for the rules themselves, they are founded on a straight forward principle: "Since system resources must be accessed through system calls, disallowing invocation of a system call disallows access to resources" [39]. Chari et al distinguish three types of such core resources: file system objects, uid/gid lists, and signals; they consequently produce rules mapping against these types of resources. The authors fully instrumented the Apache 2.0 httpd server for this purpose, and proceeded to a formal evaluation, where the impact on performance was measured as well. Overall, the authors estimate a performance degradation in the range 8% - 10%, while noting that for the case of dynamic content web hosting this penalty will increase observably, due to the spawning of new processes (by the monitored server) whose rules must be thusly loaded. Nonetheless, we retain the work done by Chari et al as significant, due to the novel manner in which the system calls are hooked in order to

[37] The authors implemented their system on a Linux platform.
[38] Linux 2.2.14 kernel was used for evaluation.

minimize kernel impact. Furthermore, the authors identify a series of optimizations that will further diminish the performance impact.

2.1.8. Anomaly Detection Operating Metaphor

By contrast with a misuse detection metaphor, an anomaly based technique is predicated upon the IDS learning the protected system's normal behavior, such that it can immediately flag anomalous one without the need to cross reference its patterns into a signatures database[39]. As Ilgun et al explain [34], the roots of this approach have being seeded by Anderson in his ground-breaking report [4], where three classes of malicious users are identified: the *masquerador* – an attacker impersonating a legal system user, the *misfeasor* – a legal user engaging in malicious activity, and the *clandestined* user – an attacker employing various techniques to hide the audit traces of his or her activity. What is more, Anderson infers that activities performed by the first two categories of illicit users can be detected via audit trail monitoring. Obviously, this becomes possible once the legit audit trail is known, and can be used for cross reference; then, the IDS's task is to continuously monitor the inhabited system's logs for deviations. In Table 4, we detail a few of the specific techniques used in anomaly detection IDS's [14], [34]:

[39] We have provided more theoretical insight in Section 2.1.2.

Technique	Description
• Threshold-based	• The simplest technique. • Encompasses specific attributes as hard-coded counts. Once reached, an alarm is triggered. • Example: the number of files a user can access or delete in one session, the number of processes he or she can launch, the number of allowed failed login attempts, etc. • It need not be static; it can be designed to change weekly, or according to some other heuristic. • Inferring the correct threshold for a specific system is a non trivial exercise.
• Statistical (Profile)-based	• This technique uses statistical measures. • We distinguish two kinds: o Parametric: the measured attributes are assumed to comply with a particular pattern o Non-parametric: the measured attributes are inferred from historical data
• Rule-based	• Similar with statistical profiles – in the sense that the acceptable usage is defined. • Instead of quantifiable values, this approach uses rules.
• Novel techniques	• We encounter here ([40], [41]): o Neural networks o Genetic algorithms o Immune system models

Table 4: Techniques Used in Anomaly Detection

As inferred in various forums ([14], [16], [34])[40], anomaly employing IDS's are fundamentally based on the assumption that all intrusive activity is *anomalous*. However, since we know that there exist intrusive activities which are not anomalous (or can be

[40] And analyzed in Section 2.1.2

presented as such to the IDS), we thus infer the main drawbacks of this approach: false *negatives*. We consider them far more dangerous than false *positives*, as the latter case is more or less of a nuisance when compared to the danger of treating intrusive activity as normal. In the following table, we briefly outline the advantages and disadvantages of this type of IDS [14], [16], [34], [39]:

Anomaly Detection Advantages	Anomaly Detection Disadvantages
• Detect attacks unseen before	• Unacceptable number of false positives, in most cases.
• Produce information that can be used in the creation of signatures for the misuse-based IDS's	• Require training data and training time before they can be put to use on a real[41] network
• More accurate than humans experts when learning the behaviors of a category of users	• The circumstances of real network implementation often result in high difficulty in inferring "legal" patterns of usage
	• Knowledgeable attackers can gradually train the system to accept their actions and treat them as legal
	• The set of statistical parameters to be measured and monitored is often specific to a given network. As such, it is hard to infer a blue-print applicable across several network and user configurations
	• When the usage patterns of a given network change, the system must be trained again to incorporate these new behaviors in its knowledge base of legal actions

Table 5: Advantages and Disadvantages of Anomaly Detection

[41] We mean by that outside of a laboratory environment; at a customer's site.

Last, it is important to understand that the fundamental difficulty in implementing an anomaly-based IDS resides in defining what *normal* behavior is for a given system. Furthermore, this definition must continuously update itself at the same pace as the inhabited system changes its hardware configuration, its usage pattern, or both. Forrest et al [42] for example, point out a main shortcoming of previous work: the focus used to be towards an overly encompassing definition of normal behavior, and this opened the door to this logic breaking at the smallest variation in the inhabited system's configuration. As such, Forrest et al attempt the opposite approach, whereas the definition of normal is predicated upon a core set of short system call sequences (translated into signatures) which the authors prove have low variance over a wide range of normal operating conditions, while still exhibiting noticeable perturbations when anomalous behavior occurs.

We close the introduction of the anomaly intrusion detection operating metaphor by pointing out that IDS's operating under this aura can be built around any of the sensing techniques we already presented[42].

As main examples of IDS's employing the *anomaly detection* metaphor, we highlight the following commercial or open source[43] systems.

We begin our mini survey with "Wisdom and Sense" (W&S, hereafter), an IDS written by Vaccaro et al [43]. W&S is a rule based anomaly detection IDS, as outlined in our introduction to this section, Table 4. It is important to note that the system does not employ built in templates at time zero; rather, starting point rules are created from audit data prior to the system going "live". The latter term is otherwise not a casual insertion, as W&S is a real time IDS. The rule-building process consists of an initial pass over a set of audit logs[44] whereby a rule-creation program transforms the audit records in an in-memory data structure containing the rules. We infer this as a major flaw of W&S: the data used to construct the rules defining "good" behavior are likely, if not bound to contain traces of intrusions ("bad" behavior). The authors specify that "[...] these are heuristically filtered out of generated rules" [43], but fail to detail how this is achieved. Likewise, the authors do not explain the heuristic used to infer the alarm threshold for a

[42] Sections 2.1.4, 2.1.5, 2.1.6.
[43] Managed under the Open Source Initiative (OSI); for details see [15]
[44] W&S is implemented on a specific hardware and software platform, both made by IBM.

specific user thread (detailed hereafter). These two details aside, W&S pushes the rules approach one step further by providing an aggregation algorithm (whose end product the authors call *threads*), such that discrete observations related to a given user (or host) can be combined. In action, if the arrival of a new audit record pushes a given thread's score over its critical threshold, an alarm is raised. As in-memory data structure, Vaccaro et al leverage *forests*, which are refreshed on a recommended two-week cycle. From an architecture standpoint, W&S is build around three components: a *data processor*, a *rule base generator*, and a *transaction analyzer*. The transaction analyzer is responsible for the matching process; this is mapped around the structure of a rule: a left-hand-side (LHS) rule-component contains a (numeric) threshold and the sequence of events required to achieve it, whereas a right-hand-side (RHS) term contains the rule's conclusion. An incoming record will trigger the rule if the LHS component is satisfied, and a RHS matching process will follow (we note here that in the absence of a RHS component, the behavior is considered "normal" even for a perfect LHS match). We find worthy of mention the efforts Vaccaro et al underwent to keep the rules' forest memory footprint low, for allowing the SysAdmin to view and edit the current set of rules, and even create new ones which the IDS will incorporate, and for insuring the matching process is reasonably fast for an average rule-load[45]. Last, we mention that the system was thoroughly tested in a laboratory environment, though under a very specific application umbrella[46].

We continue our presentation of the anomaly intrusion detection metaphor with IDES [44], one of the pioneering works involving the use of statistical measures predicated upon the concept of profiles – for both users and hosts. Its authors present IDES as "intended to provide a system-independent mechanism for real-time detection of security violation, whether they are initiated by outsiders who attempt to break into a system or by insiders who attempt to misuse their privileges" [44]. It accomplishes that goal via the use of profiles, for both users and hosts; in the latter case, remote hosts are likewise profiled to detect unusual activity originating from outside the protected system. If such a profile does not exist yet for a specific user, then IDES will start one from a

[45] 20-40 transactions per second, for a 100,000 rules forest [43].
[46] The IDS is proprietary, developed at the Los Alamos National Laboratory.

default template. The IDS mines specific audit data for the purpose of updating the user profiles on an ongoing basis. To achieve this, all hosts must install a data-collection tool whose goal is to transform raw audit data into a digest the system can use; a typical inconvenience of host-based IDS's. A specially crafted *aging* process is build into the profile updating routine, such that for a typical period of thirty days, the last half of the month weights significantly more than the first. From an architecture standpoint, IDES's mass is divided into four components as follows: the *target domain*, the *realm interface*, the *processor*, and the *user interface*. In IDES terms, a *realm* is a logical grouping of monitored hosts. Thus, a target domain consists of the set of realms IDES monitors at a given point in time. IDES's processor, as the name implies, is the computing engine where the collected information is analyzed. The user interface component is self explanatory, and the realm interface is an entity saddling the target domain and the processor components, respectively. As such, it features two modules: a *realm client* – geared at the protected hosts, and a *realm server* – geared at the processing entity. The realm interface's main role is data collection and transfer. We close our presentation of IDES with a brief overview of its brain: the statistical processor. The major parts of the processor are: the *interprocess manager*, the *activity processor*, the *anomaly detector*, and the *profile updater*. The interprocess manager plays a center role, insuring the communication flow between the other components; it is chiefly a dispatcher. Also, upon the initial start, it loads the required statistical parameters from the system's permanent storage. The activity processor the data received from the realm interface into an *activity vector*, containing the statistical parameters the systems uses as numeric measures. Once the vector is constructed, it is send over to the anomaly detector, whose role is to perform the specialized computation required for each of the parameters stored in the activity vector. The profile updater, as its name implies, is in charge of updating the user profiles with the relevant data collected during the day, and for integrating the aging algorithm before saving the profiles to permanent storage. Last, a presentation of the statistical computation required to infer whether an observed activity is within the normal bounds for a given user (or host) is beyond the scope of this thesis; we retain that the SysAdmin has the ability to revert a decision made by the system if he or she considers that a specific activity was misclassified as normal by the system.

We close our anomaly based intrusion detection survey with the work by Sekar et al [45], which build on the groundbreaking, system call sequence-based redefinition of the normal behavior concept we introduced above (Forrest et al [42]). Sekar et al significantly improve its implementation however, by developing a superior state keeping mechanism: a compact FSA[47]. The IDS build by Sekar et al bases its underpinning functionality on the fact that anomalous behavior produces system call sequences not otherwise generated by legal iterations over the operating system's built-in capabilities. Both works predicate thus their work upon first obtaining the chain of calls characterizing normal system usage. Sekar et al stay away from the classic method of modifying the kernel[48]; instead an available library for process tracing[49] was utilized. Likewise, both works leverage the concept of *N-grams*, whereby a given chain of system call sequences is represented as string of fixed length N. Sekar et al improve not only on the classic N-grams algorithm developed by Forrest et al – which limits both the length of the N-grams and their number, but also on previous work employing FSA's – by first providing an automated process for FSA construction, and second by incorporating a *program counter* in the state information, which ultimately enables Sekar et al's FSA to detect attacks missed by all previous work leveraging N-grams or manually constructed FSA (Sekar et al provide references to such works, and we encourage the interested reader to follow the respective pointers). As a last noticeable improvement, Sekar et al's IDS was enhanced by mitigating the typical difficulty arising when the currently running program makes external library calls which in turn make system calls. This is achieved by "record[ing] the location from where the library function was called, rather than recording the location within the library code from where a system call was made." [45]. Among the achievements of their solution (when compared with previous work), Sekar et al enumerates: faster learning time, better detection rate, reduction in the false positive rate, and a faster detection. Furthermore, Sekar et al proceeded to a formal testing of their system, and the IDS under test was instrumental in detecting almost all buffer overflow, Trojan Horse, maliciously crafted input, and Denial of Service attacks. We note that despite the paper's focus on the formal analysis of the theoretical underpinnings of the

[47] Finite State Automaton [46].
[48] We introduced this technique in Section 2.1.5; also featured in our presentation of IDIOT [28].
[49] *strace* is a classic example for UNIX systems

presented system, Sekar et al did build a full featured anomaly-based IDS, which in the case of the HTTP protocol, was tested on a live web server (FTP vulnerabilities were evaluated in a laboratory environment).

2.1.9. Hybrid Detection Metaphor

A Hybrid IDS in this category is designed with the goal of addressing the shortcomings characteristic of the two operating metaphors outlined above[50]. In particular, such a system attempts to mitigate the chief drawback of anomaly-based IDS's – the high rate of false positives, and the misuse based contra part – the inability to detect anomalous activity for which a signature does not exist in the IDS's database.

One such example of IDS is NIDES [47], a natural extension of IDES[51]. Its underpinning idea is straightforward: detect the "unknown" via the statistical, user-profile based component, and the "known" via the misuse based component. For the latter's implementation, NIDES's authors leveraged Lindqvist et al's P-BEST (Production-Based Expert System Toolset) rule set[52]. We remind our readers that P-BEST uses a forward-chaining rule-based matching technique, whereby an audit trail is analyzed via increasing stepping through the rules, and whose subsequent matching allow the system to infer the occurrence of a multi-stage intrusion scenario. As an enhancement over Lindqvist et al's implementation, NIDES affords the modification of the rules data set in real time, as well as turning specific rules on or off as desired, equally in real time. Furthermore, Anderson et al significantly augmented the rule-set with signatures for the FTP protocol and several remote-user specific exploits. Aside from the two mentioned building blocks (statistical and rule-based analysis), NIDES features a *resolver*, an *archiver*, a *batch analysis*, and a *user interface* components. In a manner similar to IDES's, historical profiles are maintained, and constantly updated for all registered users. All its parameters are now customizable via the user interface, including the parameters affecting the profile's *aging* capability. Regarding the latter, NIDES enhances the profile update mechanism, which governs the IDS's ability to adaptively learn a user's behavior. In turn, this results in

[50] See Table 3, Section 2.1.7 and Table 5, Section 2.1.8.
[51] Introduced in Section 2.1.8 [44].
[52] Introduced in Section 2.1.7 [33].

accurate detection of intruders posing as legitimate users of the protected system. A similar, user customizable approach has been applied to the rule-based analysis component as well, such that the rule database can be augmented with signatures pertinent to the environment where NIDES runs. The role of the resolver is to act as a buffer between the alarms generated by the two analysis components and the user interface. Its role is to aggregate and filter redundant alarms, presenting the User with only the relevant alerts. A operationally similar component exist at the input side of the statistical and rule-based analysis block: the *arpool*. Its role is to gather the incoming audit data flow from the monitored hosts, and channel it towards the analysis block. Various other software engineering and performance enhancements distinguish NIDES from its predecessor. We retain its architectural acknowledgement that a rule-based component is best suited for the detection of known intrusion types and offensive traffic patterns, augmented by environment specific usage policy rules.

Yet another example is EMERALD, written by Porras et al [48]. Typical for hybrid IDS's in this category, EMERALD features both a statistical analysis and a misuse component. However, the key highlight of its implementation seems to be its distributed architecture. This is driven by the intended target environment for EMERALD: large corporate enterprises. Such environments are characterized by the presence of administrative domain conglomerates: truly independent, albeit interconnected network clouds providing various services to their registered users, while interacting in a peer-to-peer or hierarchical manner with each other. Among the difficulties in securing such environments Porras et al enumerate: inappropriate centralized storage of audit data, the need to use highly specialized signatures – which limit the scalability of the IDS via their express need to maintain state, and the resource exhaustion occurring at the host in charge of the centralized audit processing. These difficulties are all addressed via the hierarchically layered architecture of EMERALD. In particular, EMERALD implements analytical operations on three hierarchical layers: first, a *service analysis*, covering malicious activity within the boundaries of a single *domain*, then a *domain-wide analysis*, covering malicious activity across multiple components, and lastly, an *enterprise-wide analysis*, corroborating analysis activities across multiple domains. The first – service – layer has as its workhorse a *service monitor*: an analysis component capable of accepting

flexible inputs, such as simple audit log files, live traffic feeds, other service monitors, or input from additional, third party sensors (provided they observe EMERALD's grammar). Correspondingly, a *domain monitor* is responsible for performing analysis at the domain layer. They provide one extra level of abstraction, whereas *enterprise monitors* provide the highest level of analysis, aggregating information originating from all lower layer monitors. We note thus that the central processing unit of EMERALD is the monitor. Its architecture is generic: it features any number of components performing both misuse (*signature engine*) and anomaly (*profiler engine*) detection, and provides for runtime insertion or deletion of such components. The component insuring independence from the inhabited system's native audit data format is the *resource object*. It is constructed as a pluggable library, and it interfaces with the engines part of the same monitor, providing them with the required audit data for mining. The role of the two engines is to analyze the raw stream of input audit data, and generate a digest that is forwarded to EMERALD's *resolver*. The latter component is the "brain" of the IDS: it processes and correlates information flowing from any number of monitors it subscribe to, and has the ability to manage the subscribed engines. On the anomaly detection side, EMERALD further deliver upon its decentralizing goal by separating the user profile management from the theoretical algorithms used to infer the anomalous nature of events. Furthermore, a one-to-one map exists between engines and data sources; meaning, an engine is instantiated for every event stream to be processed, in an effort to improve efficiency. On the misuse analysis side, EMERALD enhance the classic (pattern matching) approach by postulating the need for state keeping, such as to accurately infer whether a given sequence of malicious actions result in transitioning the protected system into a compromised state. Moreover, since a large number of signatures is a known performance bottleneck, EMERALD further divides the misuse engines into specialized sub-components, each responsible for analyzing a particular data stream. Finally, a complete separation between the data analysis and decision making arms of the IDS is implemented via the resolver. These components are unique in the IDS architecture field in their ability to accept input from third party analyzers, and incorporate their input into their correlation efforts. We retain EMERALD's distributed architecture as a major

contribution to the field, and commend its efforts to provide for transparent integration into the protected system's topology.

Finally, we present JiNao [49], where Jou et al present a hybrid IDS focusing on the protection of the routers supporting the internet backbone. Though essentially host-based, JiNao also features a *remote management subsystem* where the conclusions inferred locally are aggregated so as to infer threats or attacks of a distributed nature, or who are hard to detect with a limited, local perspective – which it gains via its *local subsystem*. JiNao however is not to be confused with a simple host-protecting IDS: its role is to monitor neighboring OSPF[53] routing entities, and infer large scale routing infrastructure disruptions due to hostile network activities. JiNao accomplishes this via a two-layer traffic analysis procedure: a misuse based *prevention module*, and an anomaly based *detection module*. JiNao's sensors are actually stand alone modules – *interception modules* – directing incoming traffic to the prevention module. If so directed, interception modules can also temporarily hold packets when a remote correlation procedure requires it. The prevention module's role is to perform a quick, rule-based analysis of the incoming traffic, and drop any packets featuring clear protocol anomalies or security threats. The designers kept its signature database light, to avoid performance penalties. Traffic deemed "clean" by the prevention module is handed over to the detection module for statistical analysis, but unlike other IDS's in this category, it is simultaneously forwarded to the protected router's protocol execution engine. Again, it is clear that Jou et al favored router performance in their design, and implemented the ability to disable compromised routers should worse come to happen. The detection module features two components: a statistical analysis module and a protocol analysis one. The former functions in typical manner, comparing current traffic patterns with historical data to infer statistically-significant deviations. The latter module goes one step further by executing the traffic patterns in a sandboxed router engine, and observing potential occurrences of corrupted states. The decisions made by the two statistical analysis components are forwarded to JiNao's *decision module*, which correlates the data and determines if a compromised state has been reached by the monitored routers. In addition, the decision module forwards the analysis results to the remote management

[53] Open Shortest Path First routing algorithm. See [50] for details.

subsystem, where broader, global-view conclusions can be inferred, and rule enabling or disabling decisions are taken and forwarded back to the local subsystems. We retain JiNao's novel approach in protecting the actual backbone network – as opposed to only the end points, its capability to correlate analysis results from multiple local subsystems to infer global-scale attacks, and its implemented mini-protocol for determining if a given router has being compromised.

We close our mini-survey of prior work in intrusion detection by pointing our interested readers towards the following comprehensive surveys of IDS products: Stefan Axelsson's Technical Report [51], and the listings [52] for NIDS's, and [53] for HIDS's.

2.1.10. IDS Complementary Tools

The network security array of products is not limited however to the various flavors of Intrusion Detection Systems. Not substituting for them by any means, but rather complementing them, we distinguish: Vulnerability Analysis Systems (VAS's in what follows), File Integrity Checkers (FIC's in what follows), and Honey Pots (HP's in what follows) and Padded Cells (PC's in what follows).

Vulnerability Analysis Systems' main goal is, as their name implies, to assess the health state of a system from the perspective of known attacks. Their mission is thus not to detect when an attack occurs, but rather to assess if the system will be compromised should the attack happen. We do emphasize the fact that such a tool can only infer responses to *known* attacks; hence, any pretense of IDS-like capability is pointless. Nonetheless, it is a fact however that for small deployments, where the owners either cannot afford the costs of running an IDS, or where it is not possible to have an IDS deployed, a Vulnerability Analysis System provides significant input in maintaining a healthy state for the deployment in question. The input of a typical VAS is two-fold: the tool needs first the audit data to be analyzed – most often a set of core system files, and second, a database of attack *outcomes*. The latter detail further distinguishes the capability of a VAS versus that of an IDS: a VAS cannot detect the signs of in incoming attack; it can only detect the artifacts of a successful known one. Far from being a limitation, this capability renders any VAS into a highly effective forensic analysis tool.

As Bace and Mell describe in their report [14], VAS's also help SysAdmin's the thoroughly verify a system for configuration errors, compliance with local security policies, and successful deployment of system-wide patches. This is achieved via a set of successive snapshots of a core set of system attributes, which are then matched – at runtime – against a master baseline [14]. We consider the creation of the reference set to be a delicate step, as extreme care must be taken so as to insure no corrupted (i.e. already compromised) system files are allowed into the baseline. Bace et al actually point out that the reference set can be partially, or even entirely artificially constructed from an "ideal configuration template;" approach which will mitigate this risk in its entirety.

In a manner similar to IDS's, VAS's can be either *host* or *network* based – from a topology perspective, and operate in *credentialed* or *non-credentialed* mode – from an operating metaphor perspective [14]. When host-based, a VAS has direct access to protected system status files – hence it is also credentialed. In this configuration, it is very effective in detecting when a particular application (or user) attempts to modify system files – a typical case of *privilege escalation*. When network-based, a VAS mostly resembles an actual attacker. That is, in its most proactive implementation, the VAS will execute a known exploit against the protected system, and then assess its success. A similar effect can also be accomplished in a more passive manner: the VAS will launch non-threatening queries against the protected system, and will analyze the answers for evidence of successful (past) attacks. Since in both cases, the VAS does not have access to the system, this implementation enters under the non-credentialed heading [14].

We estimate the chief advantage of VAS's to be its ability to reliably spot a change in the protected system's state, and all depending upon the sophistication of the tool, specify the nature of the change. This ability can be used not only in read-only (i.e. scanning mode), but also in write-only mode: immediately after every major patch application (or significant composition or topology change), the tool can be run to fingerprint the system, and thus refresh the baseline set to be used as future reference. Nonetheless, VAS's do have disadvantages: for once, they can trigger a false alarm when enacting an exploit (in the network-based configuration) against a system already protected by an IDS. Even worse, repeated use of the VAS can ultimately train an anomaly-based IDS to accept VAS's hostile probing as part of the normal traffic [14].

Last, but definitely not least, network-based VAS's are as prone to false alarms as their IDS contra parts, whereas the host based ones must be, by their very nature, operating system specific – accessing system-state files is radically different on Windows-based systems versus Unix-based ones.

As an excellent example of Vulnerability Analysis System we cite the work by Ko et al [54], a host based approach. In particular, the VAS built by Ko et al proposes a close monitoring of any programs with a high security risk: though this is a rather all encompassing category, on a Unix clone, it encompasses programs which have read-access to system files, or which spawn shells during their execution. Furthermore, in the fingerprint-creation phase, the VAS also takes into account the *sequence* of operations a monitored program undertakes, such that an abnormal behavior can be readily detected. The VAS becomes thus more general in its operating metaphor, as it is not limited to detecting a specific attack pattern. The authors introduce thus the concepts of *trace* and *trace policy*, to define the normal execution path of a monitored program. Then, to specify the valid sequence of operations, the authors developed specific grammars whose alphabets are a given system most primitive operations, titled *parallel environment grammars* (PE-grammars). The end result is a VAS that can also detect attempts to exploit race conditions amongst privileged programs, or improper synchronization between distributed applications. This is accomplished via the system's leveraging a set of pre-defined heuristics. First, the authors enumerate *access to system objects*. That is, the series of objects (i.e. system files) a privileged program needs to access is always finite, and it can be easily enumerated (on a Unix system), which makes enforcing it achievable. Then, Ko et al substantiate *sequencing*. That is, in addition to the specific set of objects a privileged program can access, the order in which these objects are read or written to is, in several cases, pre-defined, and must be enforced. Third, the system takes into account *synchronization*; and the authors use the example of a potential concurrent modification – by regular user and system administrator – of the password file to exemplify this aspect, and the need to insure all accessed objects are left in consistent state. Last, the VAS Ko et al built has provisions to mitigate *race conditions*, since it is known that many exploits are targeting privileged programs suffering from this problem. From a topology perspective, the system is host-based, and runs in credentialed mode.

However, it is designed to run concurrently on several connected hosts, and has components that can be shared by them. One such is the common repository for holding the trace policies, or the white list of allowed operations for a given privileged program. At host level, the following components are required: an *analyzer* (the monitoring engine), a *trace dispatcher* (sends the merged log traces for all the instances of a given monitored program to the analyzers), and a *trace collector* (sends the low level audit data to the trace dispatcher). Ko et al implemented a prototype of their VAS on a Unix system, and successfully tested it for vulnerabilities in the *rdist, sendmail,* and *binmail* programs. Also tested was a simulated session whereas a concurrent modification of the password file was successfully detected. Finally, the authors argue that the VAS can be easily enhanced to monitor network components (or services) that constitute a preferred target for non-host-specific attacks, or, following a more extensive effort, the VAS could apply the same monitoring techniques to entire network subsystems.

Perhaps the most important contribution File Integrity Checkers bring to the network security community is their ability to accurately and reliably infer the footprint of a successful attack [14]. Their operating metaphor is rather simple; this however can be seen as an advantage, as it enforces their reliability. For any set of system files, the tool will compute – for each file – a fingerprint (often an MD5SUM[54], or another cryptographic checksum) which is then compared with a reference value stored on secure media; then, by constantly updating the set of changed files, the footprint of the attack emerges.

Undoubtedly, the best known tool in this category is Tripwire [56], created in 1992 by Eugene Spafford and Gene Kim at the Purdue University. In its original form[55], Tripwire was designed for the UNIX platform, and its main capability was reliable detection of additions, deletions, and modifications of any kind to any given set of files. Tripwire features four operating modes. The first, naturally, is an *initialization* mode. In this mode, the tool uses two separate inputs: the system's hard drive(s), and a configuration file where the local administrator specifies which files should be monitored – in a format affording various optimizations and preferences. The tool then proceeds and

[54] 128 bit cryptographic hash function; for details, see [55]
[55] as we specify next, Tripwire evolved in complexity, and became a commercial venture

creates a baseline fingerprint-database for the specified files. Tripwire's normal running mode is the *integrity checking* mode. In the first phase, the tool reads the configuration file, and generates a snapshot database. The latter is then compared with the baseline database, and after taking into consideration potential *ignore* options (for example files that are expected to grow in size, like various system logs), a report of the observed changes is presented to the SysAdmin. The remaining two modes are maintenance only: a *database update* mode, whereas the SysAdmin triggers an update of the baseline to capture main, or periodic changes, and a *interactive database* mode, which only differs from the previous by allowing the SysAdmin to veto each potential update. It is notable to consider the various optimizations Tripwire offers. For example, the configuration file can be shared among multiple hosts on the same system, thus heavily automating system-wide fingerprinting. Also, the tool supports several cryptographic mechanisms for computing the signature of a given file; this affords various degrees of cryptographic complexity to be matched with a given file's relevance, or site security policy.

Perhaps due to its immediate success and wide adoption, in 1997 Gene Kim transformed Tripwire into a private enterprise. Nonetheless, a similar tool is still available today to the Open Source community via the efforts of Samhain Labs, a German non-profit organization [57]. Samhain has the added ability to run in Client-Server configuration, whereas a central server collects and processes the results forwarded from individual hosts. This ability can also be extended in configuration mode, whereas the individual hosts access the server either to read the configuration file, or to write their respective baseline fingerprint-database. In a manner similar to Tripwire, Samhain requires an initialization phase to construct its baseline. It offers however the additional ability to store the fingerprint together with the file in question, mimicking watermark behavior (for example, it only appends extra bytes to a JPEG file). As opposed to the original implementation of Tripwire, which could only run at given time intervals (e.g. as a cron job[56]), Samhain runs as a daemon, and can interactively update its configuration (either configuration file of baseline database). Most importantly however, Samhain can be run in *stealth* mode, whereas it leaves no trace of its existence on the system disk. Furthermore, its inputs (configuration file and baseline database), as well as its outputs

[56] UNIX process scheduler; see [58] for more details

and Client-Server interactions are equally obfuscated. Though this requires tampering with the system's kernel, we consider this ability a great asset, considerably enhancing the tool's strength and reliability.

A Honey Pot predicates its operating metaphor upon a basic heuristics: any access to such a system is suspicious. This stems from their architecture: an HP is an imperfect replica of a real system, built as a decoy system and meant to attract traffic that cannot possibly come from a legal user. Its goals vary according to the intended application: it can be attached to a real system, and act as a magnet for would be attackers, or stand alone, meant to collect intruding traffic for later analysis [14]. In both cases, the HP attempts to credibly replicate a potential target system; its main task however is to collect as much data as possible about the incoming traffic, and in its most pro-active implementations, to respond to the attacker's probing with enough interesting data such that the intruder is lured away from the real target long enough for the SysAdmin to take appropriate defensive action. A Padded Cell tool works slightly different, but is similarly built [14]: as opposed to attracting an intruder on its own, it is used by the system's main IDS to transfers an incoming attack away from the protected hosts. Once the latter objective is achieved, the PC will likewise attempt to keep the attacker interested for as long as possible, while logging intruder's actions.

We were not able to find IDS's leveraging Padded Cells in their operating activities; numerous Honey Pot implementations exist however: the work by Vanderavero et al [59] is one such example. The authors perform their normal activities on an university campus, and seized the opportunity to leverage the magnet-factor[57] this location affords on one side, and the large number of available, un-assigned – and thus unused – campus-wide IP addresses on the other. The system the authors built – nicknamed *HoneyTank* – functions by replying to specific TCP segments send to a distinct subset of IP addresses characterized as above. Furthermore, the authors decided to implement a connection-stateless approach, so as to be able to handle the expected high number of connections characterizing the spreading phase of an Internet worm. This approach does have a main drawback, as the authors themselves point out [59]: since no

[57] We refer here at the fact that on any given campus, there exist a multitude of hosts in a confined (IP) address space, combined with the diversity of ongoing computing activities.

state is maintained for the received connections, the HP cannot fool a real human attacker into believing there is a real system at the receiving end. However, since the authors' main goal was to analyze automated attacks, the mentioned advantage of their stateless approach has distinct merit. As Varadero et al mention, a third party tool[58] can be utilized to analyze the collected data, and to actually generate signatures that a misuse-based IDS can utilize. For the purpose of their project, the authors utilized a version of ASAX[59] to analyze and respond to the incoming traffic. Then, the three-way TCP handshake was implemented in a manner implying a guaranteed answer for every incoming connection, and a simple response to an incoming Client request was equally provided for, insuring thus a basic emulation of an HTTP server. In like manner, the HoneyTank features a plain implementation of an SMTP server – mostly via regular expression parsing and processing. As the authors point out in the evaluation section, during a 5.5 hours evaluation period, HoneyTank received 3,433,579 TCP segments (1,702,632 containing data) from 12,859 distinct IP addresses. Not surprisingly, when sorted by port number, the top four spots were port numbers known to be leveraged by popular internet worms: 9898 – the Dabber worm and 2745 – the Beagle worm among them. Varadero et al also used Snort[60] to further analyze the accumulated data, and another no-surprise conclusion emerged: the top triggered rule was "SHELLCODE x86 NOOP", "[...] a generic rule which searches for a stream of 14 consecutive NOP instructions (0x90) in the payload of the packets and which indicates potential buffer overflows" [59]. The authors conclude by pointing out an obvious advantage of their stateless approach: the HoneyTank is bullet-proof against DoS attacks. The stateless approach to HP's is not without drawbacks however, and Varadero and Charlier proceed to extend the initial prototype in [61]. In particular, the authors implemented several versions of the TCP stack, in an effort to render HoneyTank more realistic, and thusly, more malware-friendly. However, the results were inconclusive, as their particular TCP/IP stack emulation was molded around *nmap*'s[61] behavior.

---

[58] Such as *Honycomb*; see [60] for details.
[59] We presented ASAX in Section 2.1.7 [35]
[60] We presented SNORT in Section 2.1.4 [18]
[61] Widely known network exploration utility [62].

Finally, we would like to mention in this category the contribution by King et al [63]. The project's goal is to precisely identify not only the set of changed files during an attack – like FIC's do, but more importantly, the *sequence of steps* that occurred during an attack, as the former does not usually imply the latter. Furthermore, BackTracker[62] can correctly identify the point of entry of the attack, and display the chain of events that occurred thereafter as a dependency graph. In particular, the tool functions by back tracking the set of activities and changed system objects starting from the known compromised system artifact[63] that alerted the SysAdmin (or the system administrator) to the intrusion. To achieve this, BackTracker constantly monitors specific system objects, and generates its own log data – required in the reconstruction phase. Namely, this is achieved at the boundary between the application and kernel levels, such that rich semantics can still be obtained while hardening the tool against shell-access disabling. BackTracker monitors three types of system objects: processes, files, and filenames [63], while keeping track of all related dependencies. We also note that the authors took extra care to provide for a pruning-mode of the resulting graphs (and related filters), such that the users can visualize an as simplified version as possible of the attack's inprint. The authors proceed to enact a series of attacks to evaluate BackTracker, and the resulting graphs are indeed outstanding in their clarity and fish-eye affordance of the attack's trace. This approach naturally creates the drawback of burdening the protected system's performance[64]; however, offsetting benefits emerge for specific cases where such monitoring is desired.

Finally, for a comprehensive survey of IDS complementary tools we direct our interested readers towards Talisker's listing [64].

2.1.11. Evaluation Of Intrusion Detection Systems

Naturally, one will wonder how well these engineering efforts achieve the goals of real life intrusion detection. Further, a potential buyer will certainly like to know how scalable a particular IDS solution is, what functional requirements assumptions are built

[62] The authors nicknamed the project BackTracker.
[63] The authors refer to this instance in space and time as *detection point.*
[64] The authors cite 9% CPU overhead and 1.2 GB of log data per day.

in for a given product, what intrusion area, if any, is a particular IDS better at detecting, or how was the false alarm rate computed – to name only a few criteria of essence at the IDS deploying end. Equally expected, a manufacturer's performance description is prone to be biased. It becomes thus essential to survey the evaluation approaches and results published by the network security community.

From a theoretical foundation for IDS evaluation perspective, we find solid guidelines for anomaly detection metaphor evaluation published as early as 1993. Indeed, in [65], Helman and Liepins attempt to define formal boundaries for the evaluation of anomaly-based IDS's by first defining a legitimate and a misuse processes, and then demonstrating that the upper bound for the IDS's accuracy is defined by a function of the difference between the densities of the two processes over the space of analyzed transactions. The authors also show how varying core parameters, like training data sample size and transaction (statistical) attributes influence the accuracy of the detection procedure. Most importantly perhaps, Helman et al demonstrate that *exactly* optimizing any of the anomaly-based IDS's criteria is NP-hard; thus, the employment of heuristics is likely to be validated as the norm for factory (or end user) tuning of anomaly detection systems' parameters.

From a more engineering-biased perspective, we outline the work by Puketza et al [66], where by adopting a common software testing technique, the authors developed a black-box testing framework destined to primarily evaluate misuse-based IDS's. To formalize the framework, the authors outline first a set of performance objectives against which an IDS should be hold accountable. First, a *broad detection range* is defined as the IDS's ability to detect an intrusion across a large set of attacking behaviors; then, an *economy in resource usage* is defined as the ability of the IDS to correctly function with as little as possible inference with the protected system's resources, and last, a *resilience to stress* is defined as the ability of the IDS to maintain an accurate detection rate despite high levels of local computing activity. Puketza et al also grant due consideration to the test case selection process, by defining intrusion partition classes which are either *intrusion technique* based, *host system vulnerabilities* based, or *intrusion signature* based. The authors proceed to develop a set of scripts exemplifying each partition class, and

employ them against a local implementation of the Network Security Monitor [7][65]. We retain as a significant the results obtained in the resilience to stress partition class, and where the IDS under test shows significant performance degradation (i.e. misses in attack detection) under high computing activity – likely in direct proportional relation.

We mention next two of the most significant IDS evaluation works of the 90's: the 1998 and 1999 DARPA Off-Line Intrusion Detection Evaluations ([67], [68]) performed at MIT's Lincoln Laboratory, and which can easily be qualified as mammoth in scope and complexity. However, given the popularity these two works enjoy, we will not present their findings in this paper, and refer our interested readers to the actual papers (they are published under the umbrella of several forums; our citations are by no means exclusive). Instead, we will focus in our presentation on McHugh's critique [69] of the two popular DARPA evaluations mentioned above.

As a case in point, McHugh begins by pointing out shortcomings in the manner in which the synthetic data used approximates real life network traffic; more specifically, as to whether the signal to noise ratio in the synthetic data corresponds to the same ratio in real systems' network data. In this respect, it appears that the DARPA evaluators failed to show that the false alarm rate exhibited by the systems under test is similar to the rate the evaluated systems showed on real data, and McHugh points out that the normal Internet traffic exhibits perturbations that are benign in nature, and yet not part of standard-complying payloads. Similarly, McHugh points out that the two studies provide little insight into the attack data distribution. This is obviously an issue, as regardless of which attack taxonomy one uses, it is not true that all attacks occur equally likely in real network data. This, combined with the operating systems and hosts mixture of the protected computer systems raises the question as to how representative was the selected attack mix for the targeted system. To give an extreme-scenario example, launching a Microsoft Windows attack against a host running a UNIX clone is not likely to cause much harm, or to be representative in an exhaustive IDS evaluation. Since the evaluation was supposed to target IDS's protecting an United States Air Force base computer system (and not say, an educational computer science laboratory, which is often composed exclusively of UNIX hosts), the point of accurate host representation, both in flavor and

[65] We introduced NSM in Section 2.1.1

number, is rather important. McHugh makes a further point in criticizing the attack taxonomy used by the evaluators, by mentioning that for what is listed as one atomic category, i.e. *denial of service*, a fair number of unrelated intrusions can be enumerated, and are likely to be detected as different by a given IDS. What is more, McHugh points out that the attacker-centric taxonomy used in the two studies leaves little room for a "normal" false positive or negative, where "normal" are such cases where the IDS did not flag a signal below its pre-set detection threshold, or a case where an alarm is not based upon detection of a unique event, but rather following transition to a subsequent state, based on latter events. Finally, McHugh dedicates a significant section of his critique to the methodology used in the two studies for results analysis. The main issue concerns the particular use of ROC[66] curves in the DARPA evaluations. To begin with, McHugh questions the departure from the typical ROC curve – "[...] a single operating point that expresses the percentage of true positives (the proportion of the actual attacks detected) plotted against the false positive percentage (the proportion of the units of analysis for which the system signaled an attack when none was present) for the entire evaluation" [69], by asking the IDS operators to return a result other than '0' or '1'. Or, as McHugh points out, this is unrealistic for many IDS's, as finding a matching pattern within a unit of data is a categorical event. Furthermore, McHugh opens the floor for open criticism of the decision to change the 'X' axis of the ROC curve from percentage of false alarms to number of false alarms per day. Although it seems that the evaluation authors did so following specific input from the DARPA sponsor, this creates the unwanted effect that "[...] using the dataset as provided for the evaluation, but reassigning values to the timestamps attached to the data items, the false alarm rate per unit time can be manipulated to any degree desired" [69]. It is further pointed out that given this, and should the evaluation test-bed be faced with megabit traffic-rates, the observed false alarm rate might have increased ten, or even a hundred fold. Though we have only outlined the main points of McHugh's critique, we hope we have exemplified the vast array of difficulties one is prone to encounter in evaluating IDS's, and that no one single method is likely to be the definite answer.

[66] Receiver Operating Curve [8], [9], [10], [11]; concept we presented in Section 2.1.2.

In the light of that conclusion, we propose next a brief survey of Axelsson's base-rate fallacy paper [70]. As its title implies, the cornerstone of this particular paper revolves around the Bayesian statistics concept of *base-rate fallacy*. When applied to the Intrusion Detection problem under a specific – though certainly realistic – set of assumptions, the author arrives at the surprising conclusion that the main performance limiting factor of an IDS is the unexpectedly low *false alarm* rate it has to afford. In particular, Axelsson starts from the assumption that the average modern-day computer system is subjected daily to high volumes of network traffic, of which only a small percentage constitute actual intrusions. Since most IDS rely on logging and analyzing an overwhelming percentage of system activity to afford detection of intrusive behavior detection, the false alarm rate can be computed based on this set of assumptions. We leave to our readers the perusal of the actual equations, and we retain for our analysis the conclusion: a desirable *detection rate* "is completely dominated by the factor (0.99998) governing the *false alarm* rate [...]" [70]. These results are further emphasized via a plot featuring the Bayesian detection rate against the false alarm rate. Between the two perfect extremes (100% detection and 0% false alarms), one can easily note that a highly unreasonable false alarm rate (1×10^{-5}) is required for a quite reasonable Bayesian detection rate of 66%. In the latter section of his paper, the author proceeds to plot the ROC curves of several published evaluations[67], and judge them in the light of its prior reasoning. Axelsson's ROC curves are benefiting not only from conforming axes definitions[68], but also from the author's initiative to draw an "ideal," base-line curve based on his starting point assumptions. Also, a mix of both misuse and anomaly based IDS's are plotted on the same graph for comparison. As the author concludes, several factors influence the performance of various IDS's, the nature and composition of training data, and its influence on performance being the most poignant. We retain Axelsson's work as a highly precise question mark raised against the choice of deciding factor in evaluating IDS performance.

Next, we introduce an equally exact approach in IDS evaluation; this time, the authors employed a benchmarking metaphor. In their paper [71], targeting the anomaly

[67] Some of the IDS's tested in the two DARPA evaluations previously mentioned included
[68] See our prior presentation of the critiques addressing DARPA evaluations' ROC analysis

detection IDS metaphor, Maxion and Tan start with the following two hypotheses: first, variances in data regularity will influence IDS performance, and second, such variances occur in real-life network data. Further, their approach is motivated by the just observation that the signal to noise ration occurring in the to-be-analyzed data will undoubtedly vary according to a specific deployment environment (i.e. educational, medical, military, ecommerce, etc). To test the first hypothesis, the authors construct several synthetic data sets, as required by systems in this category – namely, training and test data sets, plus the attack (anomalous) data, and measure changes in the data's regularity via entropy[69]. The approach in data sets generation semantics was straight forward: a subset of the Latin alphabet as normal data, and a pre-defined set of anomalies was also crafted; we refer our readers to the published work for additional details concerning synthetic data sets construction. We retain two main observations regarding the observed results: first, despite the fact that 165 data sets were constructed, using 5 alphabets and 3 types of anomalies, the resulting ROC curves were highly similar. Second, the results confirmed the starting point hypothesis: variance in the analyzed data's regularity translated into variance in IDS performance. Furthermore, even for the case of perfect (i.e. 100%) detection rate, it was observed that as randomness in input data increases so does the much dreaded false alarm rate. Maxim and Kim's work also provides confirmation for the second hypothesis; however, we retain their work with respect to the effect in analyzed data's randomness as significant, if only for its implications in real life IDS deployment: the assumption that a particular IDS will perform similarly across a wide range of environments seems to be invalid, and this even if and adequate training period is observed.

Last, we emphasize the work by Ulvila and Gaffney [73], as they seem to provide an answer to many of the previously noted shortcomings. In particular, the authors' answer to the problem of how does one select the best IDS to fit a particular deployment situation is a decision cost tree. Further, their approach can be extended to a apples-to-apples comparison of two IDS's, and also to showing how one might re-tune an IDS following a major operating environment change. Ulvila and Gaffney's main idea gravitates around the concept of operating costs. In particular, after defining the

[69] Measure of randomness; for more details, please see [72]

probability of an IDS not raising an alarm given an intrusion as β, and the probability of the IDS raising an alarm when no intrusion was present as α, and define *operating point* as the point on a given IDS ROC curve having these two probabilities as coordinates. Further, the authors define respective costs associated with each detection error. Rather than using absolute cost terms, the paper focuses on real environments consequences of operating an IDS, such as the cost for an SysAdmin to investigate an alarm, the cost of having the host system down for recovery reasons, etc. The end result is presented in the form of horizontally laid out decision tree, whose nodes corresponds either to actions under human control, such as responding or not to an alarm, or to uncertain outcomes such as the various probabilities involved. By additionally taking into account specific values of the probability of intrusion for a given milieu, and assuming the cost of failing to respond to a real intrusion to be five hundred times greater than the cost of responding to a false alarm, Ulvila et al arrive at very precise cost analysis results, allowing a potential IDS buyer to arrive at a operating cost-based purchasing decision, provided the probability of intrusion is known[70]. From the works we surveyed in this section[71], it also seems that Ulvila et al are the only research group to arrive at the conclusion that two identical IDS's are almost always better than just one. The "almost always" qualifier is guided via the sliding window afforded by the probability of intrusion: at the two extremes we have the obvious cases where one should respond to all alarms or to none at all, whereas there exist a sweet spot in between where it is better to answer to an alarm raised by either IDS, or to both. We retain as this work's main contribution the cost analysis allowing one to steer away from the much debated[72] ROC curve, or the inherently subjective[73] rate of false alarms, and instead use them only as an aid for basing an evaluation decision strictly on the operating costs an IDS or other will entail – the authors actually provide a six-step procedure for independent IDS evaluation as part of their conclusion.

---

[70] We infer this owe to be the case for most commercial users.
[71] The two papers ([74] and [75]) by Antonatos et al are not presented here, but we encourage our readers not to overlook them.
[72] See McHugh's critique in [69].
[73] According to the vast majority of works surveyed, the deployment environment influences the observed IDS false alarm rate.

2.1.12. The Need for Correlation

We can conclude, following our IDS technology presentation, that there seem to be an obvious dichotomy present in IDS technology. On the one hand, we have IDS's highly specialized in monitoring hosts, but are completely blind to network events – and vice-versa. On the other hand, misuse based systems tend to be highly reliable and provide few false positives, if any[74]; however, they constantly require signature updates and are blind to attacks for which a signature is not already present. Anomaly based system by comparison, have none of the signature-associated drawbacks, but they usually trigger a significant number of false alarms. It comes thus as no surprise to hear that most enterprise level networks feature a combination of IDS's to monitor and defend their assets. With this added security however comes the extra complexity of managing more than one IDS. And, if the respective IDS's are not provided by the same vendor, then the chances of inter-IDS collaboration are slim. This places users in the awkward position to have to hire and train several SysAdmin's to deal with each IDS independently and have them corroborate their findings every time an alarm is raised, as one potential solution to mitigate for the varying IDS's locally deployed. Naturally, other solutions can be found; the quintessential problem here lies with the fact that the onus is placed on the User to find a way to cope with heterogeneous proprietary technology, as opposed to having the manufacturers develop one.

The network security research community was listening however, and in recent years[75] several works have being presented providing much needed solutions to manage several locally deployed IDS's and correlate their results in a cohesive final report and intrusion analysis.

One such example is the work by Valdes and Skinner [76], where the system the authors built relies heavily on the distributed architecture advantages EMERALD[76] affords. In fact, not only is the EMRALD hardware leveraged here, but also its template for alert data and management. The other two IDS's utilized are ISS RealSecure[77] and a

[74] When properly configured, of course.
[75] We mean that statement as compared to the years when the first major research IDS's were born.
[76] Presented in Section 2.1.9; see [48] for details.
[77] Presented in Section 2.1.6; see [30] for details.

Checkpoint Firewall[78]. The authors' main contribution is substantiated in the correlating algorithm: a probabilistic method for combining separate sensor alerts into cohesive "threads," thus greatly enhancing the ability of the SysAdmin to understand any particular attack, while reducing information overload at the same time. In particular, the authors have designed a top-down, attack class-based hierarchy, and analyze – at run time – the incoming sensor data in order to infer potential for merger into already instantiated such attack classes. Significant development was undertaken at the level of metadata used for grouping. For example, this metadata can be part of either the attacker's, or victim's set of identifying parameters (port numbers, IP addresses, etc), and a potential merger is also looked at from an attack state perspective. That is, the correlation algorithm attempts to infer whether the newly reported data is the next stage of an in-progress attack. Not surprisingly, the experimental results are encouraging. For a real-traffic, three-week-long scenario, "[...] the IDS sensor[79] processed over 200,000 sessions and generated 4,439 alerts. The probabilistic correlation system produced 604 meta alerts" [76]. Furthermore, Valdes et al organized a closed experiment with controlled network data, where their system's capability to merge related threads was further proven; this iteration showed especially the ability to reconstruct a distributed attack scenario, by adding any compromised local hosts to the attacker's assets, and we consider these results encouraging, especially since they do not require modifications to the managed IDS's.

A similar approach to IDS correlation is taken by Goldman et al [77]; the devil is in the details, however. Where the system Valdes et al built is a model of simplicity and ease of use, integration, and deployment, the solution proposed by Goldman et al appears to be a mammoth in complexity. Further, the results shown in the paper do not seem to justify the expense a potential user will have to incur to install SCYLLARUS (the correlation application's name) on their network. To begin with, a component of SCYLLARUS is a *security goals database*, which is particular to a given site's security objectives. Though the paper provides examples as to how entries in this database are defined, we have not seen a reference to pre-installed templates, and this seems to indicate that a potential user will have to first infer all the potential threats to her

[78] N.B. This product is a firewall, not an IDS.
[79] Meaning, EMERALD's

infrastructure, and enter them in this database using SCYLLARUS's terminology. The authors justify this by speculating that current IDS's are a WYSIWYG solution, not discriminating between the security needs of an e-commerce site versus an air force base, and provide the example of a web server: critical for the former user category, a "PR luxury" [77] for the latter. Though we do not contest the value of the argument, we can easily turn it around: what are the taxpayers of a given country going to think about their military, if a multi-million (or should we say billion?) dollar air force base cannot protect as much as its own web server? That argument aside though, we believe that burdening a system administrator and the local SysAdmin with the task of populating the security goals database from scratch owe to be avoided. Further reading clearly shows SCYLLARUS operates as a complex software layer with a master-slave architecture[80] of its own, and intercepting and aggregating the signals from the locally deployed IDS sensors. Though this is certainly nothing new, nor flawed, we fail to see the advantages it brings to justify its complexity. Without delving into the architecture itself, we will again focus on the results. The paper features (on page 5) a figure showing "how the goal of maintaining user account authentication has been violated" [77]. It is composed exclusively, aside from two ovals featuring the names of the violated security goals, of squares featuring letters and digits with no immediate meaning; they are either a "report" or an "event", connected by arrows. The section explaining the figure concludes with "the interested user can "drill down" to find the details [...]" [77], and two tables are provide showing the kind of available details. We fail to see how, after going trough the trouble of installing SCYLLARUS, is the SysAdmin any better off than by simply performing the same investigating work on the reports form the IDS's signaling the security breach in the first place. We further counterbalance SCYLLARUS's complexity against its advantages via a provided example. To exemplify their system's ability to allow multiple inheritance in the reported attacks, the authors provide the example of an exact signature match forwarded by a USTAT[81] sensor for a well defined (and known) Solaris exploit[82], and the same exploit as less accurately reported by a local anomaly based IDS. We again turn the authors' example around, and ask ourselves: how many

[80] The published work [77] provides ample detail as to the architecture and its components.
[81] We introduced USTAT in Section 2.1.7; see [36] for details.
[82] The exploit in question is *sadmindex*.

SysAdmin's out there would ignore a USTAT signature match for a known exploit, after going through the trouble of installing this IDS in the first place? We think the answer would be 'many' only if the protected network will lack any Solaris hosts. We fail to see the need for further processing, and believe this is a poorly chosen example, as signature-based IDS's are known for their virtual lack of false positives, and because the SysAdmin can tune the IDS with respect to which signatures should be strictly enforced and which not, and because further processing of this particular alarm signal cannot possibly yield more than just confirmation of this particular alarm. Further, assuming USTAT gave a false match in this particular case, Goldman et al fail, in our opinion, to make the case as to the added value SCYLLARUS brings in this particular case. As a last concluding point on the usability of SCYLLARUS, its *event dictionary* – one of its components – requires the supervised IDS's to utilized a specific vocabulary for communication. As the authors themselves admit, this also requires vocabulary adoption by the IDS developers, thus placing SCYLLARUS in the potential-future-adoption category, provided all stake holders agree to support it – clearly not a strong incentive for end user adoption.

Debar and Wespi on the other hand leverage an existing[83] (commercially implemented) event handling console for information systems [78]. This approach has the main advantage of allowing the authors to concentrate on the actual correlation algorithm, while at the same time providing corporate end users with a familiar interface. It is to be mentioned that the Aggregation and Correlation Component (ACC) Debar et al built targets large enterprise deployments (hence the assumption that the TEC will be present), where several IDS's complement each other in the intelligence gathering and alarm raising arenas. To further deal with the level of network complexity at a specific site, the ACC affords a master-slave relationship, where higher level ACC's receive and correlate reports from lower level ones. In particular, the ACC's main task is to attempt, for every received alert, a correlation analysis pass. Furthermore, the system performs a hierarchical grouping of alert views, such that the independent alerts are prevented from cluttering the information display, and in so doing, permits monitoring of an attack as a whole, even when spread over various lengths of time. This is achieved via a key design requirement: always present a single alert per attack, regardless of how many alerts a

[83] Tivoli Enterprise Console (TEC); please see the references in the actual paper for more details.

given attack has generated at the sensor level. On the other hand, false positives are avoided via intensive use of alert correlation and aggregation. On the correlation front, the ACC distinguishes two possibilities: *duplicate alerts* and *consequence alerts*, and infers heuristics for detecting them. For example, in the duplicate case, metadata such as timestamps, and source and destination IP addresses (and/or hostnames) are used. What is more, the system has the ability to generate new signatures based on the number of times a specific alert is repeated: once the number of occurrences reaches a pre-defined threshold +1, the chain of alerts is recorded as a new attack signature. To infer a consequence alert, the ACC uses first a pre-defined link, and second, a timer. That is, within a specified window of time, a predefined link between alerts *a* and *b* must occur. In addition, the ACC infers three aggregation axes (source, target, and class of attack), which are used to define *aggregation situations*. We note one interesting use of the latter concept: detecting authorized scans, and inferring whether the scan pattern changed, or if any of the system's sensors is malfunctioning. We infer as main critique of the ACC its unnecessary downplay of the classic IDS technology on one hand, and it reliance on the ACC operator to assign confidence levels for a given alert's accuracy. For example, Debar et al claim that "When deploying a large number of probes, operators are forced to reduce the number of vulnerabilities monitored to limit the number of alerts to a reasonable level" [78]. However, the authors also state that "This [sensor alert confidence value] is determined independently of the product vendor. Default values take into account the intrinsic inaccuracy of the alert and should be modified by the operator [...]" [78]. First, no explanation is given as to how the default confidence value is computed, and second, we infer as a contradiction to ask – most likely – the same "irresponsible" operator who would allegedly reduce the number of vulnerabilities monitored to now modify the confidence value of a given alert to better reflect local deployment behavior. We conclude by underlying the significant inroads Debar et al make in the area of IDS alert aggregation and correlation by formalizing an Alert Class Hierarchy and a hierarchical approach to alert aggregation, and their successful system integration with the Tivoli Enterprise Console.

At the end of this section, we conclude that Intrusion Detection Systems are a mature and modern technology which in spite of its shortcomings, has proven its

necessity and utility in today's heavily networked computing deployments, and is especially relevant given the increasingly ubiquitous nature of LAN-WWW integration.

2.2. Mobile Agent Systems

2.2.1. Introduction

A brief historical survey allows us to conjecture the following hierarchy within the evolution of machine data communication:

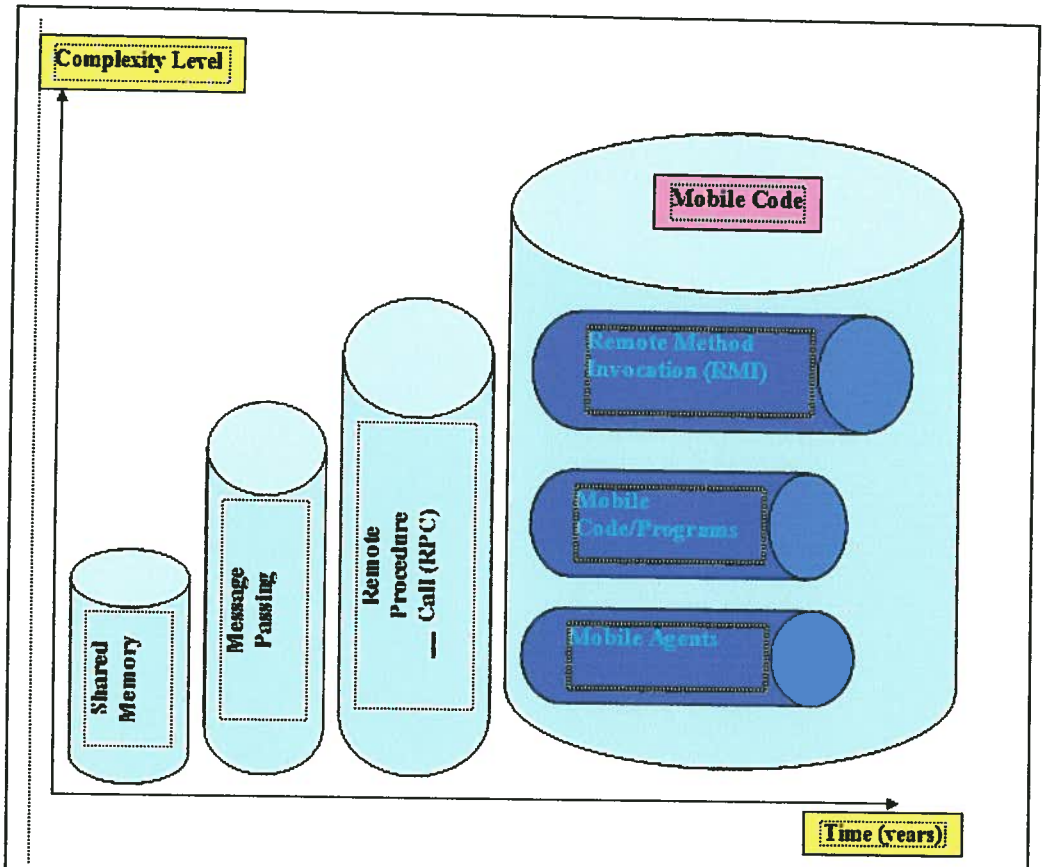


Figure 1: Inferred Hierarchy within the Evolution of Machine Data Communication

Form a high level perspective, we can infer that the need for (mobile) software agents arises from the necessity to provide an alternate mechanism for interaction, information communication, and data processing between networked nodes, without the assistance of a human, and especially in cases where the target nodes are connected via non-permanent connections. Notwithstanding, we would like to point out that it is a known fact that almost any task benefitting from the use of mobile agents can be accomplished by one of the less complex mechanisms.

However, in certain particular cases, the usage of mobile agents has offsetting benefits over its disadvantages. One such case occurs in network security applications, where analysis-triggering events occur in real time, and must be carried out at unpredictable data-store locations across the protected network. And once the main mobile agent platform security-related drawback is addressed (i.e. the trust level of the target hosts), then a (network security) solution can significantly benefit from preferring the latter method as its operating metaphor.

As we have chosen to implement our VoIP IDS on a mobile agent platform, we will introduce next the core concepts behind the mobile software agent paradigm, we then present several successful instantiations, as well as the main advantages and disadvantages of the metaphor.

2.2.2. Mobile Agents Conceptual Model

The semantics of the *mobile agent* term is two-fold: it stems from *mobile code* and *software agent*. The software agent term precedes the former, and is widely known in the computer science community: it straddles the research areas of Distributed Problem Solving and Parallel Artificial Intelligence [81]. Given this, various definitions of the term exist, all depending upon the respective application domain or research area. We can however safely infer that at its most basic root, a software agent is a module performing a computing task on behalf of another entity – human or artificial. From here, various partition classes can be inferred: for example, we can classify software agents based on the nature of the task they are designed to execute, based on whether they collaborate

with other agents or not, based on their inner implementation, based on their learning ability or lack thereof, as well as based on their physical location – to name only a few. The mobile code concept has a more mundane definition, as it merely designates code that can be executed on a remote host (i.e. not the "home-base" host of the module) without requiring any explicit environment accommodations; common scripts[84] are an obvious example. Naturally, the arrival of the mobile code component at the target host needs to be triggered first via an external event; however, once that is accomplished, no further action is required for the module's execution. Therein also lays its main interaction paradigm: Client - Server applications, where a Client node requests a service from a Server node. Fuggetta et al [82] have been instrumental in establishing the foundation for formalizing mobile code motivation, classification, and basic conceptual building blocks. From a classification perspective for example, they distinguish three main dimensions: *technologies*, *design paradigms*, and *application domains*.

2.2.2.1 Technologies

Technologies represent the mechanisms enabling the mobile code paradigm. In particular, Fuggetta et al distinguish two main components: the *Computational Environments* and the *Executing Units*. The Computational Environments' role, as their name implies, is to insure "visiting" applications find a suitable environment on a foreign host, thus enabling dynamic component relocation – one of the basic needs for mobile code execution. Executing Units on the other hand, "represent sequential flows of computation" [82] (e.g. a single-threaded process). These two building blocks are not unique to mobile code; on the contrary, they are present in all environments. The binding between the two however is what distinguishes the mobile code paradigm, in the sense that the EU can be relocated to another CE, whereas this is impossible in the classic model. Furthermore, a distinction must be made between strong mobility – where not only the code, but also the execution state can migrate – and weak mobility, where only the code can migrate. It is to be noted here that in the case of strong mobility, the additional task of re-enacting the originating resource allocation and data bindings must be performed.

[84] JavaScript, VBScript, etc.

2.2.2.2 Design Paradigms

Not surprising, the mobile code design paradigms are radically different from the classic EU-CE model, as they must provide for the fractured binding between the two. In particular, Fuggetta et al distinguish *remote evaluation*, *code on demand*, and *mobile agents*. In the remote evaluation case, the resources required for a specific computational task are not co-located with the executing code branch. As such, the latter must be deployed at the node holding the resources. In the code-on-demand paradigm, the requestor lacks the intelligence to perform a task for which it already holds the resources; as such, it must receive the know how as a visiting module. Lastly, in the mobile agent paradigm, it is no longer the question of relocating just the executing code, but the entire executing component, and its state, resources, counters and stack composition, in a manner which will afford execution restart at the target node.

Two main implementation technologies are employed in the mobile agent paradigm: a "weak" implementation, where the execution state is not preserved upon migration, and a "strong" one, performing the opposite. In the former case, additional, explicit coding steps must be undertaken to appropriately package the current state before migration, send it all following the adequate communication protocol, and un-package it at destination. In this case, the application creator must have knowledge of the protocols involved, and explicitly satisfy their criteria. Strong mobile technologies are characterized by the use of a "platform" encapsulating all the required protocols and state preservation operations needed for the module's migration, thus freeing the application creator to concentrate on solely solving the problem at hand.

2.2.2.3 Application Domains

A typical application for mobile agent technology is information retrieval. This is especially attractive when significant data stores located on nodes with limited to non-existing computation capability are at involved, as the alternative of shipping the data to the computational node would be prohibitive. Yet another common use arises in networks where the connection between several nodes is not reliable. In such a case, a classic, Client-Server interaction will be unfeasible; however, it will suit well the mobile agent model as network connectivity is required only for the agent and results shipping phases of the computation. Fuggetta et al further identify a niche application for the support and

management of advanced telecommunication services, especially those where the human actors are mobile users. Such users are bound to disappear from location A only to reappear at location B after a given time interval. Perhaps the most popular application domain for mobile agents is the field of *e*Commerce; two main areas of specific interest are travel booking activities as well as stock trading.

Conceptually, the mobile agent paradigm establishes itself as a solid alternative to the Client-Server model, via its ability to address the core drawbacks of the classic model. It is to be noted however that the mobile agent paradigm has its own drawbacks, as we detail in Section 2.2.5, and also a cost. Our work aims to exemplify one instance where the cost-benefit analysis favors the mobile agent model.

2.2.3. Mobile Agents Communication

2.2.3.1 Background

Arguably, the earliest attempts to define an agent communication model surfaced in the Life and Social Sciences field, where computer simulations of large populations proved to be a viable alternative to natural population study. Naturally, other areas like Biology, Molecular Chemistry, and most recently Economics have found the ability to generate large scale populations for equilibrium and evolution study an invaluable tool in their research arsenal. In so doing, the need to model a communication system closely replicating the one observed in the population of interest triggered several implementation approaches, which had to also comply with all dimensions of the studied population. Meaning, as Schweitzer et al explain [83], a multi agent system is composed not only from a significant number of members, but also from agents of different types. Yet another level of complexity presents itself once we take into account realistic network communication issues like latency, disconnections, congestion, etc. Finally, as the individual agents become more complex – so as to better reflect the natural contra-part it is supposed to mimic – the state space required by the agents' communication rules increases exponentially [83].

In an effort to mitigate this series of agent communication problems, several approaches have been inferred.

2.2.3.2 Proposed Models

Schweitzer et al for example propose a minimalistic approach to agent design, where information is divided amongst *functional*, *structural*, and *pragmatic*. The functional information strictly regards the agent's capability to process information from outside the system; structural information regards exclusively the syntactical aspects of data, and finally pragmatic information is the result of the agent's application of functional information onto a specific token of structural information; it is the type of information enabling agents to *act* in their environment. Additionally, and faithful to the minimalistic design paradigm, Schweitzer et al also introduce the concept of *blackboard* as communication medium, external to the agent itself. That is, blackboards act as storage at all levels of agent communication – local or remote, with open or closed access, monolithic or distributed across several sites. They also follow the publisher-subscriber metaphor, whereas information from a master table trickles down to slave ones. The entire agent ecosystem functions thus based on each agent's innate ability to process specific kinds of information. This information is stored on an elaborate network of black boards, and the processed information becomes input for other specialized agents, with byproduct information generated on the fly and equally stored on the blackboard network.

Yet another high-level approach to agent communication is proposed by Parunak et al [84]: use the pheromone model for inter-agent communication. In particular, Parunak et al note that ants do not follow an entropy trail deterministically, but their movements are influenced by the strength of the pheromone trail. As either of the nest-guiding or food-guiding pheromone trails are deposited by other ants, and the pheromone itself requires a (solid) surface for adherence, we can see a parallel with the blackboard approach suggested by Schweitzer et al. Applied to mobile agents' behavior, the pheromone model suggested by Parunak et al functions via an "advertising" activity. For example, if one agent wishes to "attract" another, it will continuously emit pheromone molecules, thus increasing their density at its location until their scent is sufficiently dispersed to be caught by an interested mobile party.

Stepping now firmly onto Computer Science field, the work by Finin et al [85] stands out as a major pioneering effort in the area of mobile agent communication:

KQML, or the Knowledge Query and Manipulation Language. KQML provides both specialized communication constructs to be used by agents, as well as provisions for the instantiation of highly specialized agents, whose sole role is to facilitate communication between the "worker" agents. In so doing, Finin et al also address the heterogeneity concern, whereas agent of different making, various hosting platforms, and non-uniform data sources all need to interact and co-operate. In particular, Finin et al start by pointing out the fact that in a significant number of Client-Server agent inquiries, the Server side is often just a facilitator, collecting the answers requested by the Client side from other parties. The authors expand on the middle-man concept by proposing the instantiation of highly specialized agents called *communication facilitators* to handle all incoming requests from worker agents. Furthermore, their role is expanded to providing directory and registry services, content-based message routing and forwarding, and "matchmaking" between interested parties. For example, some facilitator agents can specialize solely in becoming look-up directories for peer-agent finding. As for the vocabulary itself, specific KQML language constructs named *performatives*, are used for worker-facilitator communication. Other than that, KQML does not impose any particular message content requirements. Meaning, the payload can be formulated in whatever language or terms the communicating parties are designed to utilize; KQML constructs are only a wrapper allowing the message itself to pass through and to find communication participants. Given this model, KQML has the significant advantage of not imposing any particular operating system of platform for its functioning. Cross application communication for example, is materialized via KQML-speaking *router* agents, whose role is to pass messages between applications, even those who otherwise are not designed to communicate, and we believe therein resides its main strength.

A second major approach to inter-agent communication is the OMG MASIF specification [86]; its main contributions are the fact that MASIF addresses communication between agent systems, and not between agent applications, and its emphasis on agent (communication) security. Also, in addition to providing specifications for agent transfer and management, MASIF also provides a vehicle for agent naming and locating; thus, perhaps its most important contribution resides in its ability to provide interoperability between distinct agent systems. To accomplish its

goals, MASIF introduces a series of concepts: *places* are the local contexts in which individual agents "live"; *regions* are sets of agents of the same authority; *agent serialization* is the process of flushing and agent to a hibernation state from which it can be easily re-activated; equally, the concepts of *agent transfer*, *remote agent creation*, and *remote method invocation* are defined. These concepts are also instrumental in the naming convention MASIF introduces, and which plays a major role in agent finding operations. To that respect, MASIF uses OMG as its naming authority, thus greatly enhancing its interoperability. Like already mentioned, what impressed most about MASIF is its holistic approach to security. Meaning, the goals of Milojicic et al is to secure not only the agents themselves, but also the resources of the hosts where they run and access to all other resident programs. This is achieved primarily via secure authentication both at the system, as well as at agent level. Then, all requests for remote activity, such as agent creation and resource access via agents must also be authenticated.

The work by Baumann et al [87] attempts to independently address two main agent communication hurdles: inter-agent communication between previously unknown parties, and anonymous communication between interested parties. To accomplish the former, Baumann et al first identify the need for globally unique agent identifiers, to be generated by the system at agent creation time, independent of the agent's place of birth. Secondly, the model specifies a specialized agent type, the *service agents*, which are stationary, and provide an interface to the local resources for the requesting agents. As such, agent communication is now distinct: peer-to-peer, i.e. between mobile agents, and mobile agent - service agent; the resulting concept of group communication is the building block for anonymous agent communication Baumann et al propose. The latter type of communication is actually more common than its name implies; for example, an agent (or group of agents) are interested in a specific activity and have extensive knowledge regarding it, but do not know how to identify other agents whose labor is required in performing the activity. For this particular case, communication will follow a *session* model, and agents will additionally be identified via *badges:* an application generated ID which agents can advertise or hide as dictated by local circumstances. Also, the required "glue" for communication establishment is provided via *synchronization objects*, whose role is to handle the interactions between a given agent and the local

application it wishes to utilize. In particular, synchronization objects hold state with respect to the outcome of a given agent, or group of agents' activities, and can generate output events to be broadcasted to interested agent subscribers. The authors work has been concretized into a stand-alone mobile agent platform titled "Mole" [91].

We conclude this section by underlying the fact that given the expectation that mobile agents behave like independent, fully functional computational units, the fundamentals of complex communication between intelligent entities must be addressed for the technology to survive.

2.2.4. Mobile Agent Platforms

The communication models exposed above have given rise to a series of approaches to implementing full fledged agent platforms, whose goal is to free the application programmer from the burden of micro-managing the fine details of agent communication, migration, and data access, and allowing her to concentrate on solving the problem of interest instead. In what follows, we detail several major mobile agent platforms.

2.2.4.1 Telescript

Telescript [88] is a full fledged mobile agent platform, intended to work in close dependency with the World Wide Web. Its main implementation concepts are *agent*, *place*, the *telescript engine*, and the *telescript language*. Places are either home, or else landing terminals for agents travelling via the WWW. Each place is hosted by a physical node, and aside for the "home" ones, are virtual shopping malls. The basics of agent identification and place finding are handled via special system primitives, like *telename* and *teleaddress*, where the telename denotes the agent's "master", and the teleaddress the coordinates of a place. In addition, various other electronic tokens are used to represent an agent's "mission", or a request for a meeting with another agent. Communication security is insured via a system of *magic cookies*, which are randomly generated strings similar to the ones used in XWindow applications to prevent unauthorized display use [89]; also, agents are prevented from tampering with the mission-specific tokens they carry. The language is extensive, and provides adequate constructs for nested place

complexity and a system of abstract classes permitting a comprehensive place-agent-engine integration.

From a WWW usage perspective, Telescript affords dynamic page updates based on information retrieved by the agents using input collected from the request page. In addition, Telescript brings the added advantage of enabling the storage of partial web queries at place level, while waiting for the information of interest to be made available at a remote location. We retain as its core strengths Telescript's easy integration with Java, and the simplifying of HTML document parsing on the programmer's side via localized, agent goal-specific parsing.

2.2.4.2 Agent TCL

Agent TCL [90] proposes to address a few shortcomings of earlier platforms, namely Telescript. In particular, its main focus is to be operable on all flavors of UNIX, and to support several languages and agent transport mechanisms, while preserving the agent-centric view to supported operations (e.g. allow single instruction migration operations). To achieve this, Agent TCL proposes a four layer architecture, as follows. The transport protocols are located at the lowest level, and provisions are made to allow mechanisms beyond TCP/IP, like simple email, for example. Immediately above, there exist a *server* component, much like Telescrpt's engine. Its role is to performed all house-keeping tasks related to the agents' state and innate behavior: keep (visiting) agent status, accept incoming visitors, handle inter-agent communication, provide name space and directory services. Further up we find the *interpreter* layer, where language interpreters for every supported agent making are located; Agent TCL assumes only interpreted languages are viable for mobile agent implementation due to security concerns. In addition, interpreter modules have additional components for agent state capturing and also for security against malicious code execution. Finally, the topmost layer is occupied by the agents themselves, interacting with the server via their corresponding interpreter.

Agent TCL's key merits are its open-platform operating metaphor, and its proven applicability in the remote data-mining arena [90].

2.2.4.3 Aglets

The Aglets approach [92], [93] is based on the Java Aglet API,[85] and capitalizes on the Java platform's established mechanisms for code execution regardless of computing environment. Aglets significantly differs from the two platforms presented above in the sense that it implements *weak* migration, as opposed to *strong* one. As a reminder, the main difference between the two is that in the case of strong migration, the agent's state is also transferred, thus preserving execution state (i.e. execution can restart *automatically* from the same point after migration). Aglets thus has the advantage of allowing for a significantly trimmed data transfer for agent migration operations, at the cost of having to specify a "run" or "start" method which is to be executed immediately after migration. Lange et al denominate "aglet" a mobile agent, and further follow the established operating metaphor: a "context" is the agent's place, and similarly, "itineraries" and "identifiers", plus agent migration and communication tokens are defined. Standing out is the concept of "proxy," which is an agent protective shield, preventing direct access to its public methods. Aglet agent programming follows the established Java operating metaphor, with appropriate classes and libraries adequately defined. As hinted above, a "run" method is mandatory, to be run for every incarnation of the agent at a given location. Lange et al also authored a specialized protocol, the Agent Transfer Protocol (ATP), to handle integration of agents belonging to other platforms. In addition, a complex set of security policies has been designed to protect both the agents and the places in the system. The Aglets platform stands out via its novel leverage of the ubiquitous and heterogeneous attributes of the Java platform, its implementation of the weak migration paradigm greatly reducing the amount of data needed for agent migration, and also due to its formalizing of a set of security rules and policies meant to address the complex issue surrounding operating a mobile agent system.

2.2.4.4 Grasshopper

Grasshopper [94] is the platform of choice for our project, and will be detailed as a stand alone component in Section 3.1.

---

[85] Developed at the Japan IBM laboratory.

2.2.5. Mobile Agents Pros and Cons

Like most technologies, the mobile agents approach to distributed environments application development is not the ultimate answer. It answers specific needs like no other method; however, it has a known number of drawbacks. In what follows, we will briefly outline both.

2.2.4.5 Pros

Perhaps the most obvious advantage of using mobile agents versus classic remote procedure call is the known "bring the computation to the data, not the data to the computation" adagio. The ideal case is when the data stores of interest are heavily distributed across a network, and the amount of data to be mined is significant. In such a case, the typical Client-Server application will incur a prohibitive cost for data transfer, and in addition, will face the added difficulties characterizing the average computer network: congestion, latency, and periods of disconnect. Mobile agents on the other hand only require connectivity for migration, are limited in size, and will only return the result of the mining operation back to the initiating host.

Agents also have the distinct advantage of being pervasive to the changes in environment at the inhabited host [95]. A typical example is the shutdown case; when it occurs, agents can either migrate to other hosts, or enter "hibernation" mode, to be awaken when the host is powered up again. The typical Client-Server approach will invariably result in data or computation result loss when such an event occurs, and the requested action will have to be restarted from the beginning upon the power on.

Mobile agents are better suited for mobile device application than the Client-Server model. This is mainly due to the fact that network connectivity can no longer be taken for granted in this case[86]. First, the connection has a much higher probability to be terminated due to the mobile nature of the device, e.g. the owner travels and enters an area with no signal. Second, the connectivity may be terminated by the will of the owner; this is inherent in the device's operating paradigm. Since mobile agents become independent of the parent process that instantiated them, they can mimic perfectly the

[86] It cannot be taken for granted in the wired case either; here we focus on the wired versus wireless distinction.

unpredictable nature of connectivity typical for a hand-held device, and mold their operating time and reporting pattern to the device's actual usage.

Related to the above, but applicable to a wider range of environments, is the mobile agents' ability to function perfectly in "disconnected", non-blocking mode. The typical case here occurs when a Client issues a request, after which it blocks waiting for an answer. In addition, the Client usually must also remain connected to the network and "listen" on the established connection. These constraints no longer apply should the Client have the option to simply dispatch an agent carrying a well defined task list; the Client no longer has to enter busy-wait mode, as the agent will return on its own once it obtain the query answer (or just send back the answer, as applicable).

Yet another main area of applicability for the mobile agent paradigm is the network system administration domain. Bauman et al [91] provide the example of software distribution on demand. A typical computer network is rarely composed of hosts running the same (generation of) operating system, same hardware, and same application software. Applying updates and patches to such an environment becomes an elaborate task, especially when upgrade package dependencies are taken into account. Specialized agents may be better suited for on-demand software package deployments than a classic approach, where the cumulated knowledge of all the differences and variances in host environment needs must be centrally maintained.

Finally, a broader area where mobile agents can offset classic Client-Server methods is information processing. As Lange et al detail [95], agents are perfectly suited for monitoring activities. For example, if a given User is interested not directly in a particular information, but indirectly in its changing state, then a mobile agent permanently monitoring that particular information token is a arguably a better approach. On the same vein, agents can be highly instrumental in information dissemination activities [95]. This would be the reverse of the prior case, where the User is presumed to be interested in a particular information change, but is not actively monitoring that information source on her own. Lastly, Lange et al mention the potential mobile agents hold for parallel computation. This stems from their innate ability to duplicate themselves on the same node or a remote one. Should the processing power needed to accomplish a

task greatly surpass the capacity of a single agent, it can trigger a cloning process adequately tailored for an optimized completion time.

2.2.4.6 Cons

The core of the mobile agents paradigm's disadvantages stems from its very operating essence: once the agent leaves the parent host and arrives at its destination, a tri fold security concern arises. First, the agent's author has a legitimate concern with respect to the security of the travelling agent. For example, the agent in question may carry sensitive information, perhaps of monetary nature (like a credit card number), or of personal identification nature. Given the open nature of the internet, any connected host is open for a given attack vector, and as such, can become a hostile environment for the traveling agent. Furthermore, not only the agent's information can be stolen, but also its code base can be altered to suit an attacker's purpose. Vigna refers to this latter possibility as agent "brainwashing" [96]. And this leads to the other side of the coin: the security of the visited hosts. That is, the party owning or maintaining the hosts providing services for agents needs a form of insurance that the visiting agent is genuine, and will not engage in malicious behavior. Given that computer viruses and Trojan horse applications are increasingly common internet-related threats, a host maintainer has valid reasons to distrust any form of foreign code with executing powers arriving un-invited. As an added twist to the core trust issue, we also need to take into account the agent-to-agent attack possibility. This is the case when a malicious agent has no interest in the hosts it visits, but in the information or know-how carried by other agents, thus specializing in attacking peer agents.

This category of problems can be categorized under the broad agent-security umbrella, and several methods have been devised to address it. For example, in a similar fashion to dealing with securing internet transactions and confidential data exchange, Secure Sockets Layer protocol can be used for transfer, and cryptographic methods can be leveraged to encapsulate confidential agent data. However, the issue of agent paradigm security is complex, and far from being exhaustively addressed.

Another main difficulty in mobile agent adoption is the lack of universally accepted common platform. Indeed, though we have only introduced four agent designs in Section 2.4, all four require their own particular environment. This immediately

creates an obstacle against adoption of the technology, as service providers must now either take sides, and accommodate a given subset, or else implement and maintain all platforms to maximize their reach. In this respect, it can inferred that Java-based mobile agent technologies have an edge, as Java is a mature an – by now – certainly ubiquitous platform. However, as Lange et al [92] rightfully point out, any vulnerability affecting the Java platform automatically affects the mobile agent technologies based on it.

Lastly, as Vigna details [96], the lack of an universally accepted protocol for inter-agent data exchange is yet another major drawback against adoption of the paradigm. Given that data exchange must occur at several levels, e.g. agent-agent and agent-host, and that the agent-agent relationship is especially complex due to some agents providing specialized directory and other look-up services, the lack of a unified data exchange standard continues to be an obstacle.

While it would be unreasonable to deduce that the mobile agent paradigm is destined to be forgotten or that it will perfectly overcome all existing problems, in the absence of a "killer application", the burden remains with the application designer to infer when the appropriate conditions are met to extract maximum advantage from the technology, while insuring that the applicable security concerns are appropriately addressed.

2.3. Mobile Agents in Intrusion Detection

Conceptually, the mobile agent paradigm establishes itself as a solid alternative to the Client-Server model, via its ability to address the core drawbacks of the classic model. It is to be noted however that the mobile agent paradigm has its own drawbacks, as we detail in Section 2.2.5, and also a cost. Our work aims to exemplify an instance where the cost-benefit analysis favors the mobile agent model, and several researchers have found the network intrusion detection area as a prolific application field for the technology.

Perhaps the earliest established mobile agent-based IDS is AAFID (Autonomous Agents For Intrusion Detection) [97]. The authors propose the mobile agent paradigm in an effort to mitigate the main drawback of the classic architecture IDS: the single host data gathering and processing. The authors envisage a three tier architecture, with *agents* at the lowest level, *transceivers* immediately above, and *monitors* at the top level. Agents have the role of low-level workers, in charge exclusively with data gathering and its transmission to the transceivers. The latter components have a dual role: control and data monitoring. From a control perspective, they keep track of the agents running on its assigned host, start and stop local agents, and communicate with the monitors. Transceivers also have limited data processing capabilities, transmitting only a digest to the top level. A monitor's main role is *correlation* of the data supplied by the various transceivers; ultimately, the monitors are the components raising an alarm, and communicating with the SysAdmin via AAFID's user interface. A key point of this work is the concern for concurrent data access, as several agents may "fight" for the same audit log. The authors thus proposed the use of an *audit router*, designed following a blackboard metaphor, and keeping track of subscribed agents and the specific class of data they are interested in. Balasubramaniyan et al have built a highly modular system, capable of incorporating agents written in any other language – as long as they implement the transceiver communication interface, and affording component replacement without the need to restart the IDS. We also retain as its main merit the moving of the computation at the lower levels of the IDS's architecture, thus departing with the known central processing metaphor.

Asaka et al [99] start building their IDS by concentrating on the occurrence of most probable signs of intrusion on a networked host; they denominate this token MLSI, for Mark Left by Suspected Intruders, and it represents an atomic detectable event likely to indicate intrusion. The system the authors built (IDA: Intrusion Detection Agent System) has a highly componentized structure, with specialized workers for each logical task. In a manner similar to AAFID, IDA utilizes *sensors* at the lowest, data collecting level. Once an MLSI event is reported to a *manager*, the latter spawns a *tracing agent*, whose role is to mimic the path followed by the intruder, in an effort to assess the footprint of the attack. At each visited host, the tracing agent activates an *information-gathering agent*, whose main task is MLSI data mining. This data is shipped to the manager, where it is stored on a blackboard system; the manager interacts with the SysAdmin via a local interface. The paper features an interesting evaluation result, where in a simulated typical attack (privilege escalation, file-related permissions attacks, buffer overflow) IDA was effective in 92.3% of the cases. The work by Asaka et al stands out by being among the first to leverage the *mobility* capability of the agent platform.

The Micael system [100] constitutes an important step in agent based IDS technology, via its novel agent role segregation, and most importantly via the implementation of an agent audit mechanism. That is, at the extremes of the worker hierarchy, Micael is no different: it equally features a *head quarter agent* at the top, and a *sentinel agent* at the lowest level. First of all however, even the head quarter agent can move to another host under certain circumstances (e.g. its host is under attack). Then, de Queiroz et al define a highly specialized agent, not performing any IDS specific duties, as an *auditor*. Its role is to monitor the health of Micael's agents, including the head quarter, and terminate and recreate them according to a set of agreed heuristics. Of particular interest is the fact that all auditor agent related communication follows a separate protocol, specialy designed for auditing purposes (i.e. not the same used by the other agents in the system). In addition, all messages between any two agents are authenticated and encrypted. Last, given that Micael uses Aglets[87] as its platform, its reach of monitored platforms is far greater, and in fact, the pilot project presented in [100] monitored a mix of UNIX, Novell Netware, and Windows hosts.

[87] We introduced Aglets in Section 2.2.4.

The work by Helmer et al [101] constitutes a landmark in mobile agent based IDS, for several reasons. First, the authors decided against using "regular" agents; that is, their system utilizes a special type of mobile agents, *lightweight.* These agents are the byproduct of a specific agent platform[88] and stand out by being strictly minimalistic implemented. Notwithstanding, this type of agent has build in upgrade capabilities, making thus perfect candidates for "traveling light" and upgrade-on-demand activities. For example, such an agent needs not know to what platform is it destined for; it will learn that upon arrival, and request UNIX or Windows (for example) capabilities as appropriate. The choice of agent technology aside, the work by Helmer et al further distinguishes through its communication mechanism and system hierarchy structure. In particular, communication in the system is not only vertical, from lower-level agent to higher-level ones, but also horizontal – agent-to-agent. This decision was made so as to be able to dynamically increase a probing agent's sensitivity, in an effort to curb false positives. A good example here is the occurrence of repeated false log-ins: are they due to a legitimate user mistyping or not remembering her password, or due to an attacker's attempt to guess an account's password? The IDS uses horizontal communication to perform real-time correlation (with events observed by another agent, at another host, perhaps) and infer at the lowest level in the IDS hierarchy which situation applies most likely. On the topic of system architecture, the IDS Helmer et al built features in addition to the usual upper, middle, and lower tier, a sub-lower tier level. The role of the latter is pure *data cleaning*; this category of agents are the only ones aware of the operating system installed on the monitored host, and their role is to minimally parse the audit data, and format it for consumption and aggregation at the lower level. The low-level agents are platform independent (like all the other agents in the system), and highly specialized in specific attack detection, e.g. failed log-ins, FTP attacks, etc. The low level agents are managed by *mediators*, who aggregate audit data and submit it to the IDS's database. From there, *data fusion* and *data mining* agents parse the data asynchronously and infer attack patterns. Lastly, the IDS leverages *dynamic aggregation*, a Voyager platform built-in capability to enhance low-level agents' competences while the IDS is running as response to various events. Helmer et al conducted several experiments with test data to

[88] The Voyager Java platform; please refer to the work cited for details.

infer the false alarm and accuracy rates of their system, and the results were encouraging[89]. We retain as this work's main contribution its leverage of lightweight agent technology, enabling thus on-demand agent upgrades as well as horizontal agent communication in the context of detecting network intrusions.

As an alternative approach to mobile agent IDS, we would mention the SPARTA system (Security Policy Adaptation Reinforced Through Agents) built by Kruegel et al [102]. SPARTA differs from the other mobile agents based IDS from a high level approach: it is mainly a security query mechanism, allowing the SysAdmin to perform aggregate security queries regarding particular local policies violations or known attacks. To obtain the results, the pattern entered via the UI is entrusted to an agent whose first task is to query a directory service for hosts mentioned in a specific query clause. The agent then visits each host in turn, and looks for events matching the query terms. When a match is found, the "seeker" agent makes a request for a helper agent which will start an investigation at the host mentioned in the audit data as the originator of the event. The seeker agent waits the return of the helper(s) it spawned, aggregates their results, and returns the correlated data to the UI. To enable this mechanism, Kruegel et al developed a query language based on SQL syntax, and allowing terms for metadata of local interest (e.g. IP ranges, send/receive, etc). The authors have also considered IDS security in their implementation, and implemented an elaborate system of privileges. For example, "seeker" agents can only communicate with the helper agents they have spawned. In addition, agents' privileges are rigorously limited, and cryptographic signatures for cross-network agent migration. We retain SPARTA's contribution as having implemented the correlation activities in fully distributed manner, at the lowest level of the hierarchy: the "seeker" agent level.

Pushing the notion of distributed IDS to a new extreme, and also implementing an anomaly detection metaphor is the work by Foukia et al [104], [105]. In their earlier work ([104], 2001), the authors set out the groundwork for their approach to anomaly based intrusion detection: mimic, with the help of mobile agents, the immune system. As the anomaly metaphor dictates, a first stage must be dedicated to learning "normal" system behavior, accomplished by collecting a significant sample of system calls from a

[89] Please refer to cited work for details.

controlled environment. In the detection phase, mobile agents record deviations from the normal system-call pattern surpassing a pre-defined threshold, and raise an alarm. The distributed aspect is emphasized via the fact that the mobile agents have no "home"; they constantly roam the subnet-divided network of the system they protect, performing their task uniformly throughout. This approach is improved in the authors' later work ([105], 2003) by introducing pheromone-based communication based on the social insect model. The intrusion detection phase is unchanged; however, the intrusion response scheme is now based on *pheromonal messages* attracting rapid response agents and guiding them towards the source of the intrusion alert. We retain the work by Foukia et al as pioneering an alternate means to central IDS management: the mobile agents in this system are only aware of their immediate surroundings, and yet are able to respond to alerts in a manner similar to the "dispatching" model found in central-command-post implementations.

2.4. Voice over Internet Protocol

Some sources [106] credit the 1996 Israel based company *VocalTec* with the manufacture of the first IP telephony gateway connecting the internet protocol with a PSTN[90] line. Several years later, according to a July 2005 *Information Week* report [107], the North-American VoIP market will hit four billion dollars by 2010, a growth of over 1,300% over 2004. According to the same report, *Frost & Sullivan* [108] predicts that a significant number of non-telecom companies will enter the VoIP market for a share of the profits, raising the number of VoIP lines to eighteen million by the same year.

Given that Internet Telephony does not rely on an established connection between the conversing parties, one has to wonder "why bother" trying to mimic that model over the Internet Protocol, whose functioning is entirely different. We infer the answer must be convenience combined to added features for most North-American markets. In other markets (such as Europe), where internet bandwidth is several orders of magnitude more expensive [109], [110], cost is an added reason. For example, most working professionals need to keep track of their email, handle voice calls and voice mail, and use a fax and a mobile phone on daily basis. VoIP technology could collapse all the incoming communication of such a professional into one unified messaging location [111]. We equally argue that most internet connected consumers currently paying for a second telephone line stand to gain from transforming the second telephonic land line into a VoIP one. Perhaps more attractive for businesses large and small would be the significant enhancements in teleconferencing VoIP will bring. On a classic line, the most Users can do is add several party to the audio flow. With VoIP, which in the simplest teleconferencing case will not require additional equipment besides the existing work stations, the addition of video becomes trivial [111]. In addition, several electronic white boarding applications can easily be leveraged, and durations in the order of hours no longer pose a financial burden on the participants.

Our position is that VoIP is a technology that will likely increase its presence over the incoming years, expanding its adoption to a significantly diverse category of Users. We are thus motivated in our approach to build an Intrusion Detection System for VoIP

[90] Public Switched Telephone Network; the classic telephony technology.

as the technology is not without exploitable flaws, as we have already hinted in Section 1.1. Before we scrutinize its vulnerabilities however, we would like to introduce its main building blocks.

IP telephony needs little components aside internet connected computers – the *terminals*, as long as there is no need to interface with regular telephones, and Users are happy interfacing with the audio-video capabilities of their workstations. The terminals can also be more sophisticated and dedicated VoIP handsets, manufactured to implement a specific protocol (see following subsections). Regardless of end point, a *(call) server* is required next to map the association between an end point and its IP address. The server also performs other duties, like call authorization and accounting, in the same manner as a traditional PBX[91] [114]. Since most users will welcome the ability to call regular phones however, a *gateway* is required. Its main duty is to enable voice connectivity between IP and public telephony networks, by translating one signaling protocol into the other – voice signals are analog, and need to be transformed into digital. Voice based communication, be that over IP or traditional, remains a connection-based technology, and thus requires a *call setup phase*. This is achieved today via two main standards: *H.323* and *SIP* (Session Initiation Protocol).

In what follows, we will briefly introduce the two standards, and present their main components and functional areas, for the purpose of introducing our readers to the technology utilized in our project. For a detailed view of the technology, we refer our readers to the works cited: [106] – [114].

2.4.1. H.323

H.323 [115] became an ITU-T[92] approved standard in June 1996 [114]; it was initially developed to afford a broad area of digital communication, like video and other multimedia; VoIP was viewed at that time as a special case of video telephony [112]. It was quickly adopted by the major vendors, as it allowed development of proprietary

[91] Private Branch Exchange; the server component for traditional telephony.
[92] International Telecommunications Union

implementations capable of interoperating. The main components required by the standard are *terminals, gateways*, and *gatekeepers*.

The gatekeepers perform access control for the entities involved, and also bandwidth control and call management (address translation, zone management) [112]. End points must thus register with their respective gatekeeper before it can perform active call duty. *Call signaling* is the process involved in fulfilling a voice connection between two end points. Its observable actions follow the classic telephony metaphor: a terminal will ring upon an incoming call or will provide dialing tone when picked up, likewise, the appropriate end-call processes will ensue once the call has ended. Signaling can be performed with various degrees of gatekeeper involvement. At one extreme, we have *direct endpoint call signaling*, which involves the gatekeeper only for address translation and admission control. At the other end of the spectrum, the gateways only handle the media stream between the two endpoints involved in a call; everything else is in gatekeeper's charge [112], [114]. The main signaling protocol used in H.323 is Q.931 [116], [117].

The gateways are perhaps the most critical component of the Internet Telephony metaphor from a User perspective: they perform the translation between the IP and telephony networks. There are two type of gateways required for a VoIP call: *signaling gateways*, converting classic telephony signals into IP contra parts, and *media gateways* (MG in what follows), performing voice transcoding [114]. Media gateways do require a "master" component however; the *media gateway controller* (MGC in what follows), whose role is to interpret call signaling, support multimedia streams, and handle termination of streams at the participating MGs.

For an in depth view of the inner workings of the H.323 protocol, we refer our readers to the works cited: [115] – [124].

2.4.2. SIP

The Session Initiation Protocol [125], [126] was developed by the Internet Engineering Task Force[93] (IETF) as an alternative to H.323. In particular, SIP intends to

[93] See http://www.ietf.org/ for an overview of the organization.

overcome the master-slave architecture of the H.323 protocol, and was thus designed with the Internet metaphor in mind: it is an end-to-end signaling protocol, which eliminates thus the MG and MGC components from the architecture [114]. This accomplishes its main design goals: improved scalability and elimination of single points of failure [129]. Furthermore, the SIP architecture is also a radical departure from the classic telephony architecture, where the end points are equally "dumb", and the intelligence is located either on the network itself, or else at critical components along the network. This results in an incomparably higher potential for enhanced user applications available at the terminal level.

The end points in the SIP architecture function thus according to the peer-to-peer metaphor, and are called *user agents*. As the HTTP metaphor implies, each user agent can act as either Client or Server; dependent upon which party initiated the call, we distinguish *user agent client* (UAC) and *user agent server* (UAS). Given the nature of the protocol, a connection between two remote end points will invariably require a number of SIP Servers in between to handle call logistics. SIP Servers can thus be *proxy servers*, which simply receive a request and forward it to the appropriate agent, *redirect servers*, whose role is to instruct the initiating agent to forward all subsequent requests to a specific user agent, *registrar servers*, processing requests from agents wishing to join the network, and finally *location servers*, provided directory services [114].

For an in depth view of the inner workings of the SIP protocol, we refer our readers to the works cited: [125] – [132].

2.4.3. Media Gateway Control

2.4.3.1 H.323

Protocols like H.323 and SIP are *signaling* protocols; they fulfill the work required to establish a session between the calling parties, and are especially visible at the interface between internet and classic telephony worlds; both SIP and H.323 are peer-to-peer protocols. The transit of *media* between the two technologies however is handles via a separate, client-server, *media gateway* protocol. The master component is the *Media Gateway Controller*, and it can handle several *Media Gateway* components. The initial protocol for MG-to-MGC communication was the Media Gateway Control Protocol

(MGCP) [118], [119], [120], and it was the effort of private enterprise [114]. Since not all technology providers were part of the initial camp, an effort was launched to group all the (remaining) vendors under the umbrella of the ITU-T. The result of this joint effort was *H.248*, or *Megaco* [121], [122], [123].

2.4.3.2 SIP

Just like in H.323's case, a separate protocol is required for the actual data (multimedia) communication. In SIP's world, the protocol in question is RTP [130]. After the SIP call set-up has successfully completed, each (SIP) terminal takes samples of the input audio stream (analog), performs the conversion to digital, and encapsulates it into an RTP *header*. The header contains information related to the voice encoder used by the sending party, as well as sequence information. Its first twelve bytes are mandatory, whereas the last ones come into play when the packet was routed by a server routing more than one RTP flow [114]. In conjunction with RTP, the Session Description Protocol (SDP) [131], [132] is used to describe multimedia sessions for all types of Internet Telephony applications; this is achieved in a manner similar to MIME encoding for internet email [106]. It basically allows two (or several) connecting endpoints to accomplish a handshake with respect to the media format and type to be used during the SIP session.

As we have mentioned in Section 1.1, the VoIP technology is vulnerable first and foremost due to its inner working taking place over the internet. In Section 1.2, we have amply detailed the nature of attacks internet-connected systems are prone to; in what follows, we will focus on VoIP-specific vulnerabilities as motivation for our work.

2.5. Vulnerabilities Affecting Voice over Internet Protocol

The concern for the security of telephony is not new, and was certainly not triggered by the advent of Internet Telephony. At the beginning of the classic telephony, physical access to the line or PBX was required for a successful breach of security. Modern telephony switches however have remote management capabilities, and are accessible from a typical desktop setup – attacking them became thus a matter of typical network intrusion. What is pertinent – in the classic telephony context – is that a sense of false security, triggered by the perceived "closed" nature of the system, lead to deployment of components with little or no self-defense mechanisms, and with vulnerabilities reminiscent of typical internet connected devices. For example, Bhattacharyya [133] uncovered in 1991 that "intelligent" telephony switches are open to security threats such as *masquerade*, *information leak*, *message modification*, *unauthorized system access*, *service denial*, and *traffic analysis*. Furthermore, Bhattacharyya surveyed a series of switches manufactured by leading providers, and noted that some switches do not have a password aging mechanism, and furthermore, do not make special ASCII characters mandatory. Also, in some cases, failed log-in attempts were accompanied by a hint indicating which of the tokens failed (password or user name). In addition, logging for auditing purposes was not available for all types of actions, and that the product documentation would not have a dedicated section for the security measures implemented by the switch in question. Roughly ten years later, the convenience of immediate access to internet and the explosion of services designed to take advantage of it made internet telephony part of the daily routine for many businesses and individuals. Given the open nature of the internet however, the range and nature of vulnerabilities targeting VoIP could only increase.

2.5.1. Overview

First, we can easily identify a series of attacks that are either common to both classic and IP telephony, or else straddle the border between the two. One main example of such an attack is *eavesdropping*. In classic telephony, this could be achieved via physical access to the circuit line, or else to the endpoint device of interest. Both can now

be achieved remotely by the skilled attacker via www. Yet another is *man in the middle*. Not exactly trivial in classic telephony, but can be achieved by recording one User's monologue, altering it via an electronic device, and replying the altered recording in a completely different context. This type of attack is much simplified in internet telephony, where the voice data is conveniently segmented in www-traveling packets. As another facet to this attack, man-in-the-middle can now be performed by impersonating an MG or an MGC for H.323, or a SIP Proxy, thus obtaining access to confidential call metadata [134]. Lastly, *toll fraud* can now be achieved with greater ease by simply impersonating the legal User's endpoint credentials at the attacker's endpoint. Second, the VoIP technology opens the door to a class of attacks without a contra part in the classic telephony world. To point just one salient feature, a VoIP soft-phone is no longer bound to the same physical location; instead, it travels freely wherever the User of the application travels – especially true if the endpoint is installed on a mobile device.

One main example of an attack avenue which was not open in the classic telephony metaphor is the remote end-point vector. Indeed, a classic telephone can be directly attacked (for example by installing a remote transmitter or recorder) only via physical access to its location. A VoIP endpoint however, first can be accessed via internet, and second, is not a simple electronic-mechanic combination device. It has to implement a specific set of features, and run appropriate procedures at the appropriate times. If any of the services such an end-point runs on regular basis is not secured with a password and its port is known, then an attacker could easily exploit this as a point of entry. Alternatively, a VoIP customer can become the victim of *theft of service*; this happens when an attacker can perfectly impersonate the legitimate User by stealing her – soft phone – user name and password. Lastly, an internet telephony end point's set of procedures is usually implemented on top of a main stream operating system[94]. As such, VoIP devices thus configured will inherit whatever vulnerabilities are inherent to the underlying operating system technology.

Moving further down the chain of VoIP components, we cannot escape the fact that the endpoints communicate upstream over the internet. As such, they appear to an outsider as simple devices featuring an IP address. If the latter is known, then a series of

[94] A Windows version or Linux clone.

typical attacks become feasible: DoS, DDoS, SYN floods, packet capture, etc. DoS in particular receives a new facet in VoIP by making an end point reject specific, or all calls as per an attacker's wishes [134].

Finally, given the different architecture of the two dominant VoIP protocols (SIP and H.323), we expect specific vulnerabilities targeting each protocol and their components specifically. In what follows, we will briefly outline the main attack vectors for the two competing technologies, as uncovered in controlled laboratory environments.

2.5.2. H.323

We have introduced the H.323 standard in Section 2.4.1; we now present how its specifications may be used as an attack vector against a given implementation.

As Ackermann et al note [135], both the payload (i.e. audio data) and the signaling information exchanged between the implementation's nodes are subject to eavesdropping, DoS, or capture. This is especially true since every technology vendor implements their own proprietary view of the standard. Ackermann et al point out however that another partition class exists: the end point managements. Most VoIP devices offer a remote management interface, whose password, as shown in [135], is sent in clear. Furthermore, given the limited range of characters available on the phone's keypad, the password itself is bound to be week. Should this be captured, then by simply viewing the endpoint's configuration intimate knowledge of the (VoIP) network configuration can be gained. The authors further point out that though the effort in creating a private, altered copy of the devices firmware is rather prohibitive, the possibility to upgrade the endpoint to an altered version of the original firmware is perfectly achievable. Furthermore, Ackerman et al were able to successfully perform a DoS attack against an endpoint due to the presence of an integrated www server in the devices; the server in question was vulnerable to a known buffer overflow (URL longer than a specific value). Equally, various attacks at the gatekeeper level were enacted in [135]: *deregistration* – presenting spoofed packets from an attacker machine thus mimicking the User moving to another host, and DoS.

Perhaps best known in the academia (and not only) is the work at the University of Oulu, in Finland: the PROTOS project [136]. The group has performed extensive work in the area of VoIP protocol testing, and the PROTOS project expands over several years; we refer our readers to the work cited for an in depth analysis. We mention here the group's 2003-2004 H.323 testing project, whose outcome was the discovery of a series of vulnerabilities affecting close to 95% of the technology vendors, and which triggered an avalanche of security advisories from watchdog agencies and vendors alike [137], [138], [139].

As a reminder that we cannot judge the protocol in isolation, and that the underlying operating system must be taken into account, we also mention the vulnerability discovered equally in January 2004, whereas the H.323 filter component of the Microsoft Internet Security and Acceleration Server is vulnerable to a buffer overflow, ultimately allowing an attacker to execute code on that host [140].

Not surprisingly, ITU-T has undertaken a separate body of work for securing the H.323 protocol, and the main outcome is concretized in the H.235 security framework. As per H.235 specification, authentication happens on a per-person, as opposed to per-device basis, via the use of specific cryptographic techniques. For in-depth details, we refer our readers to the standard's specification [141].

2.5.3. SIP

We have introduced the SIP standard in Section 2.4.2; we now present how its specifications may be used as an attack vector against a given implementation.

Posegga et al outline in their work [142] a series of vulnerabilities extracted from the SIP's RFC specification [125], and which bear a (non)surprising similarity with the ones we have outlined in the H.323 Section: *registration hijacking*, *impersonating a SIP server*, *tampering with message bodies*, *tearing down sessions*, *denial of service* – from a protocol management perspective, as well as others from the area of service usage perspective: *call hijacking*, *eavesdropping*, *endpoint impersonation*. Without loss of generality, we can single out DoS as a definite vulnerability in SIP's case perhaps more so than in H.323's case, given the open nature of the SIP servers and SIP proxies.

Salsano et al [143] further identify *snooping* – the act of obtaining illicit gain of the Users' identities, services, and network configuration, *modifications* – attacks meant to replay maliciously modified packets of the original stream, and *spoofing* – the act of impersonating a legitimate User.

As a case in point, Qiu et al describe in [144] the successful enacting of the following attacks: *replay attack, registration hijacking, INVITE request spoofing*, and *session teardown* performed in a controlled laboratory environment against a specific implementation of the SIP protocol.

Last but definitely not least, the PROTOS project evaluated several implementations of the SIP protocol as well [145], and discovered that from a series of nine different implementations under test, only one was unaffected by the tests' input.

As the protocol matured, several mechanisms were implemented to enhance SIP's security. With respect to the *end-to-end* aspect, two main methods have been singled out: *digest authentication* and *S/MIME*. The former specifies that the User parties should mutually authenticate before starting a connection, to insure the integrity of the connection initiation data. The latter, Secure MIME [146], is an option for SIP as it does use MIME for message bodies in the first place; extending to S/MIME would provide confidentiality and authentication at the same time [142]. However, as Salsano et al point out, end-to-end encryption cannot be applied to messages meant to be read (or worse, written) by intermediate – but legitimate – SIP nodes [143]. There exist another aspect of the SIP protocol where security mechanisms can be applied: the *hop-to-hop* aspect. However, since SIP does not provide security mechanisms of its own for the channel, it relies on either protocol level mechanisms – such as IPSec, or on transport level mechanisms – such as TLS [147]. Nevertheless, given that such security mechanisms also apply to H.323, we will briefly discuss them under a common umbrella. With respect to TLS, we will only mention that just because a SIP agent makes a TLS request to its next hop proxy, there exist no guarantee that TLS will be used form the next hop onwards [143].

2.5.4. Difficulties in Securing VoIP

Given that internet telephony in general uses the internet as its connectivity vehicle, we will focus our attention on IPSec as a security mechanism. At a first glance, IPSec seems to be the obvious answer to most grievances regarding the security of internet telephony. Unfortunately however, IPSec fails on the Quality of Service (QoS) aspect of the VoIP. In particular, the real time nature of internet telephony makes the technology highly sensitive to packet delay. As a case in point, the maximum acceptable delay in voice packet delivery is in the 150ms – 200ms range [114]. Furthermore, this interval must also include any buffering delay performed at the MG/SIP Proxy to alleviate for the effect of jitter[95]. To guarantee these constraints, voice packets are typically smaller than HTTP packets for example: only 10-50 bytes worth of payload. Once IPSec's overhead is taken into account, the size of the payload decreases even more, with a corresponding effect on VoIP quality. Barbieri et al for example measured a decrease of 63% in effective bandwidth for the same VoIP connection when IPSec was utilized [148].

There exists another type of overhead however associated with IPSec, aside from the mandatory use of IPSec headers: the cryptographic requirement at the router level. For IPSec traffic, new headers are constructed and packets are encrypted accordingly. Barbieri et al noted that the problem does not result from the overhead caused by the encryption procedure, but rather by the unavailability of a VoIP prioritization mechanism [148]. That is, in all cases where the VoIP traffic did not utilized a dedicated network, the cryptographic engine scheduler invariably prioritized FTP and or HTTP packets before the voice ones, and this culminated with the voice packets being discarded al together.

Given this limitation of IPSec for VoIP use, we draw on evidence that there exist a valid need for an internet telephony protection mechanism applicable to the current[96] VoIP technology, and to the current data transport vehicle. Furthermore, we advance that such a mechanism must take into account the distributed nature of the components enabling internet telephony, and shall not concentrate solely on the endpoint terminal.

---

[95] Packets arriving in a mixed order; see http://www.answers.com/topic/jitter?cat=technology for details.
[96] As of 2005, the completion date of our project.

What is more, we advance that the Intrusion Detection Systems' operating metaphor – which we introduced in Section 2.1 – is ideally suited for such a task.

Chapter 3

3. Integrated Off the Shelf Components

In this chapter, we introduced the Components Off The Shelf (COTS) we have integrated into our project.

3.1. Mobile Agent Software Component

We selected Grasshopper [149] as the mobile agent platform of choice, and enumerate the following reasons in support of our choice:

1. Open Source (at the conception/prototype time) [97]
2. mature
3. proven and recommended by others
4. strong user community
5. strong technical support
6. strong (integrated) security mechanisms
7. multi platform

3.1.1 Grasshopper Overview

In this section we provide an overview of the Grasshopper Mobile Agent platform. The figures are inspired from the *Grasshopper Basics And Concepts* Release 2.2 [149]. This document was utilized by us under the terms of the Open Source license, as we have obtained both the software and the related documentation prior to the platform becoming a private enterprise [150].

The main structural concept in the Grasshopper platform is the *Distributed Agent Environment* (DAE); a snapshot of its hierarchical composition is shown in Figure 2 [149]:

[97] The Grasshopper platform is no longer an Open Source project. It can still be obtained and utilized, but as a commercial product only.

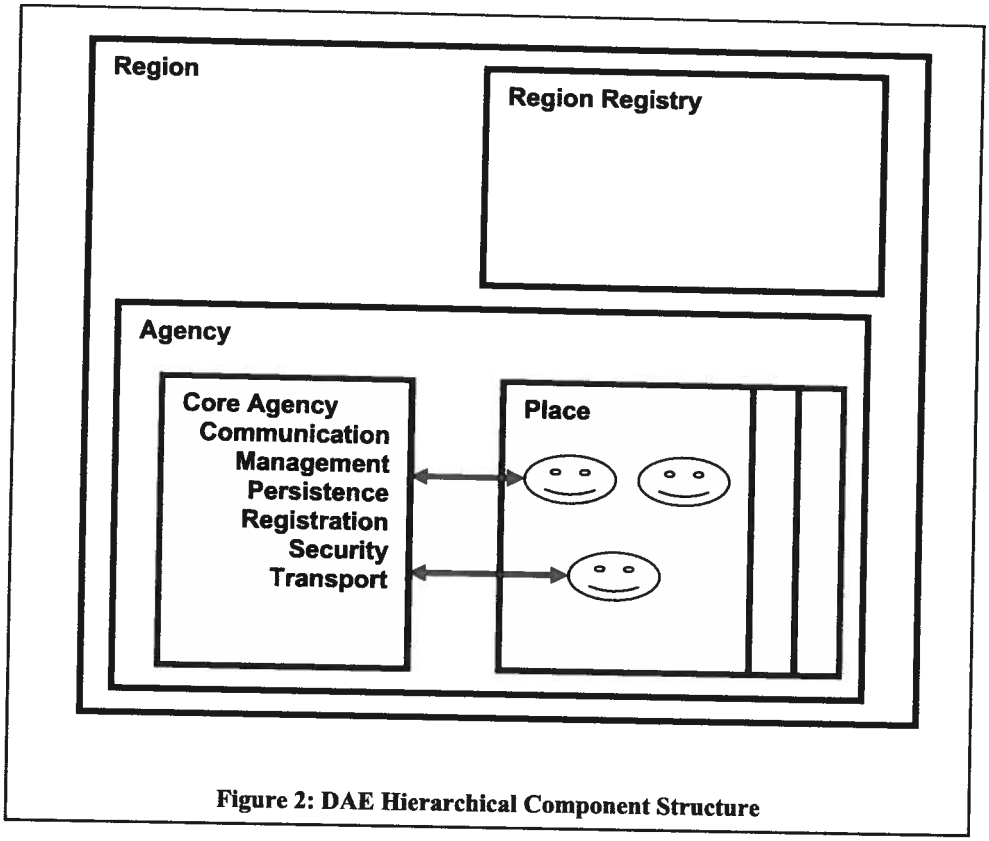


Figure 2: DAE Hierarchical Component Structure

The key components of the *Grasshopper* structure are thus: *region, agency,* and *place.* The hierarchy of the DAE's components is thus (in order of increasing complexity):

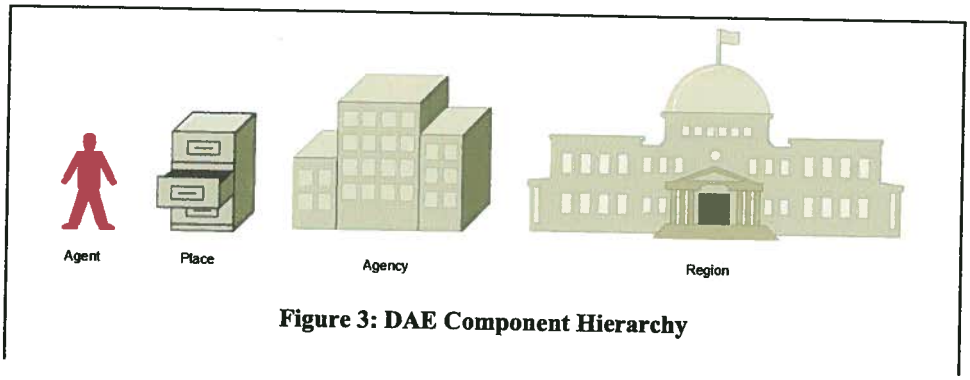


Figure 3: DAE Component Hierarchy

3.1.2 Grasshopper Communication Concepts

3.1.2.1 Overview

Communication between agents (within the Grasshopper platform) can be achieved via two main approaches:

1. The (implemented[98]) Grasshopper Communication Service (GCS)
2. OMG MASIF-compliant CORBA interface for remote interaction. That is, the Object Management Group (OMG; the most important standardization body in the area of mobile agents) has adopted in February 1998 the Mobile Agent System Interoperability Facility (MASIF), as its new mobile technology[99]

3.1.2.2 GCS Supported Protocols

GCS has provisions in its code base for support of any of the following communication protocols:

1. Internet Inter-Object Request Broker (ORB) Protocol (IIOP)
2. Java's Remote Method Invocation (RMI)
3. plain socket

In Figure 4, we provide a schematic drawing outlining the potential for interaction between two *Grasshopper Agencies* implementing the protocols above [149]:

[98] This is the method utilized in our implementation.
[99] We have introduced this technology in Section 2.2.

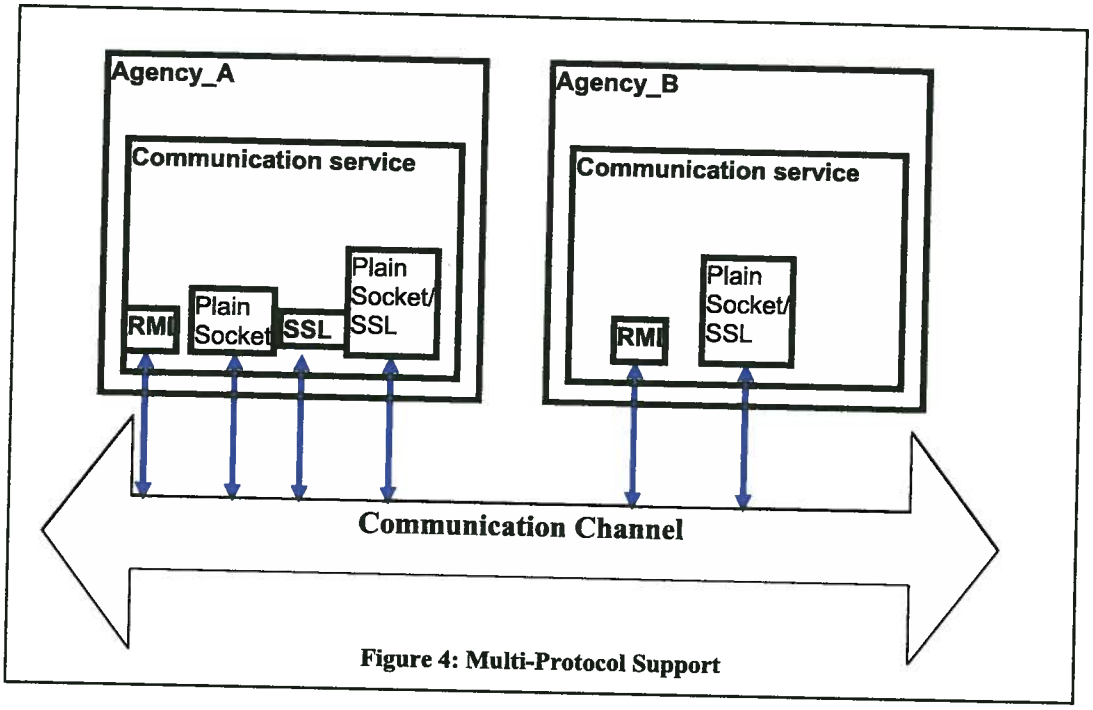


Figure 4: Multi-Protocol Support

As Figure 4 shows, it is possible to combine Plain Socket communication with Secure Socket Layer (SSL) to increase security, but at the cost of connection speed. Figure 5 for example, presents such a combined Plain Socket/SSL use, where the communication is secured with SSL only for communication between intranets, but it is allow to flow via Plain Socket only inside a given intranet [149].

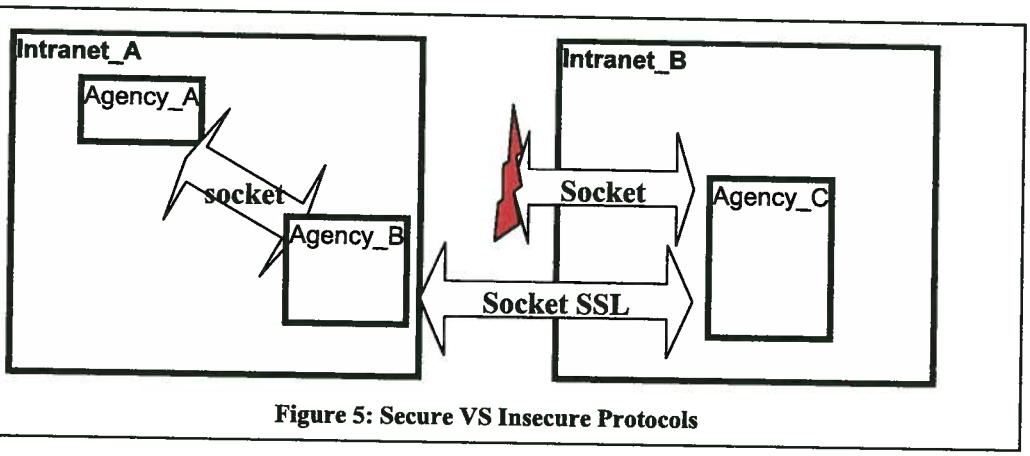


Figure 5: Secure VS Insecure Protocols

3.1.2.3 Location Transparency

Within the agent code, RMI's look exactly like local method invocations on objects residing on the same Java Virtual Machine (JVM). This is achieved by the use of *proxy objects* that are directly accessed by a client. The proxy objects forward the calls via the ORB to the remote target object (viewed as server, in this scenario). Figure 6 shows a schematic drawing of this approach [149]:

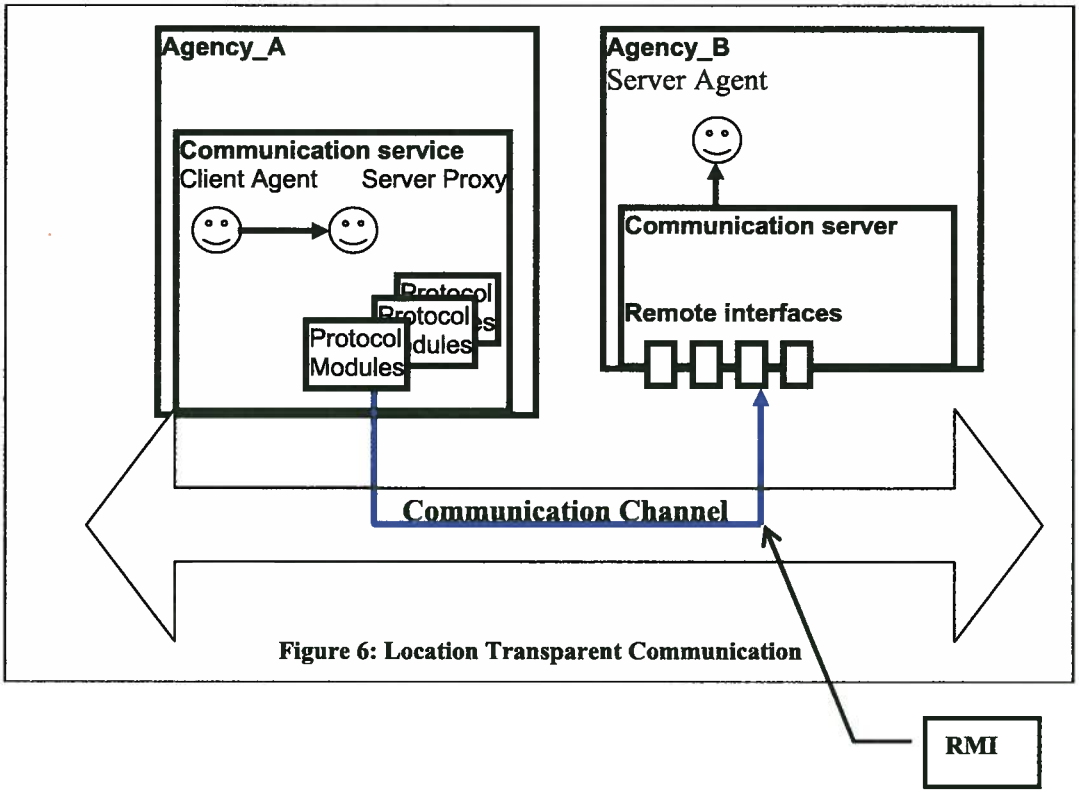


Figure 6: Location Transparent Communication

3.2. Voice Over Internet Protocol Software Component

3.2.1. IUT Overview

In this section, we present the COTS VoIP Implementation Under Test (IUT). This component constitutes the host software application which our mobile agent IDS will inhabit. Given that this particular IUT was provided to us for research purposes and under a Non Disclosure Agreement, we will refrain thereafter from naming the software vendor or the name of the product.

The IUT implements a Megaco-H.248 Toolkit, and provides [151]:

- Full MG and MGC stacks
- RTP/RTCP stacks
- Synchronous and asynchronous operation
- Multiple stack instances per process space
- Full expandability via custom APIs
- RFC 3015 and H.248 compliance
- Soft analog phone simulator

3.2.2. IUT Communication Protocols

The IUT communication operations comply with the following standards [151]:

- Standard compliance retransmission mechanism:
    - Transaction => Reply => Response ACK
    - Smooth round trip time (SRTT)
- Standard compliance Fail-over support
- Statistics gathering

3.2.3. IUT Supported Operating Systems

The IUT supports a broad range of operating systems; of interest are [151]:

- Windows 2000, Windows NT, Windows 98
- Solaris 2.6, 2.7, 2.8
- Red Hat Linux (6.2)

3.2.4. IUT Components of Interest

3.2.4.1 Core Components

On top of TCP and UDP layers, the IUT implements a series of components enabling telephony over IP[100].

First, the IUT features a full implementation of the SDP protocol [131], [132], for multimedia session management. Then, an RTP/RTCP component [130] is provided. Last, an H.248 [122], [123] full stack component is implemented, all underneath the top level MG/MGC layer implementations.

Furthermore, the IUT allows multiple instances of MG/MGC to coexist in a thread-safe process space, affords both blocking and non-blocking request/reply interactions, and exposes a set of callbacks from the Stack back to the User space for easy event notification.

3.2.4.2 Logging Component

The IUT features a full logging capability, capable of collecting an exhaustive set of run-time event and process information. It provides the programmer with the ability to set the log level according to local needs, and also to instrument code of interest for additional logging via exposed APIs [152].

3.2.4.3 Soft-Phone Component

The IUT offers a soft analog-phone simulator component as part of the toolkit. Its basic functions are as follows [151]:

[100] The basic building blocks have been explained in Section 2.4.

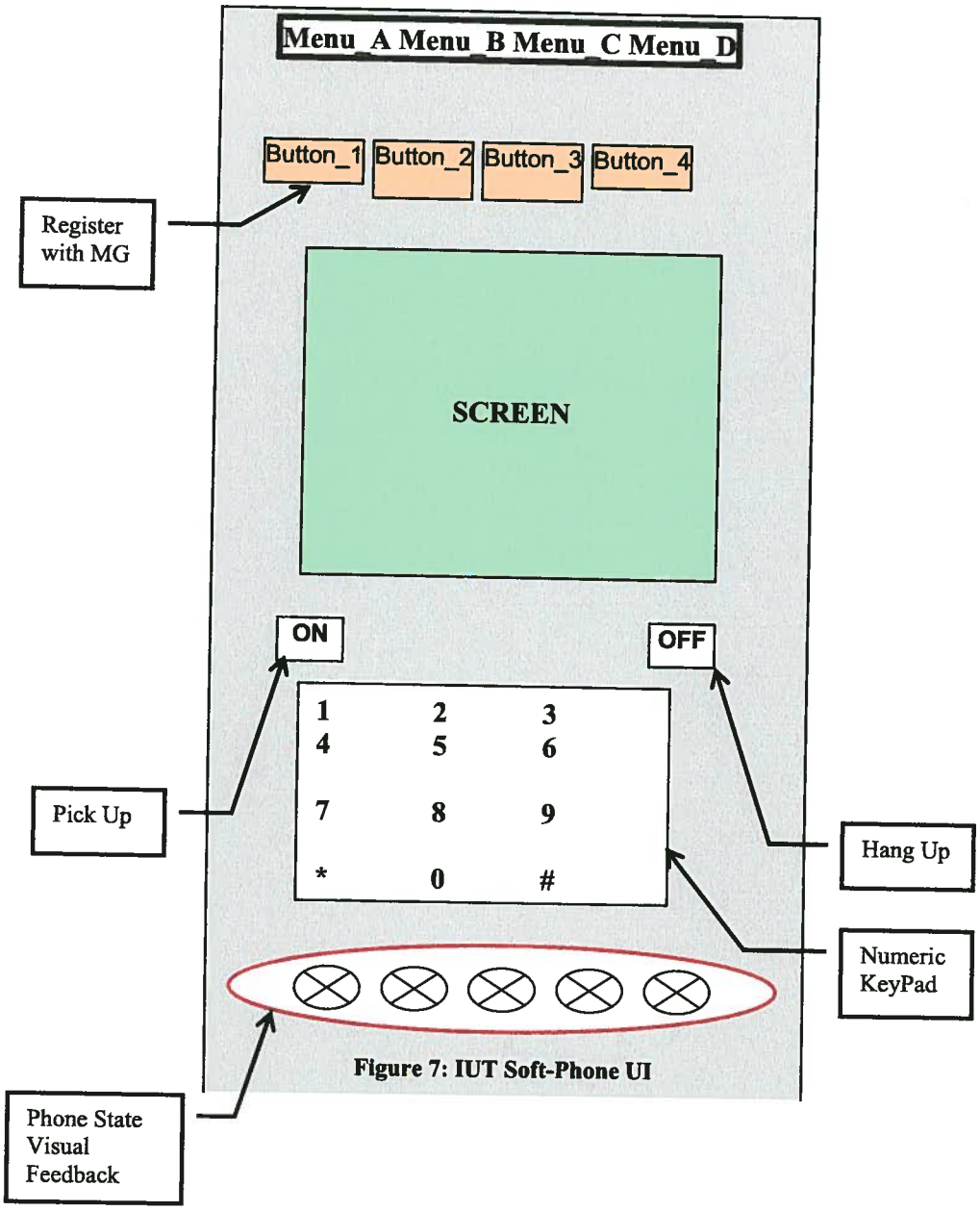


Figure 7: IUT Soft-Phone UI

3.3. Firewall Software Component

We have incorporated as part of our project the *Peer Guardian* Open Source firewall component [153], [154] for the following reasons:

- Compatible with the Windows operating system
- Blocks IP addresses as read from a configuration file
- IP Block configuration file can be updated at run-time
- Blocking occurs at OS kernel level, not in the upper layers

The application also features a graphical interface, which provides a point-and-click interface to the configuration options. For our project, we are using exclusively the configuration file for IP insertions [155].

Chapter 4

4. The MAIDS for VoIP Software System

4.1. MAIDS for VoIP Genesis

In this section, we will outline the MAIDS for VoIP genealogy, and acknowledge the contributions and ideas we received from our research colleagues.

4.1.1. APHIDS Influence

APHIIDS is a mobile agent IDS framework developed by Deeter et al [163]; we refer our readers to the work cited for a thorough understanding of the APHIDS IDS. This work has been further enhanced by Deeter et al in [164].

We have built our MAIDS for VoIP on top of APHIDS's framework, and we leveraged the main concepts Deeter et al introduced: `TaskAgent`, `TriggerAgent`, and TriggerEvent from [163], and Correlation Routine from [164]. We have also received technical advice from Ken Deeter.

We differ however from both works, as none provides for a specific action to be carried out upon a detected intrusion – both works *report* the intrusion (on the SysAdmin console), but no further action is taken. Our system provides for an `ActionAgent`, which is responsible for acting upon the detection of a suspicious event.

4.1.2. BLAZE Influence

BLAZE is an intrusion detection paradigm targeting VoIP in particular, and we acknowledge the work by Singh et al [165] as providing inspiration for our IDS. BLAZE is a mobile-agent based IDS framework specifically targeting the inclusion of the VoIP protocol into its array of monitored protocols. It has a two tier architecture, with data-handling components at the lower tier, and a *policy engine* component at the upper tier. The system's "brain" is located in the Policy Engine; the latter provides a set of dynamically adjusting set of rules describing specific, VoIP attacks. In addition, Singh et al extend the concept of User Profile to the VoIP realm, and state provisions for such in the functioning of the Policy Engine.

We have leveraged the concepts introduced by Singh et al of employing mobile agent for specific VoIP activities monitoring.

We differ however in the fact that we have decided to distribute the "brain" of our system to our agents; MAIDS for VoIP has no Policy Engine. This achieves several desirable goals in IDS design: prevents the introduction of a single point of failure in our system, insures that any given entity VoIP-MAIDS agent is capable of carrying the full array of intrusion detection tasks, and – we claim – better leverages the potential of intelligent mobile agents while "moving the computation to the data" – the adagio most touted by the mobile agent paradigm.

We acknowledge here the approach guidance we received from Kapil Singh, and technical advice from Mohammed Alam.

4.1.3. Network Puzzles

The idea behind network puzzles is to attempt to validate an incoming Client request before committing the Server resources ([166], [167]), and it has been mainly employ as a defense against DoS and DDoS attacks. In particular, Aura et al advocate the fact that the Client should always commit to authenticating itself, and the Server should verify the bona fide nature of the request before resource commitment should happen at the Server.

We have adapted this approach in our MAIDS for VoIP system, by delegating the mobile agents responsible with the *action* phase of our protocol with involving the Users of the VoIP application when there is a documented suspicion of intrusion or unauthorized use.

We acknowledge here the technical advice and critical guidance we received from Kapil Singh.

4.2. MAIDS for VoIP Mission statement

We briefly state our goal in designing our MAIDS for VoIP system: *to protect the End-Users of the VoIP application.*

Indeed, our desire is not to protect the VoIP application itself; we consider that to be the responsibility of its vendor. We equally do not set out to protect the network, nor its nodes. Nonetheless, we do not claim that the aforementioned entities are not worth protecting; on the contrary.

Our approach to the IDS metaphor design is that the real injured party is invariably the End-User of a given system or application, as they are at the forefront of the interface. In too many cases, by the time all the alarm bells have been acknowledged and the SysAdmin's closed the attacked "doors", someone's identity, or credit card has been stolen, or an innocent bystander has to pay for goods or services she did not purchased.

Henceforth, our IDS is design to prevent abuse of the End-Users' credentials for illicit use, and in doing so, we require their involvement.

4.3. VoIP Implementation under Test Vulnerabilities

In this section, we expose the vulnerabilities the VoIP Implementation Under Test is prone to, as uncovered by us in a controlled laboratory environment. As we have mentioned in Section 3.2, the IUT implements the H.248 Megaco stack; its main components are the Media Gateway Controller (MGC) and the Media Gateway (MG), coupled in a master-slave relationship.

4.3.1. Laboratory Setup

4.3.1.1 Typical Setup

In Figure 8 we show a typical Megaco setup, in a generalized fashion, by implying remote geographical location of the MG, MGC, and application Users (which we nickname Bob and Peter). This is naturally a simplified schema, as in reality we expect an MG to service several Users; in like manner, an MGC can interact with more than two MG's.

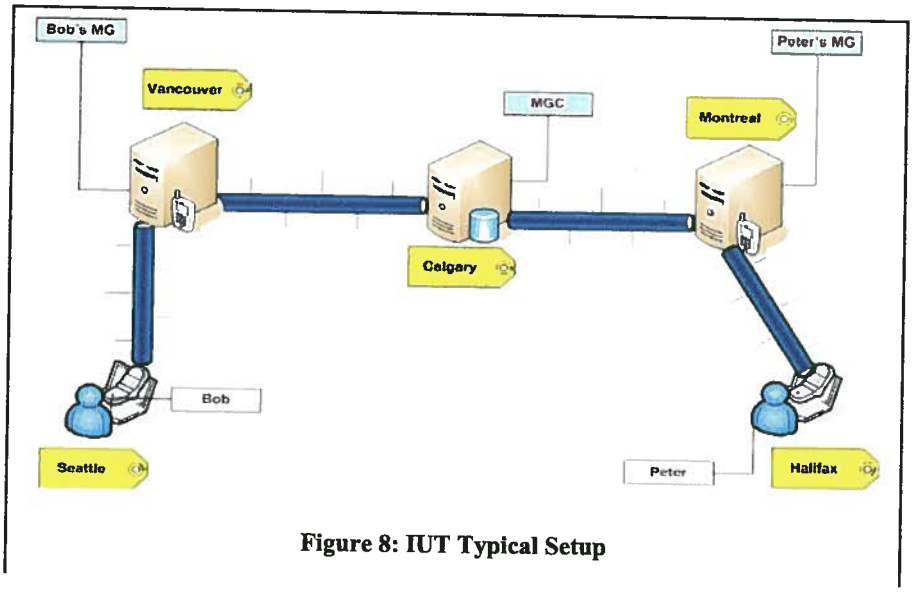


Figure 8: IUT Typical Setup

4.3.1.2 IUT Requirements

In Figure 9 we outline the IUT information tokens required for a successful handshake between the MG's and the MGC:

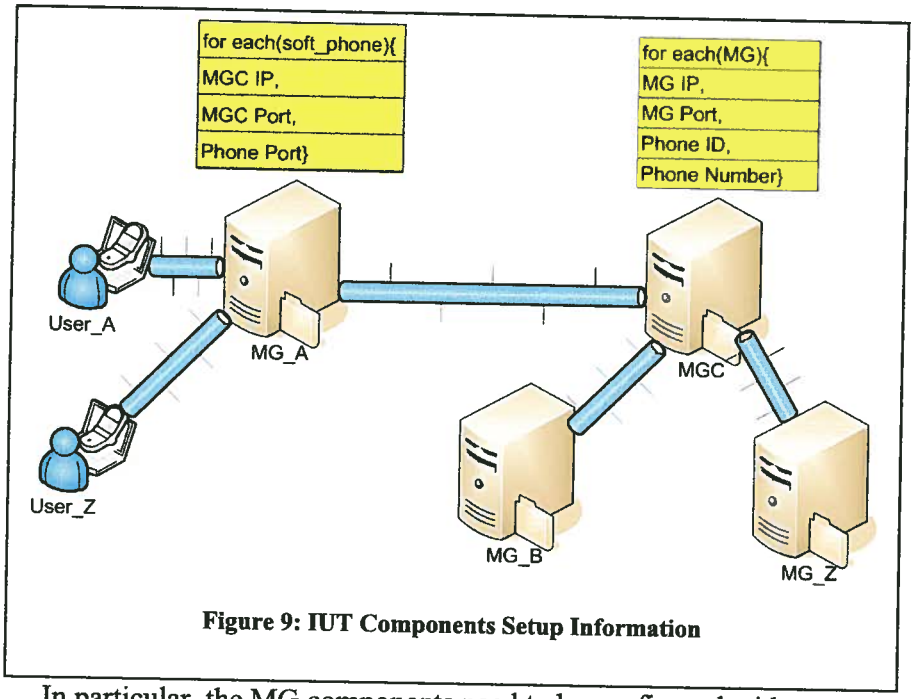


Figure 9: IUT Components Setup Information

In particular, the MG components need to be configured with:

- The IP address of the MGC
- The port where communication with MGC will occur
- The port where communication with Soft-Phone will occur

The MGC component needs to be configured with:

- The IP addresses of the MGC's it will interact with
- The ports where communication with MG's will occur
- The port where communication with Soft-Phone will occur
- A unique alphanumeric identifier for each Soft-Phone
- A unique telephone number for each Soft-Phone

4.3.2. IUT Vulnerabilities

4.3.2.1 Attack Initiation

The Attacker begins by capturing confidential setup information via a packet capture performed on the wire while the MG and MGC are exchanging information:

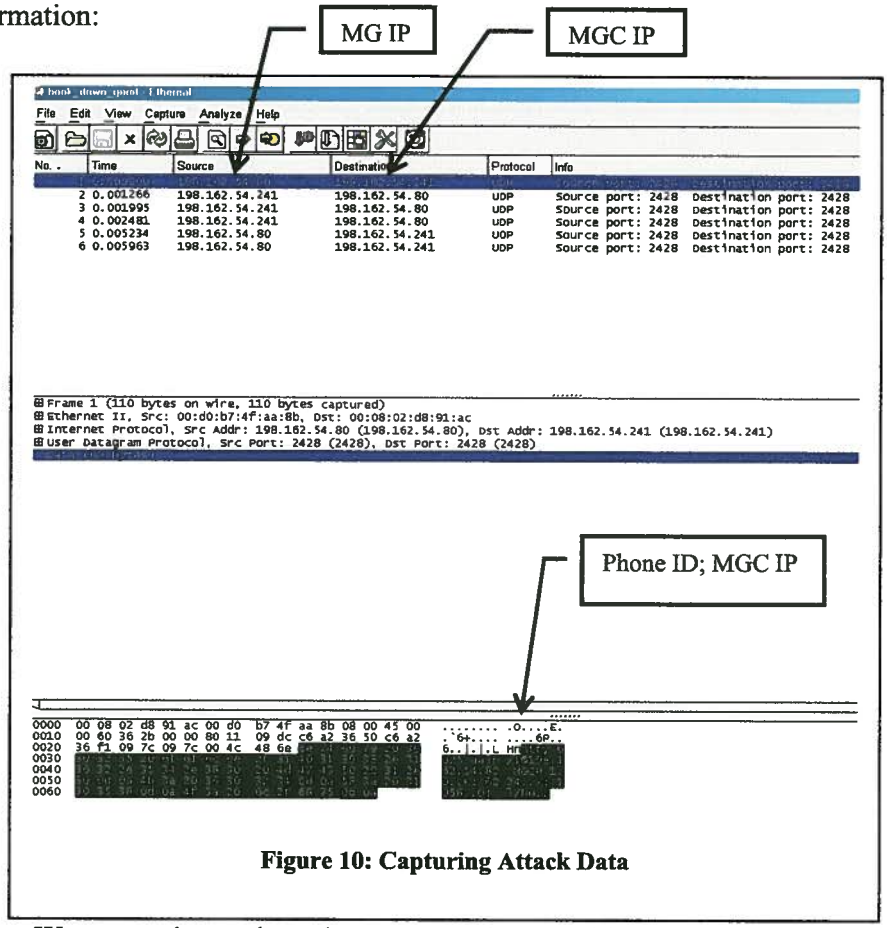


Figure 10: Capturing Attack Data

We note here that the capture is possible partly due to a poor implementation of the Interim AH mechanism in the Megaco/H.248 protocol, according to the standard itself [156]. The tokens of interest here are:

- The ID and Port Number of a(ny) "talking" Soft-Phone
- The IP address of the MGC

4.3.2.2 Service Teardown Attack

The stolen credentials are used to force a Soft-Phone *re-registration* event at the MGC; this is shown in Figure 11:

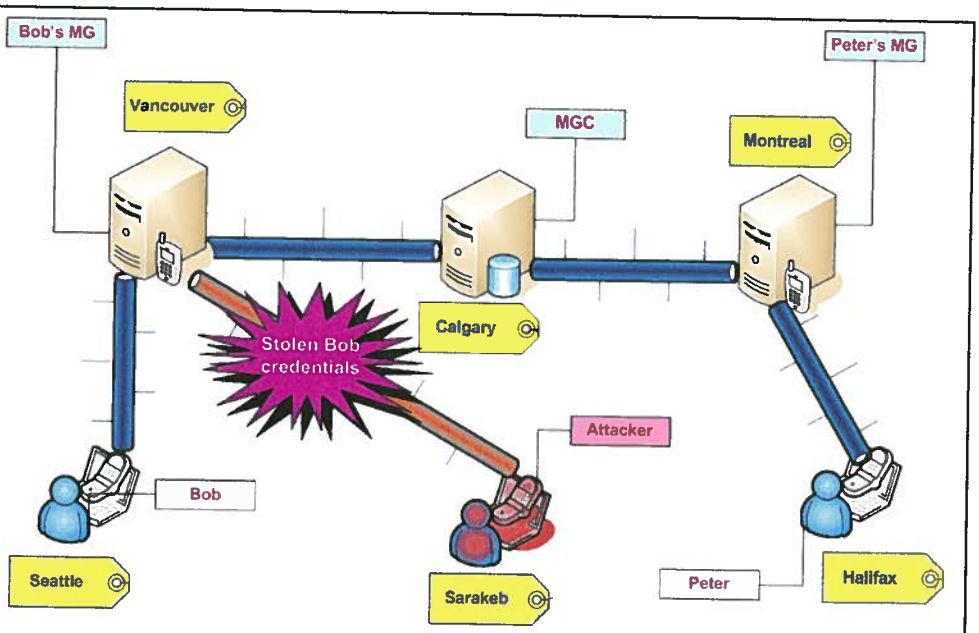


Figure 11: Service Teardown Attack

The attack succeeds, and results in:

- Peter getting aural feedback of the link being cut – busy signal
- Bob getting no feedback whatsoever
- Bob is unable to use his Soft-Phone for any kind of operation

4.3.2.3 Call Hijacking and Toll Fraud via Identity Theft Attack

Since Peter receives aural feedback of the lost connection, he may very well attempt to call Bob again; this will result in the Attacker receiving the call. Also, for any calls the Attacker will initiate he will be impersonating Bob; this results in Bob account being charged for the calls. This is shown in Figure 12:

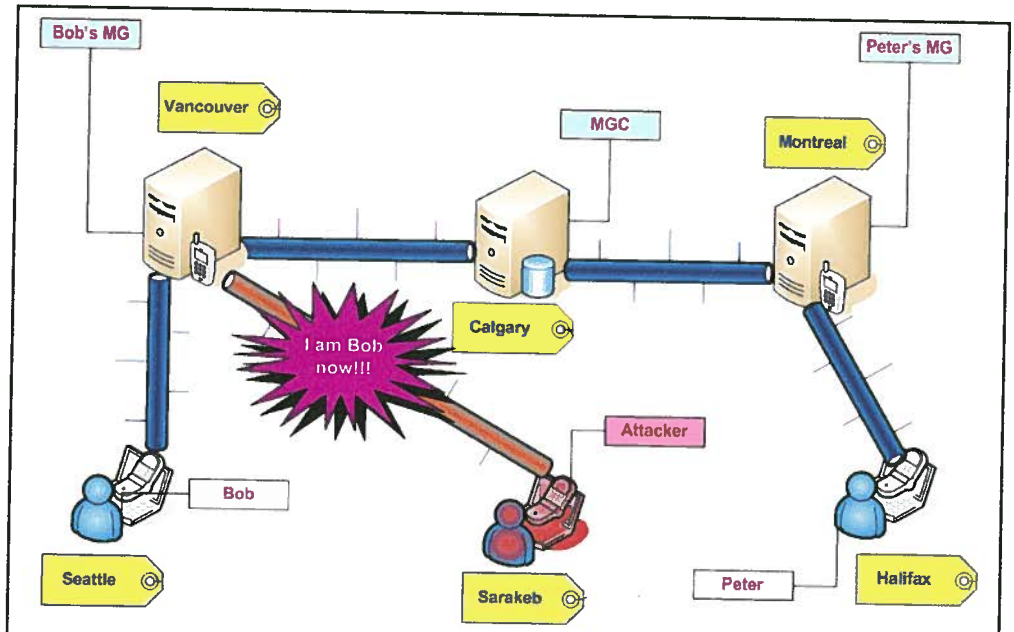


Figure 12: Call Hijacking Attack

4.3.2.4 Denial of Service Attack

The Attacker can now reiterate the same attack at Peter's MG, or at virtually any other IUT-implemented MG, denying service to all legitimate Users for which the required tokens tuple {phone_ID, phone_Port, MG_IP} can be stolen (Figure 13):

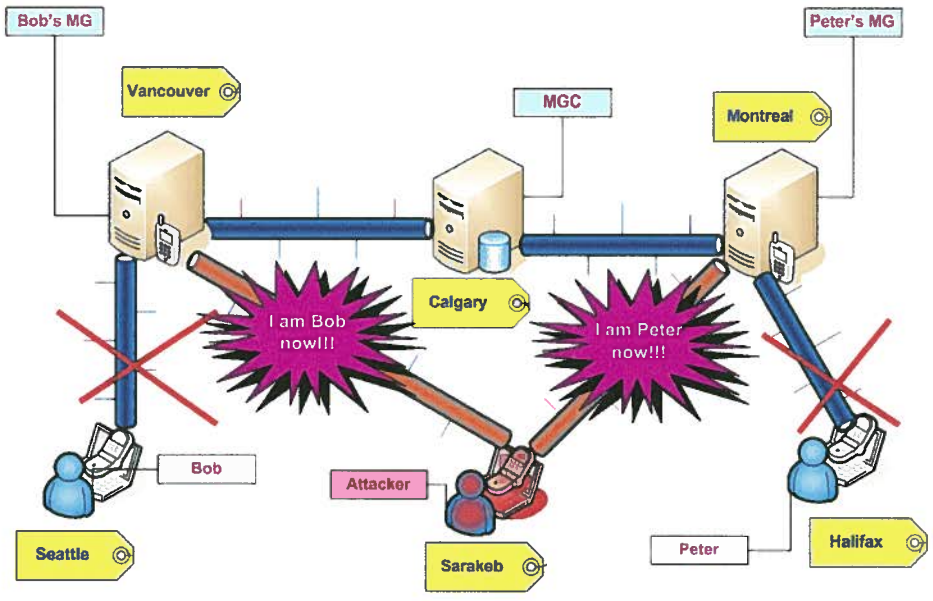


Figure 13: DOS Attack

4.4. MAIDS for VoIP Algorithm and State Machine

In this section, we will outline the MAIDS for VoIP algorithm and present the state machine the IDS continuously updates as End-Users connect and disconnect their Soft-Phones; the security problems we revealed in Section 4.3 have been instrumental in shaping our IDS algorithm.

4.4.1. MAIDS for VoIP Algorithm

4.4.1.1 Formal Algorithm

The security audit trail analysis method is well known both in the industry and academia; it mainly refers to the ability to detect attacks, and infer responses to intrusions via audit data monitoring. One important twist to it is adding temporal information to the analysis process; this has been shown not only to improve the detection rate, but also to be effective in situation where traditional statistical analysis is impractical [160]. In particular, Teng et al prove that accounting for the *sequential relationship* between audit data events permits segregating malicious from normal behavior.

This approach has been extended by Ko et al in what is currently known as *specification-based detection* [161]. The core principle revolves around vulnerability detection via a comparison between the system behavior extracted from audit data and the specification based system behavior. Ko et al extended this approach in [162], to base the system specifications on *traces*: *ordered sequences of execution events*. As Ko et al explain:

> The monitoring of execution programs involves detecting deviations of their behavior from these specifications, rather than detecting the occurrence of specific attack patterns [...]. Parsing of audit trails thus becomes the detection mechanism in a specification-based detection system; it detects operations performed by subjects that are in violation of the trace policies [...]. In some situation, it is not only the set of operations performed by a program that is of concern, but also the order of these operations [161].

Our experimental observations have firmly placed the IUT program under this latter behavioral category, and our algorithm is thus based on both a blue print of correct event sequence, and the order in which they occur.

In particular, we define the following IUT-Operation template (Figure 14) for describing a given IUT operation, where:

- *operation_name* denotes an IUT usage-related operation
- *condition_n* denotes an optional applicable condition
- *event_n* denotes the series of events which must occur in the specified sequence for the operation *operation_name* to occur

```
OPERATION::operation_name
COND (condition_1, condition_2, .... ,condition_n )      // optional
START
        Event_1
        Event_2
        .
        .
        .
        Event_n
END
```

Figure 14: IUT-Operation Algorithm Template

We further define operations and events, as per the template, as follows: Table 6 show a subset of the events VoIP-MAIDS monitors, and Table 7 a subset of the operations VoIP-MAIDS defines:

Event Abbreviation	Event Description
S_R_E	*Self_Ring_Event*: a Soft-Phone registered with the VoIP-MAIDS-protected MG receives a call
S_H_E	*Self_Hook_Event*: a Soft-Phone registered with the VoIP-MAIDS-protected MG is either lifted off, or placed back on the hook
O_H_U	*Other_Hook_Up*: a Soft-Phone registered with an un-protected MG is either lifted off the hook
O_H_D	*Other_Hook_Down*: a Soft-Phone registered with an un-protected MG is placed back on the hook
R_A	*Register_Any*: registration of a Soft-Phone with a VoIP-MAIDS-protected MG
U_I_T	*User Is Talking*

Table 6: VoIP-MAIDS Algorithm Event Abbreviation

Operation Abbreviation	Operation Description
Get_Call_Self_Hang_Up	A Soft-Phone registered with the VoIP-MAIDS-protected MG hangs up at the end of a received call
Get_Call_Other_Hang_Up	A Soft-Phone hangs up first at the end of a call with a Soft-Phone registered with the VoIP-MAIDS-protected MG
Get_Call_reRegister_Event	A VoIP-MAIDS-protected MG receives a 'reRegister' event for a an already registered Soft-Phone during an in-call
Make_Call_reRegister_Event	A VoIP-MAIDS-protected MG receives a 'reRegister' event for a an already registered Soft-Phone during an out-call

Table 7: VoIP-MAIDS Algorithm Operation Abbreviation

As an example of our algorithm's logic, we exemplify in Figure 15 the sequence of events for a normal VoIP-conversation between two parties, and in Figure 16 the case of the same sequence interleaved with an out-of-sequence Soft-Phone registration event:

```
OPERATION::Get_Call_Self_Hang_Up
COND (op_code.equals("register"), second_code.equals("new") )
START
   R_A
   S_R_E
   S_H_E
   OPERATION::User_Talking
   START
      U_I_T
   END
   S_H_E
END
```

Figure 15: Normal Conversation Event Sequence

```
OPERATION::Get_Call_Self_Hang_Up
COND (op_code.equals("register"), second_code.equals("new") )
START
   R_A
   S_R_E
   S_H_E
   OPERATION::User_Talking
   START
      U_I_T
      OPERATION::Make_Call_reRegister_Event
      COND (second_code.equals("old") )
      START
         R_A
      END
   END
END
```

Figure 16: Anomalous Conversation Event Sequence

Finally, Figure 17 shows the instantiation of a successful hijacking attack:

```
OPERATION::Get_Call_Self_Hang_Up
COND (op_code.equals("register"), second_code.equals("new") )
START
  R_A
  S_R_E
  S_H_E
  OPERATION::User_Talking
  START
    U_I_T
    OPERATION::Make_Call_reRegister_Event
    COND (second_code.equals("old") )
    START
      R_A
      OPERATION::Make_Call_Self_Hang_Up
      COND (second_code.equals("old"), third_code.equals("talking") )
      START
        S_R_E
        S_H_E
        OPERATION::User_Talking
        START
          U_I_T
        END
      END
    END
  END
END
```

Figure 17: Call Hijacking Event Sequence

4.4.1.2 Algorithm Implementation

The CERT Coordination Center was created in 1988 by DARPA[101] following the strike by the Morris worm, which resulted in disabling roughly 10% of the internet systems. Since then, its role has expanded dramatically, to include a vast array of software, network, and system security aspects [158]. In particular, the CERT CC recommends five top-levels of information system security practices: *Harden/Secure*, *Prepare*, *Detect*, *Respond*, and *Improve* [159].

For our work, the first and last phases, Harden/Secure and Improve respectively, regard the proprietary product our IDS inhabits. The other three phases however, namely Prepare, Detect, and Respond are what we propose to target within MAIDS for VoIP's algorithm.

- **Detection Phase**

The detailed steps of the detection, analysis, and action procedures are as follows:

1. `VoIPTriggerAgent` (residing on the monitored MG host) acquires read-only access to the VoIP IUT log, and places an I/O thread on it to monitor for changes in real-time.
2. `VoIPTriggerAgent` instantiate a state-machine as soon as a specific End-User has logged into the VoIP IUT system. A description of the respective states is provided in Section 4.4.2.
3. a TriggerEvent is raised whenever the state machine is in is_talking state, and a reRegister event is received, having as parameters the credentials of any of the already logged-in End-Users.

[101] Defense Advanced Research Projects Agency; US Department of Defense agency responsible for development of new technology to be used by the military.

- **Analysis Phase**

4. `VoIPTriggerAgent` makes now a call to the VoIP-MAIDS-Threaded-Client software component, with the following instructions:

```
for (caller_IP)
{
      contact(VoIP-MAIDS_Server);
      instantiate(firstCallerWarning);
}

for (original_callee_IP)
{
      contact(VoIP-MAIDS_Server);
      instantiate(firstCalleeWarning);
}
```

Figure 18: VoIP-MAIDS Analysis Phase

5. `if` any of the following VoIP IUT events occur:

   i. End-User at the Caller end-point re-dials the End-User at the – *original* – Callee end-point

   ii. End-User at the – *new* – Callee end-point makes a call (or a hook_up event is observed)

`then`, `VoIPTriggerAgent` makes a call to the (running) VoIP-MAIDS-Threaded-Client software component, with the following instructions:

```
for (caller_IP)
{
      contact(VoIP-MAIDS_Server);
      instantiate(secondCallerWarning);
}

for (original_callee_IP)
{
      contact(VoIP-MAIDS_Server);
      instantiate(passwordRequest);
      sendHome(password);
}

for (new_callee_IP)
{
      contact(VoIP-MAIDS_Server);
      instantiate(passwordRequest);
      sendHome(password);
}
```

Figure 19: VoIP-MAIDS Correlation

`otherwise`, nothing happens (i.e. VoIP-MAIDS will take no action as long as no new VoIP IUT events are observed).

- **Action Phase**

6. two scenarios are now possible:

   a. *no answer* (to the password request) is received from the End-User at the – *original* – Callee end-point,

      AND

      valid answer is received from the End-User at the – *new* – Callee end-point

   b. *valid answer* (to the password request) is received from the End-User at the – *original* – Callee end-point,

      AND

      no answer, OR *erroneous answer* is received from the End-User at the – *new* – Callee end-point

For the former case, no action is required → signifies original End-User moved to another host while a call was in progress at the original location.

For the latter case, the following action is taken → signifies Attacker is present at the *new* Callee end-point, and has registered with stolen *original* Callee end-point credentials:

i. the IP address of the *new* Callee end-point is inserted into a specific file maintained by the *PeerGuardian* software component. This file holds (in specific format) the IP addresses whose incoming traffic will be blocked for any future incoming-attempts by the running *PeerGuardian* process.

ii. the End-User at the – *original* – Callee end-point is prompted to reRegister his soft-phone with the MG:

```
for (protected_MG)
{
      blocked_IP_List.insert(new_callee_IP);
      instantiate(firewallRestartRequest);
}

for (original_callee_IP)
{
      contact(VoIP-MAIDS_Server);
      instantiate(reRegisterRequest);
}
```

Figure 20: VoIP-MAIDS Action Phase

4.4.2. MAIDS for VoIP State Machine

The State Transition Analysis technique developed by Ilgun et al [34] affords detection of system intrusions *in real time*. This is achieved by representing the protected system as a state transition diagram, and updating the state machine as audit data is processed. A main advantage of this technique is the fact that it permits state transitions only when a key event occurs, thus greatly simplifying IDS's data mining task. In particular, in defining a system penetration via State Transition Analysis, Ilgun et al infer:

> State transition analysis is based on the premise that all computer penetrations share two common features. First, penetrations require the attacker to posses some minimum prerequisite access to the target system. This prerequisite access may range from access to certain files, devices, telephone lines, etc. to possession of information regarding a particular security-relevant function. Second, all penetrations led to the acquisition of some previously unheld ability. [...] Whether the ability gained is unauthorized access to data, access to another user's privileges, [...] something is gained [34].

We draw on evidence, following our reasoning in Section 4.4 and the IUT vulnerabilities we have exposed in Section 4.3, that the IUT is a highly probable candidate for the application of the State Transition Analysis. We have thus applied Ilgun et al's method in developing our MAIDS for VoIP state machine. In particular, with respect to the three phases we have identified in the previous section, namely Detection, Analysis, and Action, we infer the following state machine model:

4.4.2.1 Normal Usage

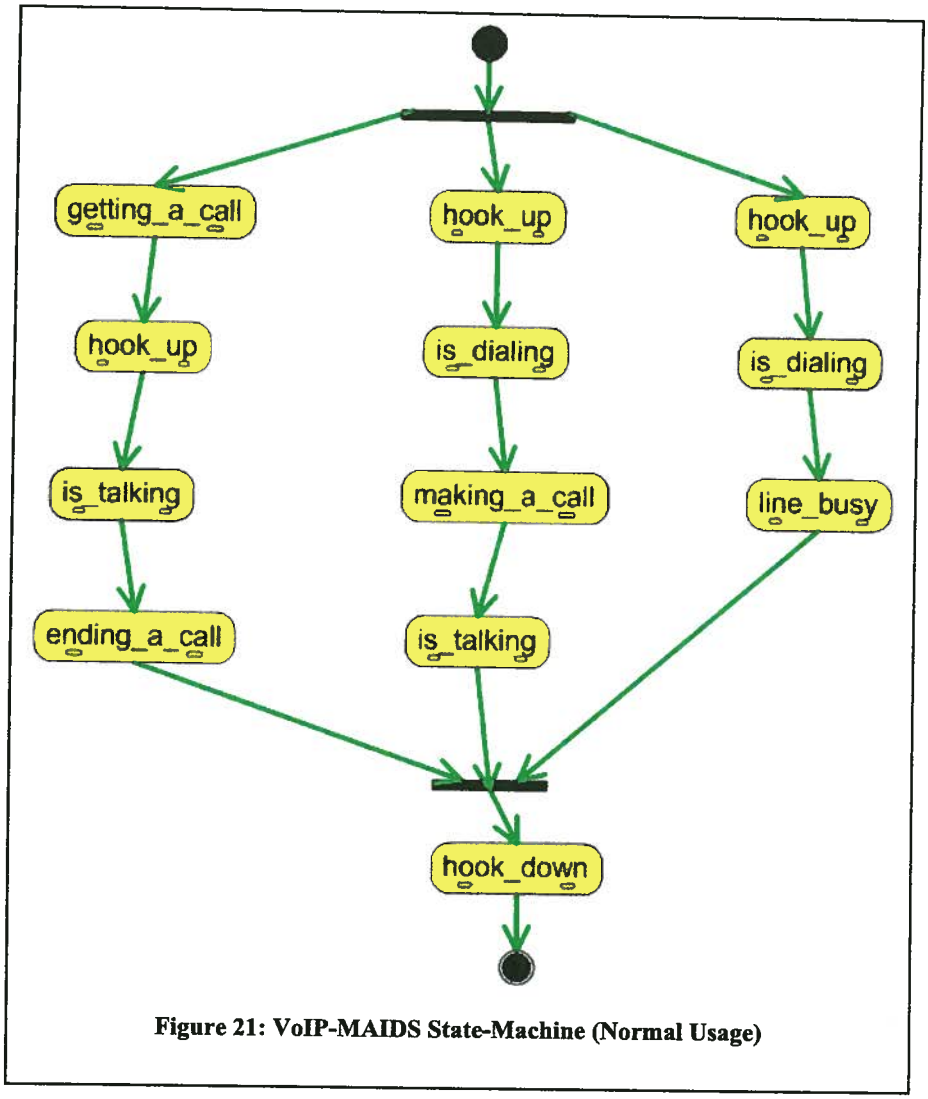


Figure 21: VoIP-MAIDS State-Machine (Normal Usage)

4.4.2.2 Anomalous Usage

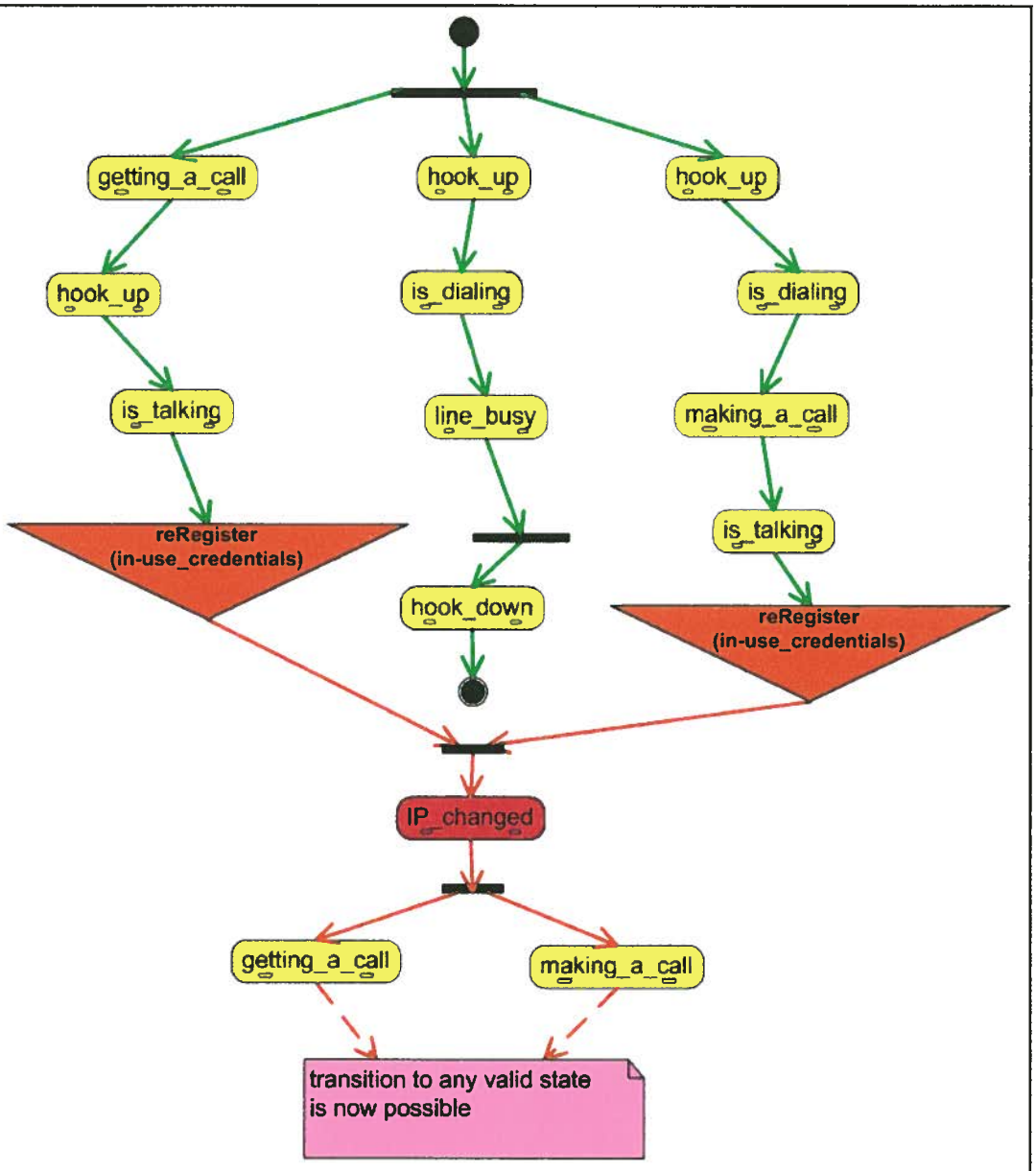


Figure 22: VoIP-MAIDS State-Machine (TriggerEvent Condition)

4.4.2.3 State Maintenance

The MAIDS for VoIP state machine is further optimized such that a reset procedure is performed periodically. This has the advantage of clearing all buffers at regular times, such that the overall memory consumption is kept low. The mechanism is implemented via a system of counters, as shown if Table 8:

R_A_count		S_R_E_count		S_H_E_count		O_H_D_count		O_H_U_count	
0	reset	0	reset	0	reset	0	reset	0	reset
1	fresh_regi	0		0		0		0	
1		1	get_IP()	0		0		0	
1		1		1	is_talking == true	0		0	
1		1		2	IIF(rsip<2)->(is_talking == false)	0		0	
unchanged | reset		0	reset	0	reset	0	reset	0	reset
unchanged | reset		1	get_IP()	0		0		0	
unchanged | reset		1		1	is_talking == true	0		0	
unchanged | reset		1		1		1		0	
unchanged | reset		1		2	IIF(rsip<2)->(is_talking == false)	1		0	
0	reset	0	reset	0	reset	0	reset	0	reset
1	fresh_regi	0		0		0		0	
1		1	get_IP()	0		0		0	
1		1		1	is_talking == true	0		0	
2	<-IIF(ntfy==1)->attack == true	1		1		0		0	
0	reset	0	reset	0	reset	0	reset	0	reset
1	fresh_regi	0		0		0		0	
1		0		1	is_talking == true	0		0	
1		1	get_IP()	1		0		0	
1		1		1		0		1	
1		1		1		0		2	
2	<-IIF(ntfy==1)->attack == true	1		1		0		2	
2		2	get_Attacker_IP()	1		0		2	
2		2		2	IIF(rsip>2)->(attacker_is_talking == true)	0		2	

Table 8: VoIP-MAIDS Counter Reset Algorithm

4.5. MAIDS for VoIP Implementation

In this section, we will outline the MAIDS for VoIP architecture, and details its implementation.

4.5.1. MAIDS for VoIP Architecture

4.4.1.1 Design Goals

In our approach to designing VoIP-MAIDS, we aimed to achieve the following goals:

- Deliver on the scalability promise characteristic to mobile agents-based systems
- Real-Time response to threats that the system already "knows" about
- Possibility to "train" the system so as to learn "normal" User behavior, and thus be able to spot abnormal system-usage patterns for a specific User

As VoIP-MAIDS aims to be a generic VoIP intrusion detection platform, it must provide meaningful abstractions and useful modular design. In addition, we identify also the following two functional requirements:

1. *Integration*: The ability to integrate existing Intrusion Detection Systems and techniques.
2. *Programmability and Automation*: The system should provide a mechanism for a VoIP-MAIDS-implementer to re-use existing components or customize existing analysis procedures. It should allow security experts to automate analysis procedures that would otherwise be performed manually.

Category	Selected Criterion for VoIP-MAIDS	
Performance	*Response Time*	The system shall afford real-time answer to incoming known threats, once it is up and running
	Throughput	The system shall be able to satisfactory answer a load of 5 incoming threads of each threat-category it knows about
Dependability	*Reliability*	Once correctly initiated and started, the system shall continue to detect incoming threats even when one, or more of its deployed mobile agents terminate abnormally (are killed by mistake or some other failure)
	Availability	Under any circumstances the system shall not shut down unless specifically instructed to do so
	Security	The system shall allow the encryption of the messages passed between the agents, in addition to secure communication channel between the agents
Maintenance	*Extensibility*	The system shall afford the addition of new threat-implementation classes without any structural change
	Adaptability	The system shall not be limited to identity theft-related threats; usage pattern analysis shall be implemented
	Portability	The system shall be available for both Linux and Windows platforms
End User	*Utility*	The system shall extend and concretely enhance the ability of the human operator to detect the nature of incoming threats

Table 9: MAIDS for VoIP Design Goals

4.4.1.2 Layers

As we have outlined in Section 4.1 (MAIDS for VoIP Genesis), our Mobile Agent platform of choice is GrassHopper, and we are leveraging the APHIDS framework. The ensuing IDS inhabits the COTS IUT, resulting in a two-tiered architecture, as shown in Figure 23:

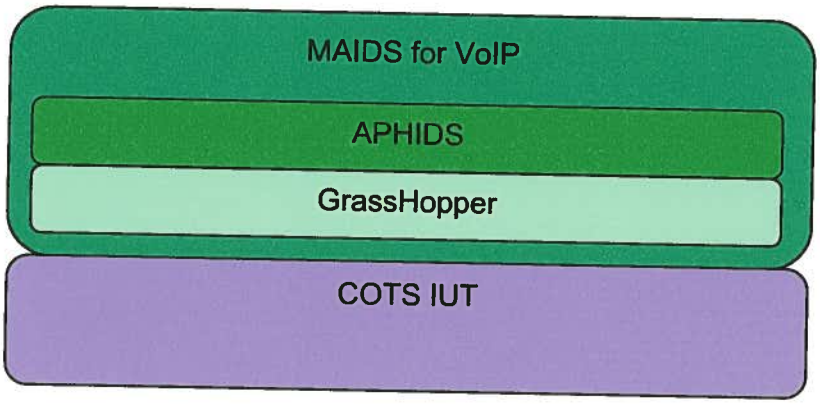


Figure 23: MAIDS for VoIP System Context Diagram

We infer thus the following participating entities:

Systems:	Grasshopper (COTS), APHIDS, Protected/Inhabited System (VoIP COTS)
Human Actors:	User, SysAdmin, Attacker

Table 10: MAIDS for VoIP Participating Entities

4.4.1.3 Use Case Diagram

Figure 24 presents the Use Case diagram of our system. We distinguish two essential Actors who need to communicate with the system before VoIP-MAIDS can function. Namely, the Grasshopper platform needs to be installed and fully operational on the protected host(s), and the System Administrator (SysAdmin) needs to start:

1. one Grasshopper Region on the host acting as central console
2. Grasshopper Agencies on all hosts where agents are expected to move at some point in the future. All the active agencies must register with the Grasshopper Region mentioned in '1' (this operation is part of the Agency start-up procedure, and is part of the SysAdmin's responsibilities).

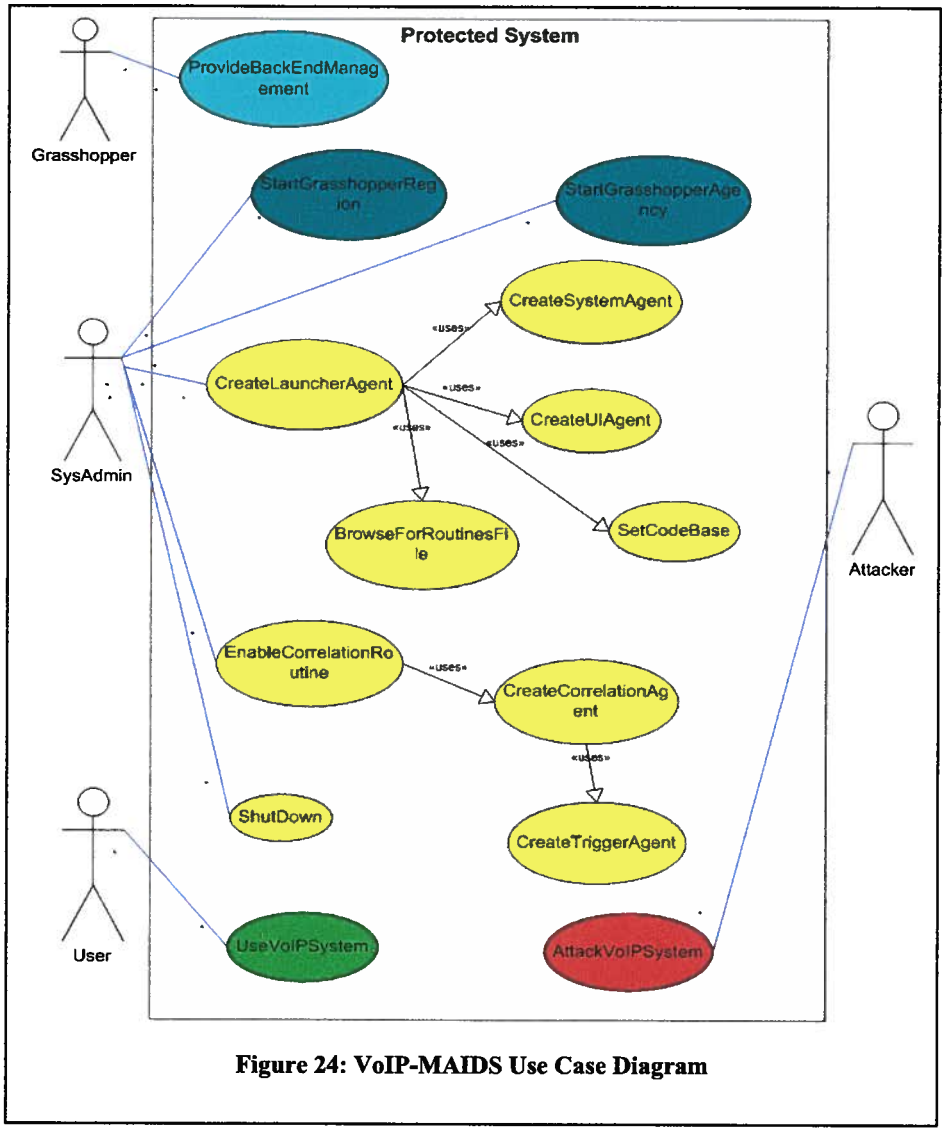


Figure 24: VoIP-MAIDS Use Case Diagram

4.4.1.4 Subsystem Decomposition

Figure 25 below presents VoIP-MAIDS's Subsystem Decomposition; we infer three main entities:

- The *Console Subsytem*: this is the component running on the IDS's console; it is responsible for the administration of VoIP-MAIDS.
- The *Mobile Agent Subsytem*: the mobile agent entities are logically grouped under this component.
- The *Client Server Subsytem*: the modules part of this subsystem are handling VoIP-MAIDS's interaction with the IUT Users.

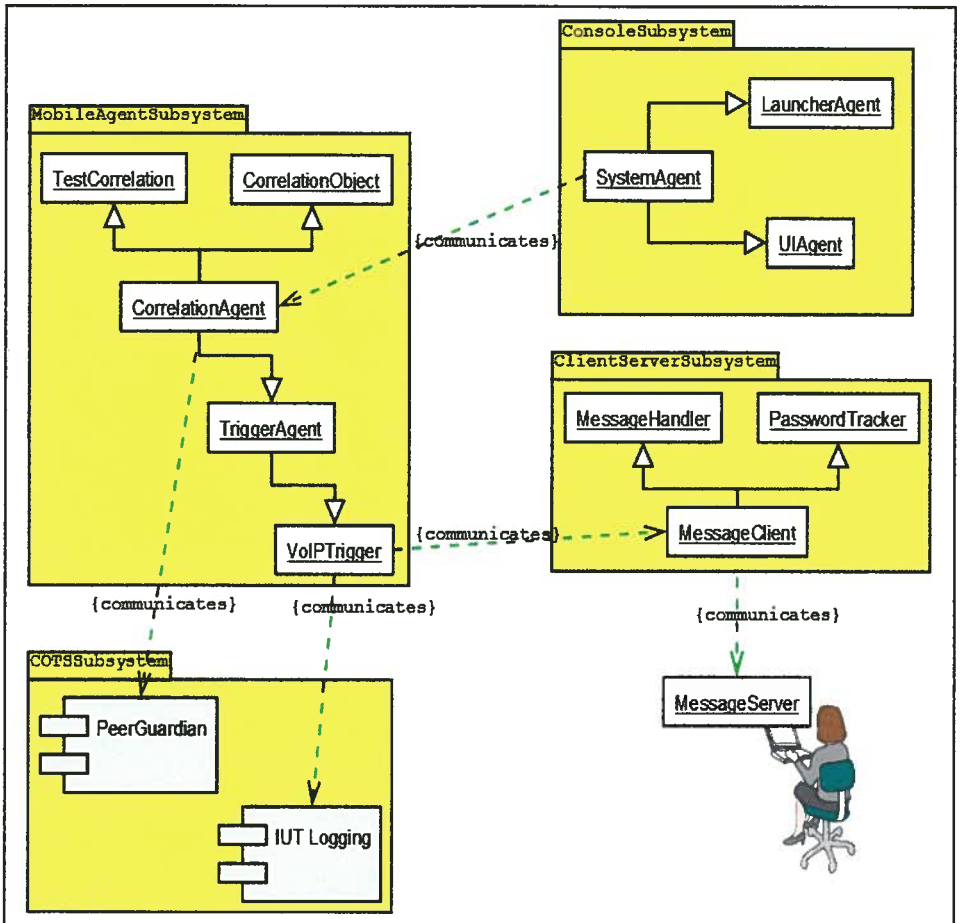


Figure 25: VoIP-MAIDS Subsystem Decomposition

4.5.2. MAIDS for VoIP Object Design

In this section we present the system's main components, and their role within VoIP-MAIDS.

The organization of agents in the MAIDS for VoIP system is guided by the goal to implement the intrusion detection process model described in Section 4.4.3. Each phase in the process model is mapped to distinct subsystems, each of which being implemented by specialized agents. While the application process model may restrict the domain of possibilities, the clear structure of the process aids in the definition and management relationship between the various cooperating agents. The following is a description of the types of agents in the system, and their functionality.

4.4.2.1 The LauncherAgent (Console Subsytem)

The role of this agent is to enable the system startup by users not familiar with Grasshopper Textual User Interface[102] syntax. Its role is to create the `SystemAgent`, and to broadcast the codebase's location. It is terminated after the `SystemAgent` is created.

4.4.2.2 The SystemAgent (Console Subsytem)

To manage the potentially large number of `CorrelationAgents` and `TriggerAgents`, a separate management agent (identified herein as `SystemAgent`) was implemented. This agent provides a high level interface to query available correlation routines and to enable and disable specific routines. The `SystemAgent` also maintains a list of references to all operating `CorrelationAgents` in the system, allowing for centralized system management functions such as system shutdown, in which all active agents must suspend their processing and free their resources.

---

[102] TUI in GrassHopper documentation.

Its role however is not to be the top of our agent system hierarchy, as this would introduce a single point of failure in our system, and we have consistently identify this as a fault in our analysis of current systems (Section 2.1).

The highest logical point in our system hierarchy is represented by the {CorrelationAgent | CorrelationObject} pair, and it affords redundancy: the abnormal termination of one pair does not influence the others.

4.4.2.3 The UIAgent (Console Subsytem)

The UIAgent is provided to present a simpler user interface on the VoIP-MAIDS console host[103]. We would like to encourage our readers to take this sentence in following context: the Grasshopper provided TUI was judged as too cryptic by our design team, whereas the Grasshopper provided GUI was deemed as too specific to the Grasshopper platform itself, and not to our system.

This agent translates SysAdmin commands into procedure calls for the SystemAgent, allowing this category of Users to interact with the system. In addition, it represents the communication vehicle between the system and the SysAdmin, as this is where messages and notifications are displayed.

4.4.2.4 The CorrelationAgent (Mobile Agent Subsytem)

Once this agent has received information regarding a TriggerEvent, it will perform any analysis task required to obtain more information regarding this event. Each CorrelationAgent is programmed with a *Correlation Routine*[104] (via an associated *CorrelationObject*) which defines its behavior when responding to a specific TriggerEvent.

---

[103] The UIAgent is a mobile agent itself, and uses only standard inter-agent communication mechanisms to communicate with the SystemAgent, thus allowing for a mobile GUI in the future.

[104] As per APHIDS's framework.

4.4.2.5 The {CorrelationRoutine | CorrelationObject} Pair (Mobile Agent Subsytem)

Abstractly, a *Correlation Routine* refers to the list of analysis and search procedures that the MAIDS for VoIP system needs to perform in the correlation phase when responding to a particular detected event. A CorrelationRoutine is described using an active object: a CorrelationObject, which provides functions that implement the analysis logic. When a `CorrelationAgent` is initialized, the name of the corresponding CorrelationObject is passed as a parameter. The `CorrelationAgent` then instantiates this object and uses it to perform correlation procedures.

A single correlation routine cannot possibly handle all types of trigger events. To reduce the complexity of the correlation routine, each CorrelationObject is responsible for the deployment of its own `TriggerAgent`. This mechanism allows the correlation routine to control exactly which types of events it will receive, and implement the appropriate analysis logic for those limited sets of events. In practice, this results in one correlation and `TriggerAgent` pair for MAIDS's particular VoIP correlation task.

4.4.2.6 The TriggerAgent (Mobile Agent Subsytem)

The detection, analysis, and action phases are all implemented by the `TriggerAgent`. This represents a significant departure from the APHIDS implementation metaphor, and we would like to dedicate the next paragraph to an explanation of this fact.

The main reason behind having the `TriggerAgent` responsible for the entire chain of VoIP-MAIDS events temporally following a TriggerEvent (the event that is reported by a `TriggerAgent` is a *Trigger Event*) stems from the following facts:

a) No other data mining task is required at other hosts (than the monitored one)

b) The VoIP-MAIDS-Threaded-Client (software component in charge with contacting VoIP-MAIDS-End-Point-Servers residing at the End-User's Soft-Phones, resides at the monitored MG host (i.e. same host as the `TriggerAgent`'s location)

c) The list of blocked IP addresses (where any Attacker IP address needs to be written) resides at the monitored MG host (i.e. same host as the `TriggerAgent`'s location)

d) The state-machine keeping track of the End-User's actions is maintained by the `TriggerAgent`, at the monitored MG host (i.e. same host as the `TriggerAgent`'s location)

4.4.2.7 The VoIPTriggerAgent (Mobile Agent Subsytem)

The role of the `VoIPTriggerAgent` is to:

a. Maintain an active-End-User state-machine, providing VoIP-MAIDS with runtime knowledge of the End-User's actions

b. monitor for any suspicious behavior, which in VoIP-MAIDS's operational context, amounts to:

> *Detecting otherwise legal events, but occurring during an incompatible state of the End-User state-machine.*

c. Upon detection of suspicious behavior, instantiate dialog with the End-Users (via the VoIP-MAIDS-Threaded-Client – VoIP-MAIDS-Servers pair) so as to infer a causal conclusion

d. Should an intrusion be detected, insert the IP address of the Attacker host into the required file of the protected MG's firewall

From a general, VoIP-MAIDS-system usage perspective, we would specify that `TriggerAgents` can be written to detect any type of desired event, and as such, a TriggerEvent only refers abstractly to the event reported by a `TriggerAgent`. We specify however that a `TriggerAgent` is a specialized agent once instantiated – i.e. it can only detect the event(s) it was programmed for.

4.4.2.8 The {MesageClient | MessageServer} Pair (Client Server Subsytem)

The role of these two (non mobile agent) components is to enable VoIP-MAIDS End-User communication, as per the system's mission statement (outlined in Section 4.2). In particular the *Threaded-Client* is a component running at every MG host location, whereas the End-Point-Server is a component running on the VoIP-MAIDS-protected Soft-Phone hosts. Their placement is as follows:

- the *Threaded-Client* resides at the protected MG location, and instantiates a thread for every Soft-Phone registering at the protected MG.
- the *End-Point-Server* resides at the Soft-Phone location, and is listening for instructions from its associated Threaded-Client working thread.

Their role is as follows:

- Upon VoIP-MAIDS's state machine entering a *suspicious* state, specific Threaded-Client threads instruct their corresponding End-Point-Servers to display a warning on the End-User's desktops.
- Upon VoIP-MAIDS's state machine entering an *alarm* state, specific Threaded-Client threads instruct their corresponding End-Point-Servers to display a password request on the End-User's desktops.

- Success, failure, or timeout for the above operations are reported to `TriggerAgent`.

4.5.3. MAIDS for VoIP Functional Design

4.4.2.1 Intrusion Detection Model

Given the design goals outlined in Section 4.4.1, VoIP-MAIDS was seen as implementing the *Model/View/Controller* architecture, via a three-layer approach. The main motivation behind this design decision was to maximize future enhancements and extensions of the IDS.

The process model used for VoIP-MAIDS defines thus three distinct phases, with a one-way progression through each phase. The three phases are as follows:

1. *Detection*: This phase is responsible for the monitoring and detection of attacks (incoming threats). This can involve monitoring for simple single events, or employing mechanisms to detect patterns of events.
   In this phase, the active agents are:
   - `SystemAgent`
   - `UIAgent`
   - `CorrelationAgent(s)`
   - `TriggerAgent(s)`

   Though they are all mobile agents, the only agent(s) spatially-away from the (VoIP-MAIDS) console is the `TriggerAgent` (more detail is provided in Section 4.5)
2. *Evidence Gathering and Analysis*: This step collects information regarding an attack inferred in the *Detection* phase. This includes searching for, and correlating distributed data captured by different monitoring systems,

and/or at different monitored hosts.

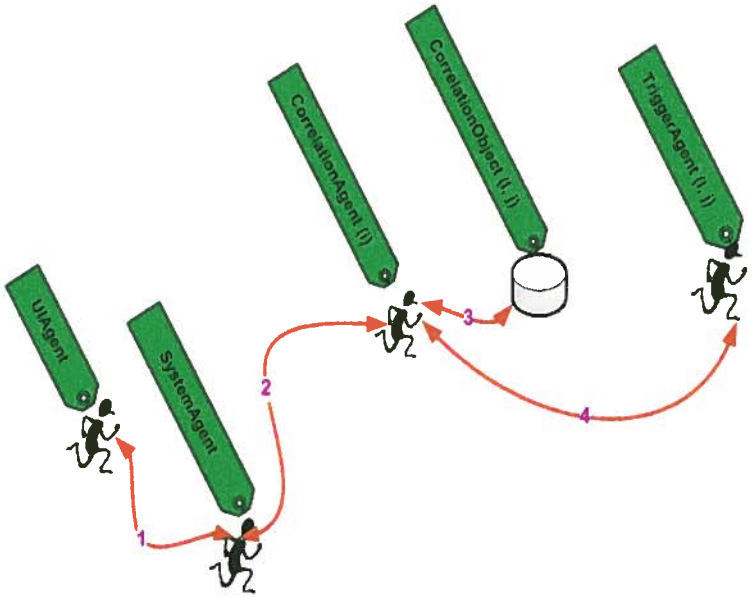
In this phase, the active agents are those specified in '1' (more detail is provided in Section 4.5).

3. *Action*: Once analysis is complete, an action needs to be taken (e.g. inform the End-User, raise an alarm with the SysAdmin for the detected threat). This would typically communicate correlation and analysis results to the SysAdmin, but for more complex cases, it will require the currently logged End-Users to authenticate themselves. In addition, the system can also use the results to intelligently re-configure itself such that at the next occurrence of the same (specific) event, no alarm will be raised. For example, it will recognize a certain range of IP's as belonging to a legally registered End-User.

   In this phase, the active agents are those specified in '1' (more detail is provided in Section 3.3.1).

4.4.2.2 Intrusion Detection Scalability

The currently implemented attack scenario exemplifies the case where VoIP-MAIDS deals with only one incoming threat, and finishes dealing with it before the next one comes. In Figure 9, we attempt a generalization, where we try to exemplify the case where the system deals with several ongoing threats.



i = number of CorrelationRoutines the system knows about (in our current setup, i = 1), as in our convention, all attacks originating from one Attacker host have a multiplicity of 1.

For each (i)
{
    1 <= j <= 5
}

That is, the system is able to deal with 5 incoming threats, originating at 5 distinct Attacker hosts, by spawning 5 {CorrelationObjects | TriggerAgents} pairs

Figure 26: VoIP-MAIDS Agents Creation, Interaction, and Multiplicities in the Case of Multiple Incoming Threats

4.4.2.3 Sequence Diagram

VoIP-MAIDS's sequence diagram is shown in Figure 27; a detail explanation of the detection algorithm is presented in Section 4.3.

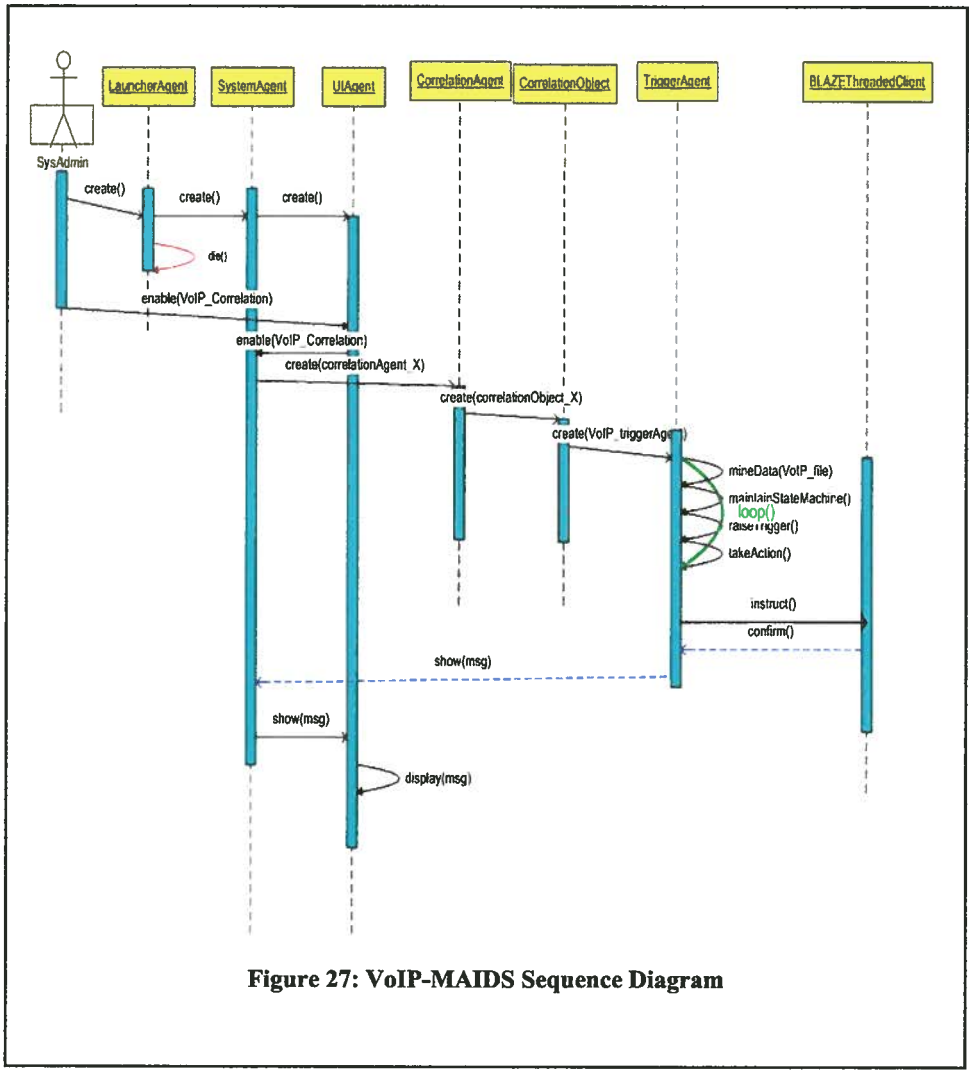


Figure 27: VoIP-MAIDS Sequence Diagram

4.6. MAIDS for VoIP Evaluation

4.6.1. Evaluation Overview

Quantitative evaluation of intrusion detection systems remains a difficult problem as confirmed by the numerous approaches and critiques described in the specialized literature; we have presented a survey in Section 2.1. Furthermore, evaluating mobile agent-based systems proves to be an even more intricate task; one example where a definite conclusion between the superiority or inferiority of mobile agent solutions versus classic client server applications could not be inferred is the work by Gray et al [157].

Our fundamental difficultly rose from the impossibility of defining a standard test scenario, as network environments can vary in many dimensions, and enterprise level COTS components have complex configuration requirements. In particular for the MAIDS for VoIP project, we experienced difficulty in inferring proper (i.e. Megaco/H.248 standard-conforming) usage of the VoIP IUT, despite support from its vendor.

Consequently, the measurements performed for this research were limited to those that could be performed on a IUT deployment as inferred from the available VoIP IUT documentation, and in this respect, they are accurate.

4.6.1.1 Assumptions

To verify proper functioning of the MAIDS for VoIP prototype implementation, VoIP IUT deployment was performed as described hereafter, and depicted in Figure 4.12.

Due to hardware laboratory limitations, the two calling parties were hosted at the two MG's, and the VoIP-MAIDS-console also proxy-ed for the Attacker host.

The following assumptions were held throughout the testing phase:

i. The Attacker has physical access to the Callee's wire

ii. The Attacker is running a standard VoIP IUT implementation (i.e. same as the one utilized by the two legal End-Users)

iii. The Attacker knows how to set-up and utilize a (network) packet-capturing utility

iv. The Attacker has approximate knowledge of the parameters required for identity theft (i.e. he can filter out noise from the captured packets)

4.6.1.2 Scenarios and Outcome

Qualitative MAIDS for VoIP evaluation was performed via a two-phase scenario, as follows:

a. With MAIDS for VoIP turned off, perform:
   i. `serviceTearDown` attack
   ii. `callHijacking` attack
b. With MAIDS for VoIP turned on, repeat the above and observe outcome

MAIDS for VoIP performed as specified (and detailed hereafter), and this concluded the empirical analysis of this particular iteration of the MAIDS for VoIP project.

4.6.2. Test Environment

For development and experimentation purposes, a laboratory cluster was created to simulate a plausible deployment scenario. A graphical description of this cluster is provided in Figure 4.12.

4.6.2.1 Hardware Specifications

The hardware specifications for these hosts are summarized in Table 11. All hosts except the (VoIP-MAIDS) console system run the Windows 2000 Professional operating system. The console host runs the Windows XP Professional Operating System. The Sun Java 2 Standard Edition SDK version 1.4.2., and the Grasshopper Mobile Agent Environment version 2.2.4 is installed on all hosts.

(internal) HostName	IP addr	CPU	Memory	Disk	Role
drax.cs.ubc.ca	198.162.54.240 (internal NIC)	Intel Pentium M, 1.5 GHz	512 MB	60 Gb	VoIP-MAIDS Console, Grasshopper Region, Grasshopper Agency, Apache webserver, Attacker
ursus.cs.ubc.ca	198.162.54.241	Intel Pentium 4 M, 2.2 GHz	261.5 MB	20 Gb + 17 Gb	Protected MG, Grasshopper Agency,
goodearth.cs.ubc.ca	192.168.54.80	X86 Family 6 Model 8 Stepping 3 800 MHz	130.5 MB	9.0 Gb + 9.0 Gb	MGC
wiseman.cs.ubc.ca	192.168.54.81	X86 Family 6 Model 8 Stepping 3 800 MHz	130.5 MB	19 Gb	MG

Table 11: Summary of Hardware Specifications for the VoIP-MAIDS Laboratory Setup

The laboratory set-up is presented in Figure 28. It introduces, in addition to the hardware nodes, the required Grasshopper components enabling our system to run.

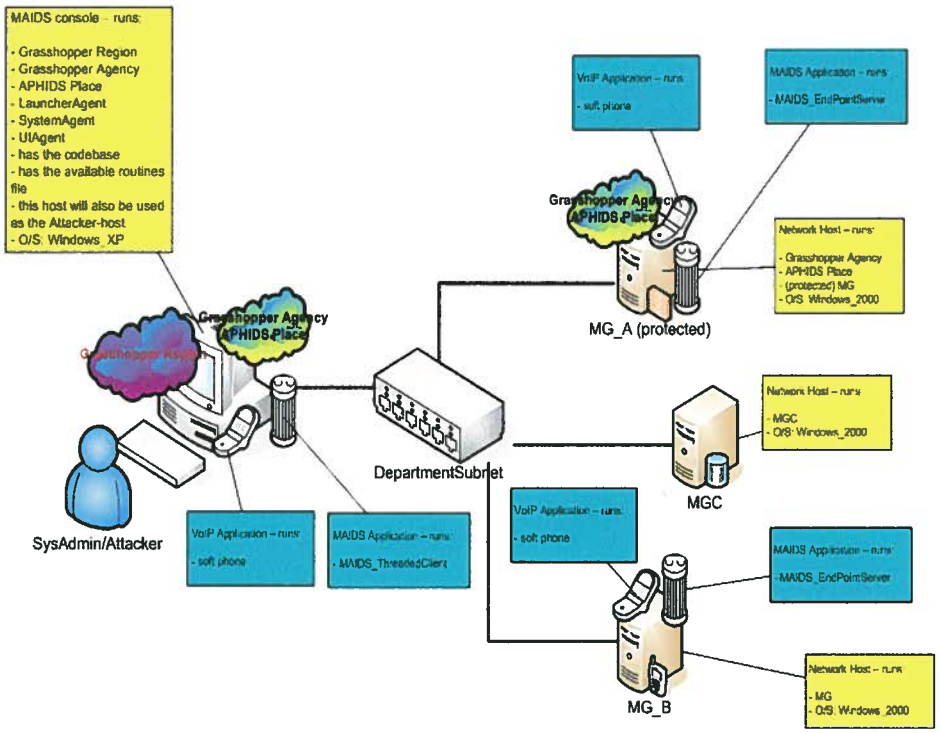


Figure 28: VoIP-MAIDS Laboratory Setup

4.6.2.2 System Initialization

Next, we introduce the sequence of steps the MAIDS for VoIP system goes trough from its initialization until action is taken following a detected incoming threat; we will exemplify each step with a supporting figure.

1. First, the VoIP IUT needs to be started. As such, we distinguish the following steps:
    a. Start the MGC (Media Gateway Controller) process on the MGC host with the following command, launched from the local IUT directory:
    ~><mgc_executable> <mgc_port>
    b. Start the MG (Media Gateway) process on the *un-protected* MG host with the following command, launched from the local IUT directory:
    ~><mg_executable> <mgc_IP> <mgc_port> <mg_IP> <mg_port> <ep_port>
    c. Start the MG (Media Gateway) process on the *protected* MG host with the following command, launched from the local IUT directory:
    ~><mg_executable> <mgc_IP> <mgc_port> <mg_IP> <mg_port> <ep_port>

2. The SysAdmin starts GrasshopperRegion on the VoIP-MAIDS console, as well as GrasshopperAgencies on all monitored hosts (*Note:* we only protect one MG host during trials – 'MG_A'). The profile used for each Agency has provisions for a VoIP-MAIDS (Grasshopper) Place – Figure 29.

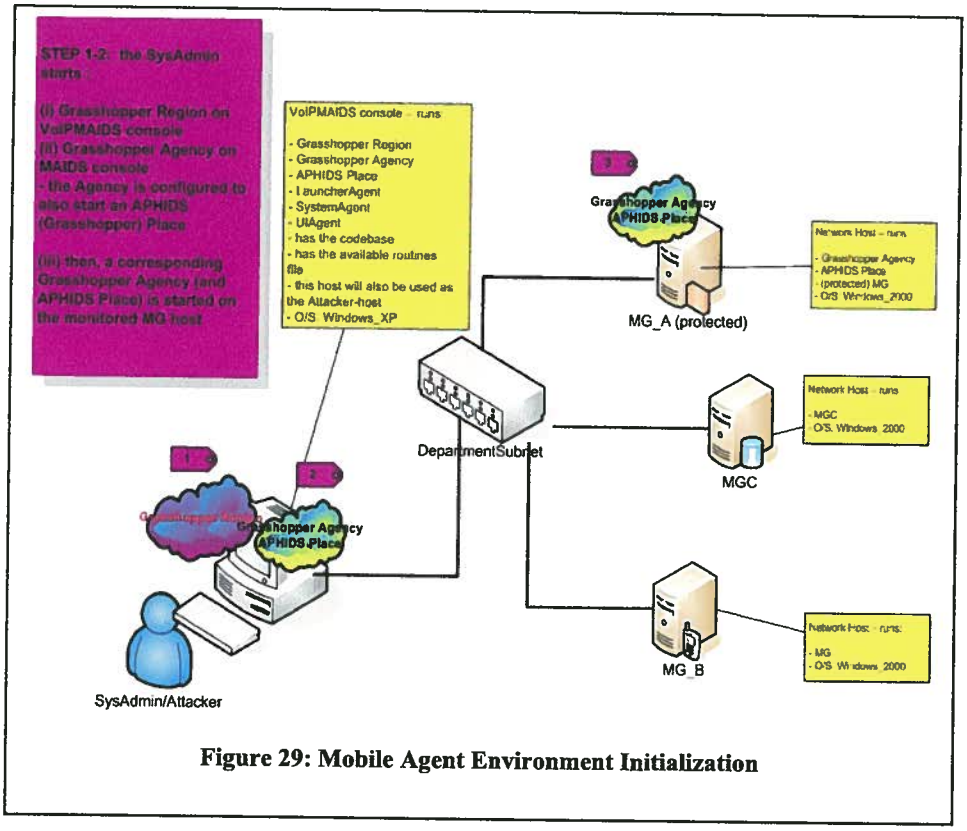


Figure 29: Mobile Agent Environment Initialization

3. Upon the SysAdmin initializing the environment as portrayed in Figure 29 above, the SysAdmin will instantiate the `LauncherAgent` using either the Grasshopper GUI (Graphic User Interface) or TUI (Text User Interface)[105]. As a result, the system's codebase is set to the 'www' accessible directory of the VoIP-MAIDS console, and `SystemAgent` and `UIAgent` are instantiated. `LauncherAgent` is terminated afterwards.

[105] For best results, TUI is recommended.

The SystemAgent creates the UIAgent in an asynchronous[106] manner. This is done so because all the commands are typed by the SysAdmin on the widget provided by the UIAgent, which then informs the SystemAgent to act on the SysAdmin-given commands – Figure 30.

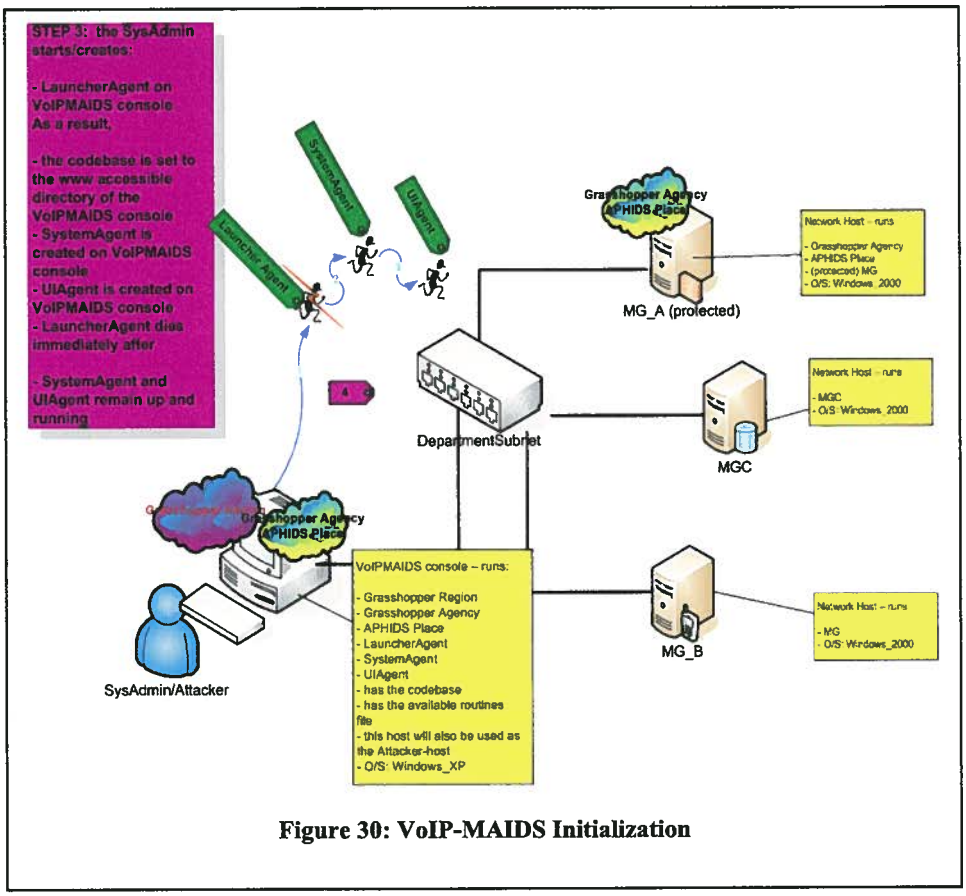


Figure 30: VoIP-MAIDS Initialization

[106] Means that the SystemAgent creates the UIAgent but does not wait for the creation of the agent to be complete. This has the advantage that the SystemAgent could do other initialization tasks while the creation of UIAgent is taking place

4. Once the SystemAgent and UIAgent are up and running, the VoIP-MAIDS system waits for SysAdmin input through the UIAgent command window.

5. At this point in the iteration, both the VoIP IUT and VoIP-MAIDS required environments are setup and ready to function, but no protection is activated until the SysAdmin types an activation command – the '*enable aphids.correlation.test.TestVoIPCorrelation*' (i.e. the *CorrelationRoutine*[107] of interest) command[108]. Upon doing so, the UIAgent passes this command to the SystemAgent. The SystemAgent in turn spawns the CorrelationAgent. The latter figures out which TriggerAgent to deploy depending upon the CorrelationRoutine parameter passed in by the SysAdmin, according to which Routine she wants to enable (in VoIP-MAIDS's case, it will be the VoIP TriggerAgent). This information is stored in the *correlationObject* class.

6. The CorrelationAgent then spawns the relevant TriggerAgent, and deploys it to the relevant VoIP IUT MG (in our set-up, this is the host identified as MG_A). The TriggerAgent then waits for a TriggerEvent to occur[109] - Figure 31.

[107] The CorrelationRoutine name is the same name as the CorrelationObject name.
[108] This step is required for testing purposes only.
[109] At this moment, the agents which are active are the UIAgent, SystemAgent, and all the TriggerAgents and the corresponding CorrelationAgents.

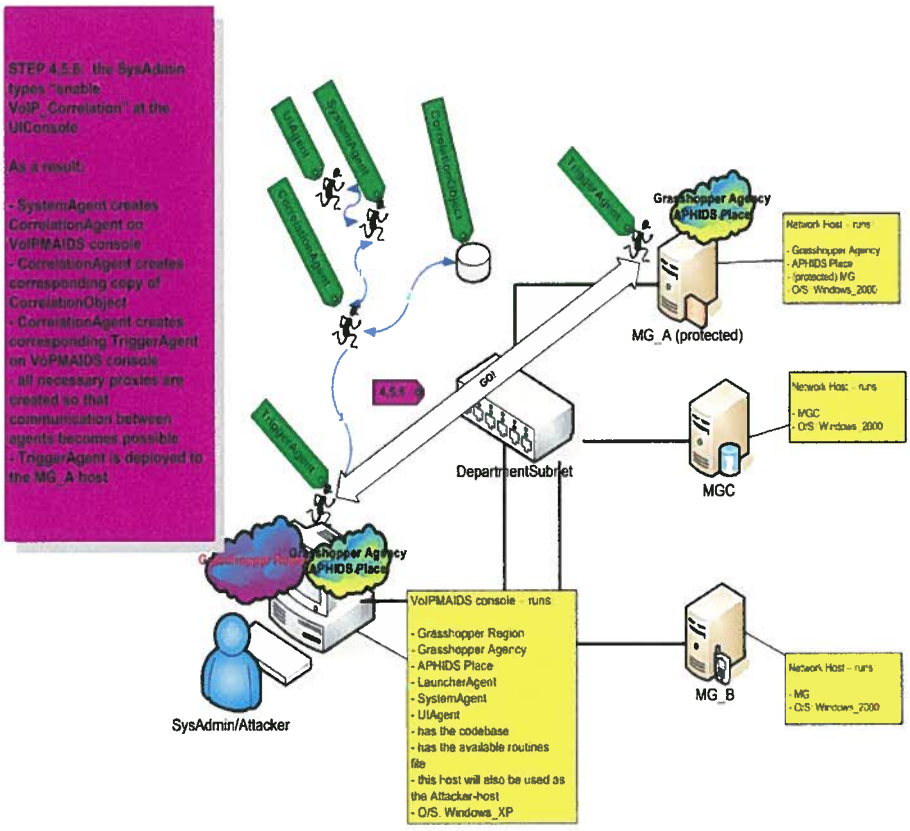


Figure 31: Enabling VoIP Correlation, and Deploying the Relevant TriggerAgent

4.6.2.3 Attack Behavior

7. At this point in time, it is assumed End-Users are making normal usage of the VoIP IUT application – we use 'Bob' and 'Peter' for illustration purposes, and they are assumed to be the currently logged-in VoIP IUT application End-Users.

    a. As soon as (any) End-User commences using the VoIP IUT (by connecting at the monitored host), `TriggerAgent` instantiates a *state machine* which monitors the End-User's actions.

    b. A TriggerEvent occurs when the state machine is in 'user_is_talking' mode, and a `reRegisterEvent` is received. The origin of such an event cannot be the monitored MG host currently employed by the User, as the VoIP IUT disallows it. However, a different host does not automatically imply an Attacker either.

    Note: for clarity of illustration purposes, we show a separate host for the Attacker, though in our laboratory setup, the VoIP-MAIDS console acts as the Attacker host as well. In like manner, Bob and Peter do not utilize separate hosts (than their respective MG's) in our laboratory setup.

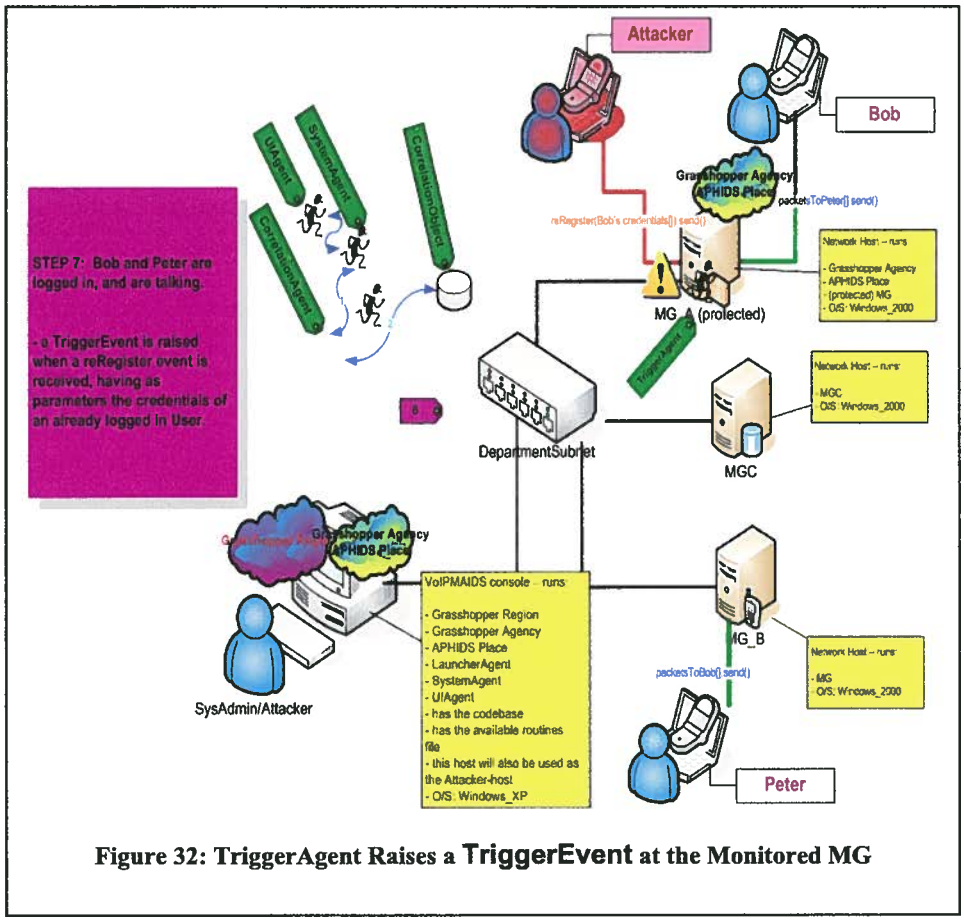


Figure 32: TriggerAgent Raises a TriggerEvent at the Monitored MG

4.6.2.3 Defense Behavior

8. Once a TriggerEvent occurs, `VoIPTriggerAgent` instructs the VoIP-MAIDS-servers running on the End-User's machines to instantiate corresponding warnings on the computer-screens of the two known-to-be-connected parties.

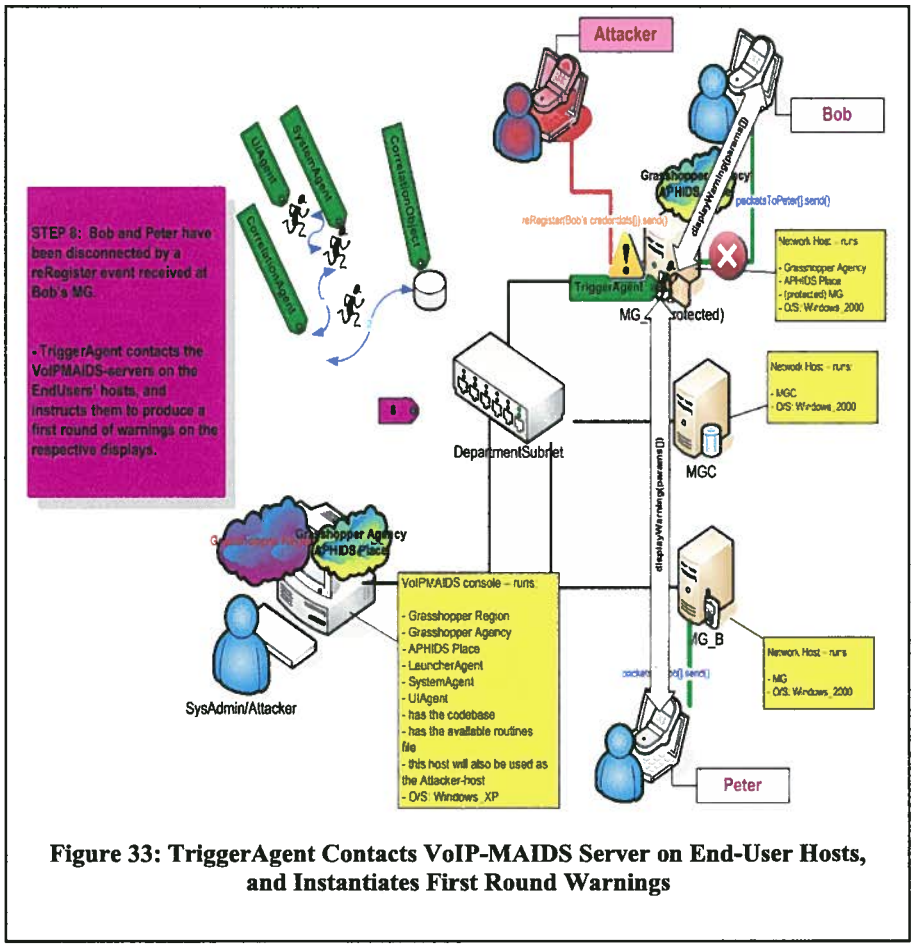


Figure 33: TriggerAgent Contacts VoIP-MAIDS Server on End-User Hosts, and Instantiates First Round Warnings

9. Since the Caller (User 'Peter' in our illustrations) is the only party to receive (aural) feedback indicating a disconnected link, he is expected to attempt reconnection with Callee (User 'Bob' in our illustrations).
    a. This results in the – supposedly – Attacker host receiving the call.
    b. Since the Attacker is now a registered party with the protected MG (albeit with Bob's credentials), VoIP-MAIDS is able to detect this.
    c. As a result:
        i. VoIP-MAIDS-server on Caller host is contacted, and instructed to produce a second round of warnings.
        ii. VoIP-MAIDS-server on – known – Callee host, as well as VoIP-MAIDS-server on Attacker host are contacted, and instructed to produce password-requests on the respective displays.

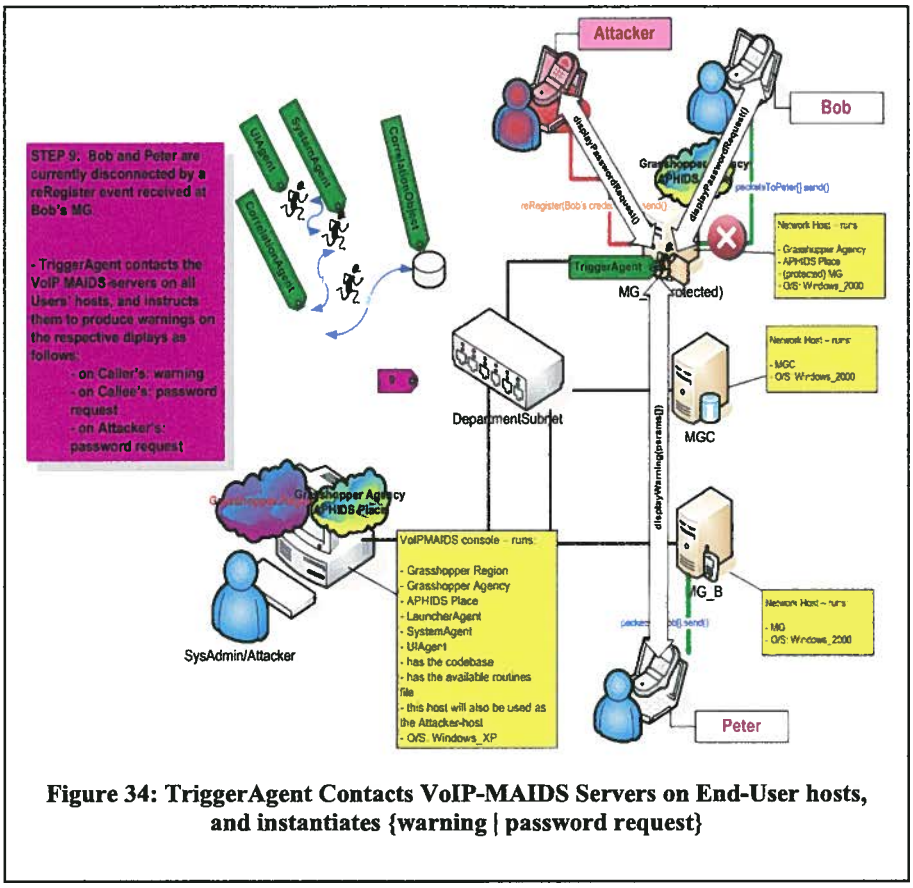


Figure 34: TriggerAgent Contacts VoIP-MAIDS Servers on End-User hosts, and instantiates {warning | password request}

10. Two situations are possible (from a potential attack perspective):

    a. The – supposed – Attacker is not an Attacker, but the Callee (i.e. Bob) who moved to another host. In this case:

        i. VoIP-MAIDS will receive no answer from the original Callee host to the password request.

        ii. VoIP-MAIDS will receive a correct answer (i.e. matching password) from the – supposed – Attacker machine.

        iii. No further action is required, as the – supposed – Attacker host is currently connected, whereas the former Callee host is disconnected.

    b. There exist a real Attacker. In this case:

        i. VoIP-MAIDS will receive no answer or an incorrect answer from the – supposed – Attacker machine to the password request.

        ii. VoIP-MAIDS will receive a correct answer (i.e. matching password) from the original Callee host to the password request.

        iii. `VoIPTriggerAgent` will insert the IP of the Attacker host into the blocked IP list of the protected MG; the Attacker host will be barred from connecting to the protected MG in the future.

        iv. VoIP-MAIDS-server on – known – Callee host is contacted, and instructed to produce re-login-request on (Bob's) display.

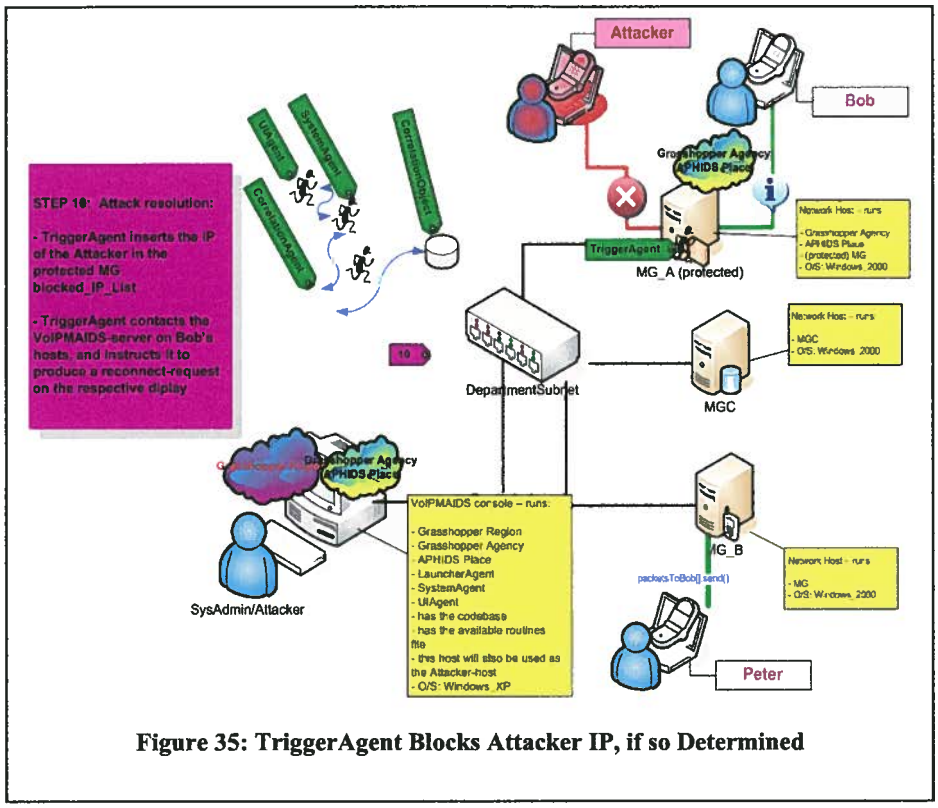


Figure 35: TriggerAgent Blocks Attacker IP, if so Determined

4.7. MAIDS for VoIP Cost

In this section, we propose to analyze what would be the memory footprint of VoIP-MAIDS, as perceived by an eventual implementer. That is, we are not concerned at this point with the impact on the VoIP-MAIDS console host (which is a host entirely dedicated to administer VoIP-MAIDS), but rather with the price VoIP IUT implementers would have to pay to deploy VoIP-MAIDS on their hosts.

As such, we distinguish two types of memory overhead that our system will impose:

1. Grasshopper Environment Cost

    - This cost is imposed by the running of the Grasshopper process by itself, abstraction being made of any agents. That is, any host that is to be visited by a (mobile) agent needs to be running a hosting environment.

    - In particular, all VoIP IUT hosts that are to receive, or to allow permanent presence of an agent, must run a Grasshopper Agency. Its estimated cost could be approximated to be 50 Mb; details as follows:

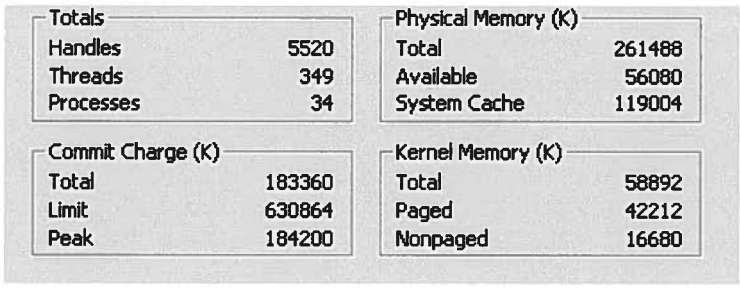


Figure 36: Memory Footprint – GrassHopper Agency Running

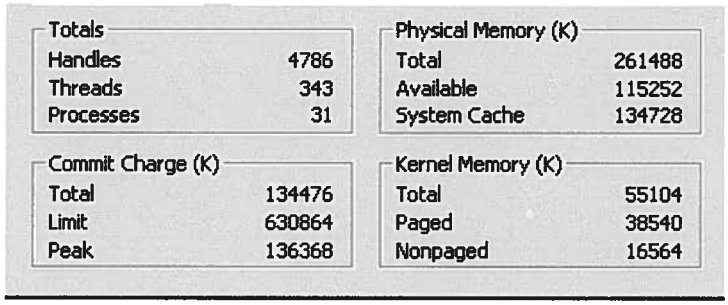


Figure 37: Memory Footprint Upon Fresh Boot (Windows 2000 Professional)

Taking the difference between the two values:

```
183,360 -
134,476 =
-------
~ 50MB
```

2. Agent Specific Cost
    - This cost is incurred for each agent allowed to operate at a given host.
    - Its estimated cost could be approximated to be 5.4 Mb; details as follows:

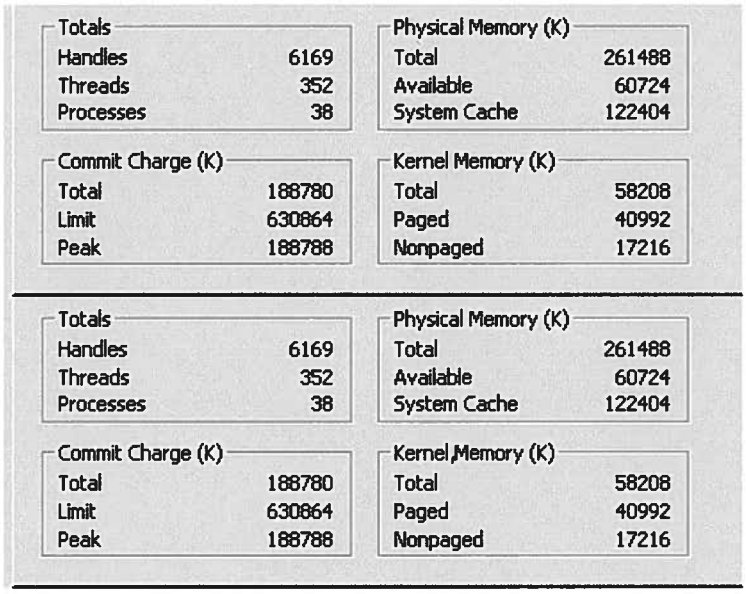


Figure 38: Memory Footprint – GrassHopper Agency + 1 MobileAgent

Taking the difference between the values with and without `VoIPTriggerAgent` running:

```
188,780 -
183,360 =
-------
~ 5.4MB
```

Chapter 5

5. Conclusion and Future Work

> Since 1988, the CERT/CC has responded to over 50,000 security incidents that have affected hundreds of thousands of Internet sites; has worked over 1,600 reported vulnerabilities, and has issued hundreds of advisories and bulletins [168].
>
> Richard Pethia, Director, CERT Centers

Mr. Pethia has repeatedly voiced concerns in public forums, oftentimes before the US Congress ([169]). CERT's findings showed a consistent, growing trend in the number of cyber attacks against legitimate business, institutions, and private entities. It could be inferred that we are all victims of the modern technology available today. That is, on the one hand, this technology lowered the barriers to entry with respect to the ability to launch powerful attacks; that ability is no longer the exclusive realm of individuals with strong coding skills and thorough understanding of software architecture and protocols. On the other hand, the advent of modern technology equally lowered the barriers to entry into the "eWorld": hosting an internet site for example is now trivial, just as is owning sophisticated computer equipment and ever renewed and affordable software packages. We believe that we need to stop and ponder whether the rush to release the next killer application has factored in the need to build a *secure* application. Feamster et al for example, explains that an estimated six hundred RFC's "explicitly acknowledge that they "punt" on security issues" [170].

Whether creating a completely secure application is utopia or not, until such application is built, we argue that there exist a need for applications whose role is to detect system intrusions, and wherever possible protect such systems and their users from vulnerabilities both known and unknown.

5.1. Summary of Thesis Contributions

In this thesis, we implemented the prototype of a tool meant to protect the most important injured party in an application security breach: its End-Users. To our knowledge, our work was one of the first attempts to implement an Intrusion Detection System for Voice over Internet Protocol *leveraging mobile agent technology*. In doing so, we have used an extensively tested and mature platform to deal with the agent-related issues (Grasshopper, in our case), thus enabling our system the focus on its detection/analysis/action tasks. As additional benefits of using Commercial Off-The-Shelf (COTS) for the mobile-agent platform, we enumerate:

- No need for modifications to the host system and/or protocols involved
- Simplification of the deployment tasks
- No single point of failure
- Strong security/encryption mechanisms available

Moreover, we draw on evidence that our system's architecture realizes the scalability characteristic of mobile agent approaches as agents are spawn upon demand, and are terminated as soon as they finish their task, thus minimizing the performance impact on the inhabited system. Furthermore, no communication or interaction is required between the VoIP IUT and VoIP-MAIDS's agent(s), aside from granting read-only access to specific audit data store. We have also designed a heuristic algorithm leveraging the sequential relationship between the audit data and the agent-monitored events. Most importantly however, by leveraging the State Transition Analysis method developed by Ilgun et al [34] we have implemented a gradual attack-response procedure, allowing us to achieve our stated goal: *protect application users in real time.*

5.2. Migrating to Improved VoIP COTS

The COTS IUT vendor has provided us with an enhanced version of the application, featuring among other things, major changes to its I/O operations and logging routine. We would like to migrate our IDS over this version as a first step towards improving our system's robustness.

5.3. Server Component and Communication Protocol

In the *action* phase of our algorithm, a connection is made to the Server component running on the End-User's Soft-Phone host. While adequate as a proof of concept, it is introducing a new vector of attack into the system, as its configuration can be abused by a knowledgeable attacker.

One possible method of addressing this would be to integrate the server component into the VoIP COTS. This will however contravene to our design goal of not interfering in any way with the inhabited system. Client-Server interaction is however a known and established technique, and several methods exist for securing it. The one we would prefer to implement would be the encryption of the communication channel. This is further substantiated by the capability offered by the underlying mobile agent platform (GrassHopper), which supports several forms of encryption, as outlined in Chapter 3.

In particular, we also envisage enhancing our IDS via a more involved End-User participation, as outlined in the next section.

5.4. End-User Profiling and Agent Participation

As Singh et al detail in [165], there exist a potential for significant enhancement of a VoIP IDS via the use of End-User profiling. In particular, call-patterns can be inferred after an adequate amount of time, and deviations from

the established set of patterns can be flagged. Conversely, they can be used to fine tune the initial pattern.

We envisage providing the COTS customer base with the option for an enhanced IDS component installation at install time. For the customers willing to participate, a mobile agent platform component will also be installed on their system, affording visits from VoIP-MAIDS agents. This will solve the open-communication problem as inter-agent communication is entirely under MAIDS for VoIP control and can be encrypted via the underlying platform's API's, will remove the need for a separate server component to be installed, and will allow the collection of application usage data for profiling purposes. For example, the following data can be used to infer the need for raising additional TriggerEvents:

- Usual calling time for a week-day
- Usual call duration for a week-day
- Usual calling time for a week-end day
- Usual call duration for a week-end day
- List of most currently dialed IP's for a week-day
- List of most currently dialed IP's for a week-end day

5.5. Improvements to MAIDS for VoIP's Algorithm

5.5.1. Data Correlation Improvements

A first step would be to perform correlation activities with the data residing at the MGC, and also at the other MG. That is, under the current implementation, MAIDS for VoIP mines data from the monitored MG only; a strong limitation of our project. It would greatly enhance our IDS's performance if correlation activities will involve information extracted from state machines running at the MGC, and also at the other participating MG's.

To accommodate for this development, the tasks currently performed solely by the `VoIPTriggerAgent` will be delegated to an adequately designed `VoIPTaskAgent` and a `VoIPActionAgent`, and this is certainly in the realm of immediate possibilities given VoIP-MAIDS's underlying APHIDS framework.

5.5.2. Signature Rules Creation

The concept of audit trail analysis stands at the basis of IDS technology, and we have introduced and elaborated on the topic in Section 2.1. In particular, we have presented the work by Habra et al [35] in Section 2.1.7, among others. The authors introduce not only a novel IDS, but most importantly an innovative language for audit trail query expression. As per its authors, "RUSSEL is a rule-based language which is tailor-made for the analysis of sequential files in one and only one pass" [35]. For example, Habra et al propose the following format for rule creation based on audit data – Figure 39:

```
rule rule_name
#
# Comments
#
begin
if evt='event_name' and not audit_token(parameter_1,..., parameter_n)
                    and not audit_token(parameter_1,..., parameter_n)

 ---->Trigger off for next system_function(parameter_1, ... parameter_n)
fi ;

Trigger off for next rule_name

end
```

Figure 39: RUSSEL language format [35]

We propose adapting the RUSSEL language to fit our state machine algorithm, and use its power to generate intrusion detection rules which can be applied at run time, and also shared with agents residing at other MG/MGC's.

5.5.3. Implement Orphan Packet Detection

It is known that upon a successful service tear down attack, the End-User at the Callee end gets no feedback from the VoIP IUT of this fact. If this particular End-User were to be talking at the time of the attack, VoIP-MAIDS can correlate the fact that this End-User is no longer connected with the MG with the fact that (VoIP) packets are being received from this particular host.

5.5.4. Realistic Work Load Handling

Our system has been tested in the simple scenario where the IUT has two registered users; a necessary first step, and also another limitation of our project. Next, a design decision must me made with respect to what would be the best resolution to the multiple End-Users problem:

- Have a threaded `VoIPTriggerAgent`; where each thread deals with a newly connected End-User.

- Have multiple `VoIPTriggerAgents`; where each new `VoIPTriggerAgent` is spawned on demand by `CorrelationAgent`, to be paired with any newly connected End-User.

- Implement an agent-caching approach, where `VoIPTriggerAgent` is allowed to spawn "children" `VoIPTriggerAgents`, that can be flushed to disk and awaken as new End-Users connect or disconnect from the VoIP IUT monitored system (we further elaborate on this idea on Section 6.6).

5.5.5. Expand Response Action

Once a decision was made that a certain host is an Attacker host and its IP address must be blocked, submit this IP address to all MG and MGC hosts part of the monitored network. This stems from the fact that the Attacker can repeat the same attack at another host part of the VoIP IUT system, since VoIP-MAIDS is not currently broadcasting harvested malicious IP addresses.

5.5.6. Handle Case of Multiple IP Owners

It is a known fact that End-Users may "own" several external IP addresses, and use whichever at any given time. To mitigate for this case, we envisage implementing and maintaining a list of known IP addresses for each End-User known to "own" more than one IP. In the case where a 'reRegister' event is received with the credentials of a currently connected End-User, but where the origin of this event can be cross-referenced with an entry in the list, there will be no need to raise a TriggerEvent.

5.6. Integrated Graphic User Interface for Vulnerability Testing

The underlying platform MAIDS for VoIP leverages – APHIDS [163], affords audit data collection from a variety of third party Intrusion Detection Systems. We propose expanding VoIP-MAIDS's intrusion detection capabilities by building into our system "awareness" of any additional IDS's which may be running on the inhabited system. Once that happens, we foresee a need for a better communication vehicle with the SysAdmin, and propose a GUI more suited to the correlation task. One possible iteration is presented in Figure 40:

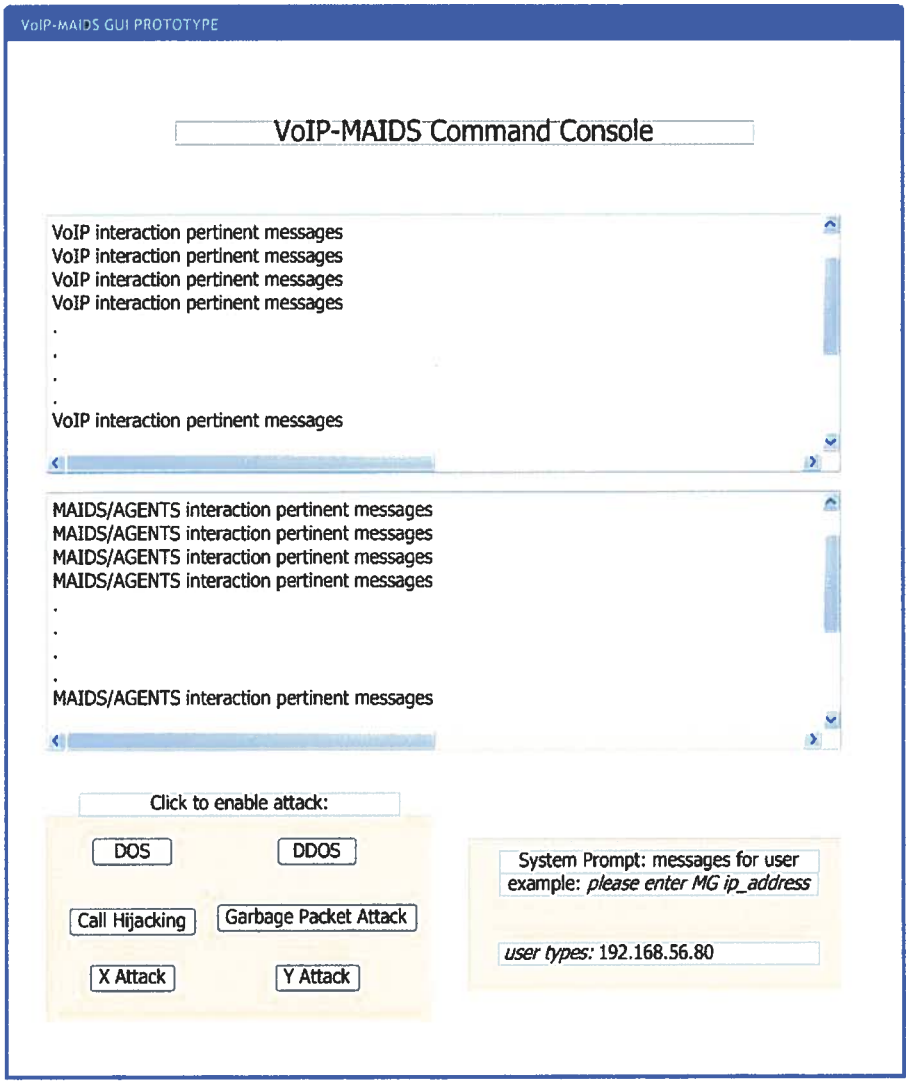


Figure 40: Suggested Graphical User Interface

5.7. Agent Caching

In this section, we outline a potential improvement we see to VoIP-MAIDS's underlying platform (APHIDS), in its multi-IDS integration capability. Namely, *Task Agents* will constitute the main target for our caching approach. This stems from the fact that the other low-level agent type (i.e. *Action Agents*) is foreseen at this point as a short-lived, per-case-instantiation agent. *Task Agents* on the other hand, will see their complexity vary according to the type of environment/system where they are deployed.

As a case in point, we do not think that we are too disconnected from reality if we were to infer a hierarchy among the machines connected to a (corporate) LAN, as follows:

[1] Machines running full bodied IDS's are dedicated machines, i.e. closed to public and not available for every-day low-level use.

[2] Systems running SSH , FTP, or WWW Servers, which may very well be open to the general user population.

Following the arrival of a pertinent *Trigger Event*, we might decide that the investigation should attempt to correlate data from the log files of both Snort and Argus – to give an Open Source example, and/or from other available IDS's. As such, *Task Agents* will be send to all the machines involved (i.e. hosting the respective IDS's). Regardless, we infer as extremely likely the probability that these specific *Task Agents* will be used again in the near future – following the arrival of yet another *Trigger Event*.

Thus, we draw on evidence that resources can be saved if we do not send a new *Task Agent* to the machine(s) running Snort (or any other IDS) every time a new *Trigger Event* arrives. Instead, we suggest *caching* these specific *Task Agents* at the machines where the respective IDS's reside.

Henceforth, we suggest all `TaskAgents` incorporate a *timer* as part of their code base. The agent will be kept alive (i.e. residing in the host machine's memory) for as long as the timer specifies, provided no new tasks arrive. Upon the timer's expiration, the agent will be either killed (if it was send to a type '[2]' machine – according to our above classification), or flushed to permanent storage on the hard-drive (if it was send to a type '[1]' machine).

The idea behind our suggested scheme is to never need to re-instantiate `TaskAgents` again. Instead, they will be flushed to the hard-drive and reactivated as needed. In addition, by always re-activating one extra `TaskAgent` on top of what is currently needed we target the problem of slow access to peripheral devices (such as hard-disk drives): there will always be (with the exception of the case when the maximum of 5 is reached) an extra `TaskAgent` waiting to be used in a data mining activity. Figure 41 depicts our suggestion:

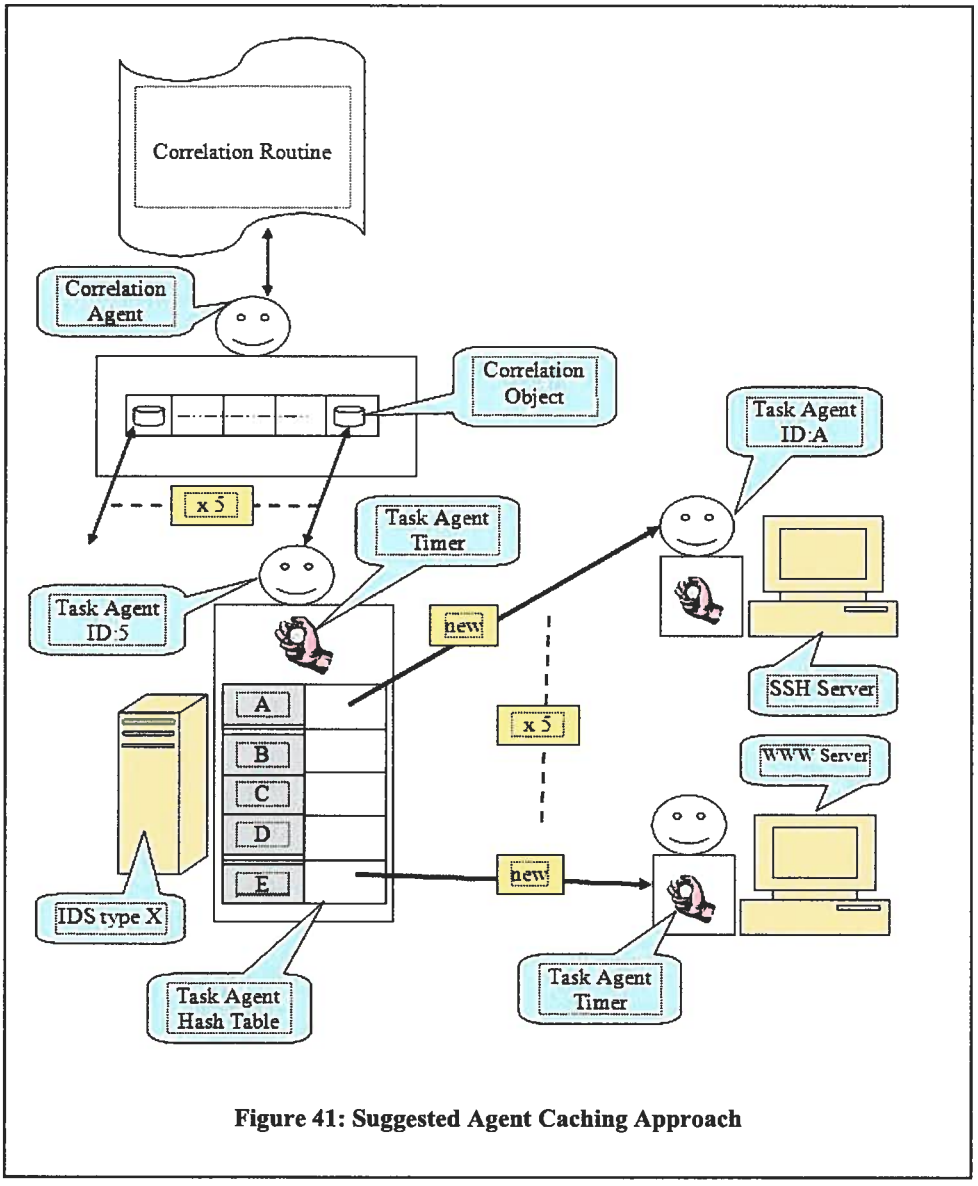


Figure 41: Suggested Agent Caching Approach

6. Bibliography

[1] Carnegie Mellon Software Engineering Institute, CERT Coordination Center, "Multiple vulnerabilities in implementation of the session initiation protocol (SIP)," 2003. [Online]. Available: http://www.cert.org/advisories/CA-2003-06.html [Accessed: July 12, 2006].

[2] G. Bruneau, "The History and Evolution of Intrusion Detection," SANS Institute. 2004. [Online]. Available: http://www.sans.org/reading_room/whitepapers/detection/344.php. [Accessed: July 15, 2006].

[3] G. Vigna, F. Valeur, and R. A. Kemmerer, "Designing and implementing a family of intrusion detection systems," in Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT international Symposium on Foundations of Software Engineering, 2003, pp. 88-97.

[4] J. P. Anderson, Computer security threat monitoring and surveillance. Fort Washington, PA: James P. Anderson Co., 1980.

[5] Stanford Research Institute, "Timeline of SRI International Innovations: 1980s," 2006. [Online]. Available: http://www.sri.com/about/timeline/timeline4.html [Accessed: August 2, 2006].

[6] D. E. Denning, "An intrusion-detection model," in IEEE Transactions on Software Engineering, 1987, vol. 13, no. 2, pp. 222-232.

[7] T. L. Heberlein, G. V. Dias, K. L. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," in Proceeding of the 1990 IEEE Symposium on Research in Security and Privacy, 1990, pp. 296-304.

[8] H. V. Poor, An Introduction to Signal Detection and Estimation, 2nd edition. New York, N.Y., Springer, 1994.

[9] L. A. Wainstein, and V. D. Zubakov, Extraction of Signals from Noise. Englewood Cliffs, IL: Prentice-Hall, 1962.

[10] M. F. Duarte, M. A. Davenport, M. B. Wakin, and R.G. Baraniuk, "Sparse Signal Detection from Incoherent Projections," in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2006, pp. III-305-III-308.

[11] Y. Qi, and T. P. Minka, "Window-Based Expectation Propagation for Adaptive Signal Detection in Flat-Fading Channels," IEEE Transactions on Wireless Communications, vol. 6, no. 1, January 2007.

[12] V. G. Easton, and J. H. McColl, "Statistics Glossary," October 2006. [Online]. Available: http://www.cas.lancs.ac.uk/glossary_v1.1/hyptest.html. [Accessed: August 14, 2006].

[13] J. McHugh, A. Christie, and J. Allen, "Defending Yourself: The Role of Intrusion Detection Systems," IEEE Software, vol. 17, pp. 42-51, September 2000.

[14] R. Bace, and P. Mell, "Special Publication on Intrusion Detection Systems," National Institute of Standards and Technology, Gaithersburg, Md., Tech. Rep. SP 800-31, November 2001.

[15] Open Source Initiative, "The Open Source Definition," Open Source Initiative, 2007. [Online]. Available: http://www.opensource.org/docs/definition.php. [Accessed: February 06, 2007].

[16] R. A. Kemmerer, and G. Vigna, "Intrusion Detection: A Brief History and Overview," Computer, vol. 35, no. 4, pp. 27-30, April 2002.

[17] G. Vigna, R. A. Kemmerer, and P. Blix, "Designing a Web of Highly Configurable Intrusion Detection Sensors," in Proceedings of the Fourth International Symposium on Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, vol. 2212, Springer Verlag, New York, 2001.

[18] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," in Proceedings of the USENIX Lisa Conference, 1999, vol. 13, no. 11, pp. 229-238.

[19] J. Koziol, Intrusion Detection with Snort, 2nd Revised Edition, Indianapolis, IN, Sams Publishing, 2003.

[20] C. Gerg, Manging Security with Snort and IDS Tools, 1st Edition, Sebastopol, CA, O'Reilly Media, 2004.

[21] Sourcefire, "Writing Snort Rules: How to Write Snort Rules and Keep Your Sanity," December 2006. [Online]. Available: http://snort.org/docs/snort_htmanuals/htmanual_261/node147.html. [Accessed: September 24, 2006].

[22] Naval Surface Warfare Center, "NSWC Shadow Index," Naval Source Warfare Center - Dahlgren Lab, Information Assurance Office, 2000. [Online]. Available: http://www.nswc.navy.mil/ISSEC/CID/Install3-MS.htm. [Accessed: September 08, 2006].

[23] V. Paxon, "Bro: A System for Detecting Network Intruders in Real-Time," Computer Networks, vol. 31, no. 23-24, pp. 2435-2463, December 1999.

[24] M. Russinovitch, "Using Rootkits to Defeat Digital Rights Mangement," April 2005. [Online]. Available: http://blogs.technet.com/markrussinovich/archive/2006/02/06/using-rootkits-to-defeat-digital-rights-management.aspx. [Accessed: August 22, 2006].

[25] F. Kerschbaum, E. H. Spafford, and D. Zamboni, "Using embedded sensors for detecting network attacks," in Proceedings of the First ACM Workshop on Intrusion Detection Systems, CERIAS TR 2000-25, November 2000.

[26] OpenBSD, "The OpenBSD Project: Free, Functional, Secure," 1996. [Online]. Available: http://www.openbsd.org/. [Accessed: August 25, 2006].

[27] J. Hochberg, K. Jackson, C. Stallings, J. F. McClary, D. DuBois, and J. Ford, "NADIR: An Automated System for Detecting Network Intrusion and Misuse," Computers & Security, vol. 12, no. 3, pp. 235-248, 1993.

[28] S. Kumar and E. H. Spafford, "A Software Architecture to Support Misuse Intrusion Detection," The COAST Project, Dept. of Computer Sciences, Purdue University, West Lafayette, IN, Tech. Rep. 47907-1398, 1995.

[29] K. Mcleish, "Petri Nets," 1999. [Online]. Available: http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C10/10-7.html. [Accessed: August 25, 2006].

[30] Internet Security Systems, "RealSecure 6.0 Frequently Asked Questions," Internet Security Systems, 1994. [Online]. Available: http://documents.iss.net/literature/RealSecure/rs60_faq.pdf. [Accessed: August 27, 2006].

[31] S. R. Snapp, S. E. Smaha, D. M. Teal, and T. Grance, "The DIDS (Distributed Intrusion Detection System) Prototype," in Proceedings of the Summer USENIX Conference, San Antonio, TX, 1992, pp. 227-233.

[32] S. E. Smaha, "Haystack: An Intrusion Detection System," in Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference, Orlando, FL, 1988, pp. 37-44.

[33] U. Lindqvist, and P. A. Porras, "Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)," in Proceedings of the 1999 IEEE Symposium on Security and Privacy, Oakland, CA, 1999, pp. 146-161.

[34] K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection System," IEEE Transactions On Software Engineering, vol. 21, no. 3, pp. 181-199, March 1998.

[35] N. Habra, B. L. Charlier, A. Nounji, and I. Mathieu, "ASAX: Software Architecture and Rule-Based Language for Universal Audit Trail Analysis," in Proceedings of ESORICS '92: European Symposium on Research in Computer Security, Toulouse, France, 1992, pp. 435-450.

[36] K. Ilgun, "USTAT - A Real-time Intrusion Detection System for UNIX," M.S. thesis, University of California at Santa Barbara, Santa Barbara, CA, 1992.

[37] K. Ilgun, "USTAT - A Real-time Intrusion Detection System for UNIX," in Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, 1993, pp. 16-28.

[38] AI Topics, "Expert Systems," The American Association for Artificial Intelligence, December 2006. [Online]. Available: http://www.aaai.org/AITopics/html/expert.html. [Accessed: February 08, 2007].

[39] S. N. Chari and P.-C. Cheng, "BlueBoX: A Policy-Driven, Host-Based Intrusion Detection System," ACM Transactions on Information and System Security, vol. 6, no. 2, pp. 173-200, May 2003.

[40] G. White, and V. Pooch, "Cooperating Security Managers: Distributed Intrusion Detection Systems," Computers & Security, vol. 15, no. 5, pp. 441-450, 1996.

[41] Yao Yu, Gao Fu-xiang, and Yu Ge. Hybrid BP/CNN Neural Network for Intrusion Detection. In Proceedings of the 3rd International Conference on Information Security InfoSecu '04, vol. 85, pp. 226-228, November 2004.

[42] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A Sense of Self for Unix Processes," in Proceedings of the 1996 IEEE Symposium on Security and Privacy, Los Alamitos, CA, 1996, pp. 120-128.

[43] H. S. Vaccaro and G. E. Liepins, "Detection of Anomalous Computer Session Activity," in Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, CA, 1989, pp. 280-289.

[44] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, and P. G. Neuman, "A Real-Time Intrusion Detection Expert System (IDES)," SRI International, Menlo Park CA, Tech. Rep. 6784, 1992. [Online]. Available: http://www.csl.sri.com/papers/9sri/. [Accessed: February 12, 2007].

[45] R. Sekar, M.Bendre, D. Dhurjat, and P. Bollineni, "A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors," in Proceedings of the 2001 IEEE Symposium on Security and Privacy, Oakland, CA, 2001, pp. 144-155.

[46] J. Daciuk, "Finite State Automata," June 03, 1998. [Online]. Available: http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/thesis/node12.html. [Accessed: August 28, 2006].

[47] D. Anderson, T. Frivold, A. Tamaru, and A. Valdes, "Next-generation Intrusion Detection System (NIDES)," Computer Science Laboratory, SRI International, Menlo Park, CA, Tech. Rep. SRI-CSL-95-07, 1995.

[48] P. Porras and P. Neumann, "EMERALD: Event Monitoring Enabling Respionses to Anomalous Live Disturbances," in Proceeding of the 19th National Computer Security Conference, Baltimore, MD, 1997, pp. 353-365.

[49] Y. F. Jou, F. Gong, C. Sargor, S. F. Wu, and W. R. Cleaveland, "Architecture Design of a Scalable Intrusion Detection System for the Emerging network Infrastructure," The Defense Advanced Research Projects Agency (DARPA), Arlington VA, Tech. Rep. CDRL A005, Apr 1997.

[50] CISCO, "Open Shortest Path First," Cisco Documentation, June 1999. [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ospf.htm. [Accessed: August 31, 2006].

[51] S. Axelsson, "Research in Intrusion-Detection Systems: A Survey," Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, Tech. Rep. TR:98-17, 1999.

[52] Talisker Security Wizardry, "Network Intrusion Detection Systems," 2004. [Online]. Available: http://www.networkintrusion.co.uk/N_ids.htm. [Accessed: June 14, 2006].

[53] Talisker Security Wizardry, "Host Intrusion Detection Systems," 2004. [Online]. Available: http://www.networkintrusion.co.uk/HIDS.htm. [Accessed: June 14, 2006].

[54] C. Ko, M. Ruschitzka, and K. Levitt, "Execution Monitoring of Security-Critical Programs in Distributed Systems: a Specification-Based Approach," in Proceedings of the 1997 IEEE Symposium on Security and Privacy, Washington, DC, 1997, pp.175-187.

[55] Network Working Group, "Request For Comments - 1321," RFC Documentation, April 1992. [Online]. Available: http://tools.ietf.org/html/rfc1321. [Accessed: August 30, 2006].

[56] G. H. Kim and E. H. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker," Dept. of Computer Sciences, Purdue University, West Lafayette, IN, Tech. Rep. CSD-TR-93-071, 1993.

[57] Samhain Design Labs, "Samhain File Integrity System Documentation," May 2006. [Online]. Available: http://la-samhna.de/samhain/manual/ [Accessed: September 30, 2006].

[58] Paul Vixie, "UNIX man pages: crontab," January 1994. [Online]. Available: http://unixhelp.ed.ac.uk/CGI/man-cgi?crontab+5. [Accessed: August 30, 2006].

[59] N. Vanderavero, X. Brouckaert, O. Bonaventure, and B. L. Charlier, "The HoneyTank: a Scalable Approach to Collect Malicious Internet Traffic," International Infrastructure Survivability Workshop (IISW'04), Lisbon, Portugal, 2004.

[60] Christian Kreibich, "Honeycomb: Automated Signature Creation Using Honeypots," June 2003. [Online]. Available: http://www.icir.org/christian/honeycomb/index.html. [Accessed: September 30, 2006].

[61] N. Vanderavero, and B. L. Charlier, "Towards a More Stateful and Accurate HoneyTank," in Proceedings of the 5th Conference on Security and Network Architectures (SAR 2006), Seignosse, France, 2006, pp. 13-27.

[62] Fyodor, "Nmap," September 1999. . [Online]. Available: http://insecure.org/nmap/. [Accessed: September 30, 2006].

[63] S. T. King, and P. M. Chen, "Backtracking Intrusions," in Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, 2003, pp. 223-236.

[64] Talisker Security Wizardry, "File Integrity Checkers," 2004. [Online]. Available: http://www.networkintrusion.co.uk/integrity.htm. [Accessed:

June 14, 2006].

[65] P. Helman, and G. Liepins, "Statistical Foundations of Audit Trail Analysis for the Detection of Computer Misuse," in IEEE Transactions on Software Engineering, vol. 19, no. 9, pp.886-901, September 1993.

[66] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson, "A Methodology for Testing Intrusion Detection Systems," in IEEE Transactions on Software Engineering, vol. 22, no. 10, pp.719-729, October 1996.

[67] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating Intrusion Detection Systems: the 1998 DARPA off-line Intrusion Detection Evaluation," in Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX '00), Hilton Head, SC, 2000, vol. 2, pp. 12-26.

[68] R. P. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and D. Kumar, "The 1999 DARPA off-line Intrusion Detection Evaluation," in Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 34, no. 4, pp. 579-595, October 2000.

[69] J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory," in ACM Transactions on Information and System Security (TISSEC), vol. 3, no. 4, pp. 262-294, November 2000.

[70] S. Axelsson, "The Base-rate Fallacy and its Implications for the Difficulty of Intrusion Detection," in Proceedings of the 6th ACM Conference on Computer and Communications Security, Kent Ridge, Singapore, 1999, pp. 1-7.

[71] R. A. Maxion, and K. M. C. Tan, "Benchmarking Anomaly-Based Detection Systems," in Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8), New-York, NY, June 2000, pp. 623-630.

[72] Answers.com, "Entropy," September 2006. [Online]. Available: http://www.answers.com/topic/entropy?cat=health. [Accessed: October 19, 2006].

[73] J. W. Ulvila, and J. E. Gaffney, Jr., "Evaluation of Intrusion Detection Systems," Journal of Research of the National Institute of Standards and Technology, vol. 108, no. 6, pp. 453-473, November-December 2003.

[74] S. Antonatos, K. Anagnostakis, and E. Markatos, "Generating Realistic Workloads for Network Intrusion Detection Systems," in Proceedings of Fourth International Workshop on Software and Performance (WOSP 2004), Redwood City, CA, 2004, pp. 207-215.

[75] S. Antonatos, K. Anagnostakis, M. Polychronakis, and E. Markatos "Performance analysis of content matching intrusion detection system," in Proceedings of the 4th IEEE/IPSJ Symposium on Applications and the Internet (SAINT 2004), Tokyo, Japan, 2004, pp. 208-215.

[76] A. Valdes, and K. Skinner, "Probabilistic Alert Correlation," in Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection, Davis, CA, 2001, pp. 54-68.

[77] R. P. Goldman, W. Heimerdinger, S. A. Harp, C. W. Geib, V. Thomas, and R. L. Carter, "Information Modeling for Intrusion Report Aggregation," in Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX II '01), Anaheim, CA, 2001, pp. 0329.

[78] H. Debar, and A. Wespi, "Aggregation and Correlation of Intrusion-Detection Alerts," in Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection, Davis, CA, 2001, pp. 85-103.

[79] P. A. Porras, M. W. Fong, and A. Valdes, "A Mission-Impact-Based Approach to INFOSEC Alarm Correlation," in Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection, Zurich, Switzerland, 2002, pp. 95-114.

[80] Answers.com, "Software Agent," October 2006. [Online]. Available: http://www.answers.com/software+agent?cat=technology. [Accessed: February 10, 2007].

[81] H. S. Nwana, "Software Agents: An Overview," in Knowledge Engineering Review, Vol. 11, No 3, pp.1-40, Sept 1996.

[82] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," in Proceedings of the IEEE Transactions on Software Engineering, Vol. 24, No 5, pp.342-361, May 1998.

[83] F. Schweitzer and J. Zimmermann, "Communication and Self-Organization in Complex Systems: A Basic Approach," in Knowledge, Complexity and Innovation Systems, Springer, Berlin, 2001, pp. 275-296.

[84] H. V. D. Parunak and S. Brueckner, "Entropy and Self-Organization in Multi-Agent Systems," in Proceedings of the International Conference on Autonomous Agents, 2001, pp. 124-130.

[85] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an Agent Communication Language," in Proceedings of the Third International Conference on Information and Knowledge Management, 1994, pp. 456-463.

[86] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White, "MASIF, the OMG mobile agent system interoperability facility," in Proceedings of Mobile Agents'98, 1998, vol. 1477, pp 50-67.

[87] J. Baumann, F. Hohl, N. Radouniklis, M. Straβer, and K. Rothermel, "Lecture Notes in Computer Science: Communication Concepts for Mobile Agent Systems," Heidelberg, Germany, Springer, 1997, pp. 123-135.

[88] General Magic, Inc, "The Telescript Language Reference," Sunnyvale, CA, 1995.

[89] D. Peter, "Mobile Telescript Agents and the Web," in Forty-First IEEE Computer Society International Conference, 1996, pp. 52-57.

[90] R. S. Gray, "Agent Tcl: A Transportable Agent System," in Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management, 1995.

[91] J. Baumann, F. Hohl, K. Rothermel, and M. Straβer, "Mole – Concepts of a Mobile Agent System," Stuttgart University, Stuttgart, Germany, Tech.

Rep. TR-1997-15, 1997.

[92] D. B. Lange, M. Oshima, G. Karjoth, and K. Kosaka, "Lecture Notes in Computer Science: Aglets: Programming Mobile Agents in Java," Heidelberg, Germany, Springer, 1997, pp. 253-266.

[93] D. B. Lange, and M. Oshima, "Mobile Agents with Java: The Aglet API," in World Wide Web, 1998, vol. 1, no. 3, pp. 111-121.

[94] IKV++ GmbH, "Grasshopper Basics And Concepts" Release 2.2., Bernburger Strasse 24-25, 10963 Berlin, Germany.

[95] D. B. Lange, and M. Oshima, "Seven Good Reasons for Mobile Agents: Dispatch your agents; shut off your machine," in Communications of the ACM, 1999, vol. 42, no. 3, pp. 88-89.

[96] G. Vigna, "Mobile Agents: Ten Reasons For Failure," in Proceedings of the 2004 International Conference on Mobile Data Management (MDM'04), 2004, pp. 298-299.

[97] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni, "An Architecture for Intrusion Detection using Autonomous Agents," in Proceedings of the 14th Annual Computer Society Applications Conference, 1998, pp. 13-32.

[98] A. Bieszczad, B. Pagurek, and T. White, "Mobile Agents for Network Management," in IEEE Communications Surveys, 1998, vol. 1, no. 1, pp. 2-9.

[99] M. Asaka, A. Taguchi, and S. Goto, "The Implementation of IDA: an Intrusion Detection Agent System," in Proceedings of the 11th FIRST Conference, 1999.

[100] J. D. de Queiroz, L. F. R. da Costa Carmo, L. Primez, "Micael: An Autonomous Mobile Agent System to Protect New Generation Networked Applications," in 2nd Annual Workshop on Recent Advances in Intrusion Detection, 1999.

[101] G. Helmer, J. S. K. Wong, V. Honavar, L. Miller, and Y. Wang, "Lightweight Agents for Intrusion Detection," in The Journal of Systems and Software, vol. 67, pp. 109-122, June 2002.

[102] C. Kruegel and T. Toth, "Sparta: A Mobile Agent based Intrusion Detection System," in Proceedings of the IFIP Conference on Network Security, 2001.

[103] C. Kruegel and T. Toth, "Flexible, Mobile Agent based Intrusion Detection for Dynamic Networks," in Proceedings of the European Wireless, 2002.

[104] N. Foukia, D. Billard, and Pr. J. Harms, "Computer System Immunity using Mobile Agents," in The 8th Annual Workshop of the HP Openview University Association, 2001.

[105] N. Foukia, S. Hassas, S. Frenet, and P. Albuquerque, "Combining Immune Systems and Social Insect Metaphors: A Paradigm for Distributed Intrusion Detection and Response System," in Proceedings of the 5th International Workshop for Mobile Agents for Telecommunication Applications, 2003.

[106] S. Chatterjee, and S. Samaddar, "Voice-Based Communication: A V*IP Architecture and its Revolutionary Implications for Business", in Proceedings of IEEE Enterprise Networking Conference, 2001, pp. 25-35.

[107] M. Friedman, "VoIP Market To Hit $4 Billion By 2010: Report," July 2005. [Online]. Available: http://www.informationweek.com/news/management/showArticle.jhtml;jsessionid=KNTVXBJAUAVK2QSNDLRSKH0CJUNN2JVN?articleID=166401027&_requestid=425036. [Accessed: February 11, 2007].

[108] Frost & Sullivan, "Company Information," June 2006. [Online]. Available: http://www.frost.com/prod/servlet/company-info.pag. [Accessed: February 8, 2007].

[109] L. Kimber, "Bandwidth-Starved Europe Gets Caching Software Early," May 1998. [Online]. Available: http://www.techweb.com/wire/story/TWB19980507S0005/. [Accessed: February 8, 2007].

[110] K. N. Cukier, "Bandwidth Colonialism? The Implications of Internet Infrastructure on International E-Commerce," June 1999. [Online]. Available: http://www.isoc.org/inet99/proceedings/1e/1e_2.htm. [Accessed: February 8, 2007].

[111] M. Hassan, A. Nayandoro, and M. Atiquzzaman, "Internet Telephony: Services, Technical Challenges, and Products," in IEEE Communications Magazine, 2000, vol. 38, no. 4, pp. 96-103.

[112] B. Goode, "Voice Over Internet Protocol (VoIP)," in Proceedings of the IEEE, 2002, vol. 90, no. 9, pp. 1495-1517.

[113] X. Chen, C. Wang, D. Xuan, Z. Li, Y. Min, and W. Zhao, "Survey on QoS Management of VoIP," in Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing (ICCNMC'03), 2003, pp. 69-77.

[114] S. Niccolini, R. GT. Garroppo, J. Ott, S. Prelle, J. Kuthan, J. Janak, S. Ubik, M. Brandl, D. Daskopoulos, E. Verhaern, and E. Dobbelsteijn, "IP Telophony Cookbook," September 2001. [Online]. Available: http://www.informatik.uni-bremen.de/~prelle/terena/cookbook/main/index.html. [Accessed: March 2, 2007].

[115] International Telecommunication Union, "H.323: Packet-based multimedia communications systems," November 1996. [Online]. Available: http://www.itu.int/rec/T-REC-H.323/e. [Accessed: February 19 2007].

[116] Javvin Network Management and Security, "Q.931: ISDN Network Layer Protocol for Signaling," June 2005. [Online]. Available:http://www.javvin.com/protocolQ931.html. [Accessed:: February 12, 2007].

[117] International Telecommunication Union, "Q.931: ISDN user-network interface layer 3 specification for basic call control," August 2006. [Online]. Available: http://www.itu.int/rec/T-REC-Q.931/e. [Accessed:: February 12, 2007].

[118] M. Arango, A. Dugan, I. Elliott, C. Huitema, S. Pickett, "RFC 2705 – Media Gateway Control Protocol (MGCP) Version 1.0," October 1999. [Online]. Available: http://www.faqs.org/rfcs/rfc2705.html. [Accessed: February 13, 2007].

[119] F. Andreasen, B. Foster, "RFC 3435: Media Gateway Control Protocol (MGCP)," January 2003. [Online]. Available: http://tools.ietf.org/html/rfc3435. [Accessed: February 12, 2007].

[120] C. Groves, M. Pantaleo, T. Anderson, T. Taylor, "RFC 3525 – Gateway Control Protocol Version 1," June 2003. [Online]. Available: http://www.faqs.org/rfcs/rfc3525.html. [Accessed: February 12, 2007].

[121] F. Cuervo, N. Greene, A. Rayhan, C. Huitema, B. Rosen, J. Segers, "RFC 3015: Megaco Protocol Version 1.0," January 2003. [Online]. Available: http://www.faqs.org/rfcs/rfc3015.html. [Accessed: February 12, 2007].

[122] M. Kallas, and Z. Bilalis, "Megaco/H.248 NAS Package," March 2000. [Online]. Available: http://tools.ietf.org/html/draft-ietf-megaco-naspkg-00. [Accessed: February 18 2007].

[123] M. Brahmanapally, P. Viswanadham, K. Gundamaraju, "Megaco/H.248 Call flow examples," November 2004. [Online]. Available: http://tools.ietf.org/html/draft-ietf-megaco-callflows-04. [Accessed: February 18 2007].

[124] International Telecommunication Union, "E.164: List of ITU-T Recommendations E.164 Assigned Country Codes (Position on 1 May 2005)," May 2005. [Online]. Available: http://www.itu.int/itudoc/itu-t/ob-lists/icc/e164_763.html. [Accessed: February 18 2007].

[125] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "RFC 3261 – SIP: Session Initiation Protocol," June 2002. [Online]. Available: http://tools.ietf.org/html/rfc3261. [Accessed: February 19, 2007].

[126] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "RFC 4123 – Session Initiation Protocol (SIP)-H.323 Interworking Requirements," July 2005. [Online]. Available: http://www.faqs.org/rfcs/rfc4123.html. [Accessed: February 22, 2007].

[127] J. Rosenberg, P. Kyzivat, "RFC 4596 – Guidelines for Usage of the Session Initiation Protocol (SIP) Caller Preferences Extension," July 2006. [Online]. Available: http://www.faqs.org/rfcs/rfc4596.html. [Accessed: February 21, 2007].

[128] A. Johnston, O. Levin, "RFC 4579 –Session Initiation Protocol (SIP) Call Control – Conferencing for User Agents," August 2006. [Online]. Available: http://www.faqs.org/rfcs/rfc4596.html. [Accessed: February 21, 2007].

[129] J. Marzulla, "The SIP Center," August 2000. [Online]. Available: http://www.sipcenter.com/sip.nsf/html/Architecture. [Accessed: March 09, 2007].

[130] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RFC 1889 – RTP: A Transport Protocol for Real-Time Applications," January 1996. [Online]. Available: http://www.faqs.org/rfcs/rfc1889.html. [Accessed: March 09, 2007].

[131] M. Handley, V. Jacobson, "RFC 2327 – SDP: Session Description Protocol," April 1998. [Online]. Available: http://tools.ietf.org/html/rfc2327. [Accessed: March 09, 2007].

[132] M. Handley, V. Jacobson, and C. Perkins, "RFC 4566 – SDP: Session Description Protocol," July 2006. [Online]. Available: http://tools.ietf.org/html/rfc4566. [Accessed: March 09, 2007].

[133] R. K. Bhattacharyya, "New Challenges For Telephone Companies to Secure Switching Systems," in Proceedings of the 25th Annual 1991 IEEE International Conference, 1991, pp. 41-46.

[134] S. Vuong and Y. Bai, "A Survey of Vulnerability and Vulnerability Testing," The Unuversity of British Columbia, Vancouver, BC, Technical Report, 2003.

[135] R. Ackermann, M. Schumacher, U. Roedig, and R. Steinmetz, "Vulnerabilities And Security Limitations Of Current IP Telephony Systems," Darmstadt University of Technology, Darmstadt, Germany, Technical Report, 2005.

[136] The PROTOS Project, "PROTOS – Security Testing of Protocol Implementations," April 2004. [Online]. Available: http://www.ee.oulu.fi/research/ouspg/protos/. [Accessed: March 10, 2007].

[137] US-CERT, "Vulnerability Note VU#749342 – Multiple vulnerabilities in H.323 implementations," January 2004. [Online]. Available: http://www.kb.cert.org/vuls/id/749342. [Accessed: March 10, 2007].

[138] CERT, "Advisory CA-2004-01 Multiple H.323 Message Vulnerabilities," January 2004. [Online]. Available: http://www.cert.org/advisories/CA-2004-01.html. [Accessed: March 10, 2007].

[139] CISCO, "Cisco Security Advisory: Vulnerabilities in H.323 Message Processing," January 2004. [Online]. Available: http://www.cisco.com/warp/public/707/cisco-sa-20040113-h323.shtml. [Accessed: March 10, 2007].

[140] SecuriTeam, "Vulnerability n Microsoft Internet Security and Acceleration Server 2000 H.323 Filter Could Allow Remote Code Execution (MS04-01)," January 2004. [Online]. Available: http://www.securiteam.com/windowsntfocus/5BP0B15BPK.html [Accessed: March 10, 2007].

[141] International Telecommunication Union, "H.235: Security and encryption for H-series (H.323 and other H.245-based) multimedia streams," August 2003. [Online]. Available: http://www.itu.int/rec/T-REC-H.235/e. [Accessed: March 12 2007].

[142] J. Posegga, and J. Seedorf, "Voice Over IP: Unsafe at any Bandwidth?," in Proceedings of the Eurescom Summit 2005, 2005, vol. 27, pp. 305-314.

[143] S. Salsano, L. Veltri, and D. Papalilo, "SIP Security Issue: The SIP Authentication Procedure and its Processing Load," in Special Issue of IEEE Network, 2002, vol. 16, no. 6, pp. 38-44.

[144] P. Q. Qiu, O. Monkewich, and R. L. Probert, "SIP Vulnerabilities Testing in Sesson Establishment & User Registration," in Proceedings of ICETE 2004, 2004, vol. 2, pp. 223-229.

[145] The PROTOS Project, "PROTOS Test-Suite: c07-sip," June 2005. [Online]. Available: http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/index.html. [Accessed: March 11, 2007].

[146] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka, "RFC 2311 –S/MIME Version 2 Message Specification," March 1998. [Online]. Available: http://www.faqs.org/rfcs/rfc2311.html. [Accessed: March 12, 2007].

[147] T. Dierks, and C. Allen, "RFC 2246 – The TLS Protocol Version 1.0," January 1999. [Online]. Available: http://www.faqs.org/rfcs/rfc2246.html. [Accessed: March 12, 2007].

[148] R. Barbieri, D. Bruschi, and E. Rosti, "Voice over IPsec: Analysis and Solutions," in Proceedings of the Conference on Computer Security Applications, 2002, pp. 261-270.

[149] IKV++ GmbH, "Grasshopper Basics and Concepts Release 2.2," January 1999. [Online]. Available: http://www.ikv.de/content/Grasshopper/Grasshopper_e.htm. [Accessed: August 12, 2004].

[150] Email exchange with Grasshopper IKV Customer Service Representative, August14, 2004.

[151] IUT Vendor, "IUT Non Disclosure Agreement Toolkit Overview", January 14, 2005.

[152] IUT Vendor, "IUT Non Disclosure Agreement Toolkit Programmer Guide", January 14, 2005.

[153] Meth Labs, "PeerGuardian," April 2004. [Online]. Available: http://methlabs.org/projects/peerguardian-2-windows. [Accessed: Feburary 10, 2005].

[154] Phoenix Labs, "PeerGuardian 2," February 2006. [Online]. Available: http://phoenixlabs.org/pg2/. [Accessed: December 12, 2007].

[155] Phoenix Labs, "PeerGuardian 2: Manual," February 2006. [Online]. Available: http://wiki.phoenixlabs.org/wiki/PeerGuardian_2:Manual. [Accessed: December 12, 2007].

[156] Email exchange with IUT Vendor representative, April 08, 2005.

[157] R. S. Gray, D. Kotz, R. A. Peterson, Jr., J. Barton, D. Chacon, P. Gerken, M. Hofmann, J. Bradshaw, M. Breedy, R. Jeffers, and N. Suri "Mobile-Agent versus Client/Server Performance: Scalability in an Information Retrieval Task," in Proceedings of the Fifth IEEE International Conference on Mobile Agents, 2001, vol. 2240, pp. 229-243.

[158] Carnegie Mellon University, "Meet CERT," February 1995. [Online]. Available: http://www.cert.org/meet_cert/. [Accessed: December 15, 2007].

[159] Julia Allen, "CERT System and Network Security Practices," in Proceedings of the National Coolquium for Information Systems Security Education, May 2001. [Online]. Available: http://www.sei.cmu.edu/news-at-sei/columns/security_matters/2001/2q01/security-2q01.htm. [Accessed: December 15, 2007].

[160] PH. S. Teng, K. Chen, and S. C-Y. Lu, "Security Audit Trail Analysis Using Inductively Generated Predictive Rules," in Proceedings of the Sixth Conference on Artificial Intelligence Applications, 1990, pp. 24-29.

[161] C. Ko, G. Fink, and K. Levitt, "Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring," in Proceedings of the 10th Annual Computer Security Applications Conference, 1994, pp. 134-144.

[162] C. Ko, M. Ruschitzka , and K. Levitt, "Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach," in Proceedings of the 1007 IEEE Symposium on Security and Privacy, 1997, pp. 175-187.

[163] K. Deeter, K. Singh, S. Wilson, L. Filipozzi, and S. Vuong, "APHIDS: A Mobile Agent-Based Programmable Hybrid Intrusion Detection System," Lecture Notes in Computer Science, 2004, vol. 3284, pp. 244-253.

[164] K. Deeter, "APHIDS: A Mobile Agent-based Programmable Hybrid Intrusion Detection System," M.Sc. thesis, the University of British Columbia, Vancouver, BC, Canada, 2004.

[165] K. Singh, and S. Vuong, "BLAZE: A Mobile Agent Paradigm for VoIP Intrusion Detection Systems," in Proceedings of ICETE 2004 – First International Conference on E-Business and Telecommunication Networks, 2004, pp. 238-245.

[166] T. Aura, P. Nikander, and J. Leiwo, "DOS-Resistant Authentication with Client Puzzles," in Lecture Notes in Computer Science, 2001, vol. 2133, pp. 170-178.

[167] D. Dean and A. Stubblefield, "Using Client Puzzles to Protect TLS," in Proceedings of the 10th Conference on USENIX Security Symposium, 2001, vol. 10, pp. 1-1.

[168] R. Pethia, "Computers Under Attack: Internet Security Trends," Carnegie Mellon Software Engneering Institute presentation, 2001. [Online]. Available: www.hpcc-usa.org/pics/02-pres/pethia.ppt. [Accessed: June 20, 2006].

[169] R. Pethia, "Information Technology – Essential But Vulnerable: Internet Security Trends," Testimony Before the House Committee on Government Reform, Subcommittee on Government Efficiency, Financial Management and Intergovernmental Relations, November 2002. [Online]. Available: http://www.cert.org/congressional_testimony/pethia-11-02/Pethia_testimony_11-19-02.html. [Accessed: May 03, 2008].

[170] N. Feamster, "Security Problems with the Internet Architecture," Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. 6.829, 2002.