Computer Science Department
University of Crete

Prototype Deployment of the Network of Affined Honeypots Architecture

Master's Thesis

Manos Athanatos

October 2007
Heraklion, Greece

University of Crete

Computer Science Department

Prototype Deployment of the Network of Affined Honeypots Architecture

Thesis submitted by

Manos Athanatos

in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

THESIS APPROVAL

Author: _____

Manos Athanatos

Committee approvals: _____

Evangelos P. Markatos

Professor, Thesis Supervisor

_____

Mema Roussopoulos

Assistant Professor

_____

Maria Papadopouli

Assistant Professor

Departmental approval: _____

Panos Trahanias

Professor, Chairman of Graduate Studies

Heraklion, October 2007

Abstract

Internet security is an arms race between attackers and the whitehat community. Honeypots are a valuable asset that helps researchers and security experts obtain knowledge for both known and unknown attacks. Honeypots are specially designed hosts set up to act as decoys for attackers and automated attacking tools. Since a single honeypot host can only receive a small portion of the unsolicited internet traffic and thus has limited chances to be the victim of a cyber attack, a distributed monitoring infrastructure of cooperative honeypots is needed.

Network of Affined Honeypots (NoAH) aims to develop and deploy a large scale system of honeypots which cooperate in order to identify new and analyze existing attacks. NoAH is composed of a core that is comprised of low and high interaction honeypots, which are responsible for interacting with the attackers and analyze the received traffic. Apart from the core, there are tunneling and funneling mechanisms that collect and direct traffic from participating honeypots to the NoAH core. Moreover, home users and small enterprises can participate in NoAH infastructure through Honey@Home tool. Honey@Home listens to the unused address space of a home or enterprise network and forwards received traffic to the NoAH core. In this thesis we present a prototype deployment of the NoAH infastructure along with some experimental results and record our experiences with the NoAH infrastructure. Results suggest that we can build a robust and effective infrastructure with moderate resource requirements.

Supervisor: Professor Evangelos Markatos

Περίληψη

Η ασφάλεια υπολογιστών είναι ένας αγώνας μεταξύ των επιτιθέμενων και τις ερευνητικής κοινότητας που ασχολείται με την ασφάλεια υπολογιστών. Τα "Honeypots" (συστήματα ανίχνευσης επιθέσεων) είναι ένα πολύτιμο εργαλείο που βοηθάει τους ερευνητές και τους ειδικούς στην ασφάλεια υπολογιστών να κατανοήσουν καλύτερα τις ήδη γνώστες και να ανιχνεύσουν προτύτερα άγνωστες επιθέσεις. Τα "Honeypots" είναι ειδικά σχεδιασμένα συστήματα τα οποία λειτουργούν σαν δόλωμα για τους επιτιθέμενους και για τα αυτόματα προγράμματα επιθέσεων. Ένα μόνο "Honeypot" μπορεί να λάβει ένα περιορισμένο μέρος της 'ύποπτης' κίνησης που κυκλοφορεί στο διαδίκτυο και έτσι έχει περιορισμένες πιθανότητες να πέσει θύμα μιας επίθεσης μεσώ του διαδικτύου. Γι' αυτούς τους λόγους χρειάζεται ένα κατανεμημένο σύστημα από πολλά συνεργαζόμενα "Honeypot".

Το έργο NoAH (Δίκτυο από συνεργαζόμενα "Honeypots") στοχεύει στην σχεδίαση και ανάπτυξη ενός κατανεμημένου δικτύου από συνεργαζόμενα "Honeypots", το οποίο προσπαθεί να ανακαλύψει καινουργίες και να αναλύσει τις ήδη υπάρχουσες επιθέσεις. Το NoAH αποτελείται από ένα 'πυρήνα' από "Honeypots" χαμηλής και υψηλής αλληλεπίδρασης, τα οποία επικοινωνούν με τους επιτιθέμενους και αναλύουν τη κίνηση που λαμβάνουν από αυτούς. Εκτός από τον 'πυρήνα', στο NoAH υπάρχουν διάφοροι μηχανισμοί στο οποίοι λαμβάνουν και προωθούν τη λαμβανόμενη κίνηση στο πυρήνα του συστήματος. Επιπλέον, οι απλοί χρήστες και οι μικρές επιχειρήσεις μπορούν να λάβουν μέρος στο σύστημα NoAH χρησιμοποιώντας το πρόγραμμα "Honey@Home". Το πρόγραμμα "Honey@Home" λαμβάνει τη κίνηση που πρωτύτερα κατευθύνονταν σε ανενεργές IP διευθύνσεις. Αυτή η κίνηση συλλέγεται και προωθείται στον 'πυρήνα' του NoAH από το πρόγραμμα "Honey@Home".

Σε αυτή την εργασία παρουσιάζουμε μια πρωτότυπη ανάπτυξη της αρχιτεκτονικής NoAH. Επίσης, παρουσιάζονται τα αποτελέσματα από μια σειρά πειραμάτων τα οποία έγιναν με σκοπό τη μέτρηση της απόδοσης του συστήματος NoAH. Τα αποτελέσματα αυτά δείχνουν ότι είναι εφικτό να δημιουργήσουμε μια εύρωστη και αποδοτική υποδομή από "Honeypots" με τη χρήση περιορισμένων πόρων.

Επόπτης: Καθηγητής Ευάγγελος Μαρκάτος

Acknowledgments

I feel grateful to my Supervisor, Professor Evangelos P. Markatos, for his valuable assistance and guidelines in my academic steps in the field of Computer Science. I am really privileged to cooperate with people of his mental and ethical values.

My best regards to the following, past and current, members of the Distributed Computing Systems Laboratory (ICS/FORTH), whom I feel to be friends more than just colleagues: Spiros Antonatos, Michalis Polychronakis, Demetres Antoniades, Elias Athanasopoulos, Antonis Papadogiannakis, Christos Papachristos, Periklis Akritidis, Manolis Stamatogiannakis, Demetres Koukis, George Vasiliades, Polakis Iason, Nikos Hatzibodozis, Michalis Foukarakis, John Velegrakis and Manos Moschous.

I would like to thank my family, my parents George and Helen and my sister Marina, for the support and education they provided me with.

Last but not least, I would like to thank Eirini for her emotional support.

To my Family.

Contents

List of Figures

List of Tables

1

Introduction

Honeypots are specially designed systems that their sole purpose is to lure attackers and automated attacking scripts. These systems can take many different forms like machines running on unused IP addresses, unprotected data records or files. A honeypot is valuable as a surveillance and early-warning tool. As honeypots are not production systems, all traffic seen by them is considered suspicious and maybe malicious. Furthermore, honeypots should be treated with caution as it is expected to be compromised with the danger to contaminate their local network.

There are two types of honeypots based on the interaction they can provide to the attacker; The low interaction (LI) honeypots and the high interaction (HI) honeypots. The low-interaction honeypots emulate the basic

functionality of a some known services and interact with the attacker up to a certain level without allowing the honeypot to be compromised. Since LI honeypots can not be compromised they are unable to automatically detect previously undiscovered vulnerabilities. High interaction honeypots run real services and applications, usually over a virtual machine or some other containment environment. Therefore high interaction honeypots can be compromised and thus provide useful information about previously unknown vulnerabilities or about new attack vectors based on already known vulnerabilities.

NoAH aims to develop an infrastructure of collaborating honeypots with multiple distributed monitoring sensors in order to detect and provide early warning of cyberattacks either new or existing. In the context of NoAH architecture [4] both types of honeypots are used. Low-interaction honeypots are used as a first level traffic classifiers reducing the amount of traffic that finally reaches the high interaction honeypots inside the NoAH core. Also funneling and tunneling mechanisms are used to forward traffic received by participating organizations' and institutes' black IP address space to the NoAH core. Finally in order to motivate normal users to join NoAH infrastructure, Honey@Home tool has been introduced which allows ordinary users to redirect traffic destinted to the dark IP address space or unused port range.

Research work such as [9] and [2] report a shift in the landscape of threats. Attackers have moved from massive wide spread attacks to more focused ones, reaching the point of targeting client side applications. Also with the massive deployment of broadband connections, personal computers have became a valuable asset for the creation of bot nets, armies of zombie PCs, which can be used to launch Denial Service Attacks (DOS) attacks, promote spam mail etc. Based on the previous remarks and in order to increase our probabilities of witnessing an attack, a distributed

architecture with multiple sensors, like NoAH should be introduced. NoAH consists of multiple distributed sensors that redirect "interesting" traffic to the NoAH core for further processing.

In this work we performed a prototype deployment of the NoAH architecture and evaluated its performance under various set ups. We show that a distributed system based on honeypots like NoAH, can be used in real life situations in order to provide precise detection of attacks and early warning for previously unknown attacks with moderate resource requirements.

1.1 Contributions

Our novel contributions in this thesis are the following:

- Prototype deployment of the NoAH architecture
- Evaluation methodology and experimental results for the NoAH architecture are provided

1.2 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 explores in detail the various honeypots categories along with various distributed architecture designs for honeypot systems that have been proposed. Chapter 3 discusses the NoAH honeypot architecture which will be evaluated in this study. Chapter 4 presents the evaluation methodology that will be followed for the in depth evaluation of NoAH architecture, this evaluation takes into consideration both the various components of NoAH independently and the NoAH system as a whole. Chapter 5 outlines our experimental results. Finally Chapter 6 concludes the thesis and presents few remarks for future work that could be done in this area of study.

2

Background

In this chapter the necessary background information on the taxonomies of honeypots and the various proposed architectures for large scale distributed honeypots systems will be presented. Furthermore, some widely used tools for setting up honeypots along with a short evaluation of each tool and each honeypot architecture is also embodied.

2.1 Honeypots

Honeypots are systems that act as traps set to detect, deflect or in some manner counteract attempts at unauthorized use of information systems. A honeypot Commonly consists of a computer which runs a number of network services that appears to be part of a network but which is actually isolated, (un)protected and monitored, and which seems to contain infor-

mation or resources that would be of value to attackers. A honeypot can also masquerade to be an open proxy which can be used to send usolicited emails or conduct other malicious activities. Generally all traffic that arrives to a honeypot is considered to be either malicious or due to systems misconfigurations.

Honeypots are divided into two categories based on their level of interaction with the attacker: The low interaction (LI) honeypots and the high interaction (HI) honeypots. The low-interaction honeypots are mostly daemons that emulate the basic functionality of a some well known Internet services e.g. Web, FTP, Telnet servers and are able to interact with the attacker up to a certain level without allowing the honeypot to be compromised. On the other hand, high interaction honeypots are closely monitored hosts running real services that can be compromised. High interaction honeypots provide better level of realism than the low interaction honeypots but have higher setup and maintenance costs. A third type of honeypot that is not widely used is the middle interaction honeypots which resembles low interaction honeypots but provide more elaborate emulation of services.

2.1.1 Low Interaction Honeypots

As aforementioned earlier Low Interaction honeypots interact with the attacker by emulating some well known services. These services could be part of an Operating System such as daemon processes and other core services or could also be high level applications like web servers, ftp servers, ssh daemons and many more. Since the real services are not running on low interaction honeypots, they can not be compromised and thus can only provide limited information about an attack. Moreover, low interaction honeypots are unable to discover previously unknown attacks or vulnerabilities since they only emulate a small part of a service and do not posses the full functionality and the details of the real service. Low interaction honeypots can usually emulate the whole TCP/IP stack of any operating system and

thus, are able to pretend that any Operating System is running by emulating the appropriate TCP/IP network stack implementation. Therefore, most network scanners like nmap [28] which identify the OS running on a machine based on the diversification in the implementation of the TCP and IP RFC from every OS, can easily be abused. Also, low interaction honeypots are more efficient in terms of performance than the high interaction ones and have less set up and maintenance costs. Finally, low interaction honeypots are able to claim multiple IP addresses and run multiple different services on each IP address by emulating the appropriate Operating System with the corresponding services.

The most widely used low interaction honeypot implementation is Honeyd [46]. Honeyd is a framework for virtual honeypots which can simulate different systems up to network level. This allows Honeyd to be able to emulate any operating system and thus to deceive most fingerprinting tools available. Honeyd emulates a single or multiple machines with different properties and connected to an arbitrary network topology using the unused IP address space of a network. Honeyd also, uses scripts for the emulation of the services running on each of the "virtual" hosts. This allows any user of Honeyd to write and use her own scripts. This makes honeyd expandable to new services that were not available at the time honeyd was developed. The level of emulation details of a running services depends on the running script, since it can emulate from a simple transport layer responder up to complicated services like rpc.

One major drawback of honeyd is that each of the scripts must be written manually. Scriptgen [41] aims to automatically generating scripts for Honeyd. In order for scriptgen to create a new script three steps must be completed. Firstly a real machine is connected to the Internet and all the inbound and outgoing traffic from that machine is recorded. Secondly the trace of the traffic is analyzed and a per port state machine is derived by

observing the requests and replies to and from the host. Finally a honeyd script is derived that can emulate the services of that machine, based on the previously derived state machine.

Another type of low-interaction honeypots are the tarpits or "sticky honeypots". Tarpits emulate a number of "virtual" hosts in the unused address space of a network. These "virtual" hosts answer to connection attempts in a way that make the machine on the other end to get "stuck". So the project's main goal is to be able to constrain the propagation of fast spreading worms like CodeRed and to make the life of attackers harder. The most well-known tarpit is being developed by the "LaBrea" honeypot project [6].

The Google Hack Honeypot - GHH [15] is being developed in order to counter a new type of malicious web attackers, the search engine hackers. The GHH is a tool that can provide valuable information about attackers that use search engines as a hacking tool against network resources. Authors of [43] demonstrate how attackers can easily identify vulnerable services on the Internet with the use of indexing services like Google search engine. These services can be exploitable messaging boards, web servers, or other network applications. These insecure tools, when combined with the power of a search engine and indexing which Google provides, results in a convenient attack vector for malicious users.

Honeytrap [5] is a network security tool written to observe attacks against network services. The Honeytrap daemon is able to detect requests to unbound TCP ports, and then it starts a server process to handle the incoming connection to that port. In this way Honeytrap doesn't need to keep thousands of ports open in order to capture a possible attack but instead it uses different "connection monitors" to detect new connection attempts. Honeytrap has limited emulation capabilities but different plugins can be used to enhance the emulation of more complex vulnerabilities that require multiple steps before the exploitation.

The Mwcollect tool [32] that was merged within Nepenthes [33] and are currently part of the Mwcollect alliance [20] is a medium interaction honeypot with the purpose to emulate known vulnerable parts of services and to collect different malware samples. Once the URL hosting the malicious code if discovered the malware is downloaded through FTP, HTTP or some other protocol and stored locally. Since nepenthes only emulates the vulnerable part of a service, it is more efficient than low interaction honeypots. With the deployment of multiple nepenthes sensor the collection of information about fast spreading malware becomes more efficient.

2.1.2 High Interaction Honeypots

High Interaction honeypots are usually closely monitored real systems running real services. Contrary to low interaction honeypots emulation is not needed since real services are running. High interaction honeypots could either, run vulnerable services so that they could provide useful information about how an attacker or a worm acts when it compromises a machine or could run services that haven't been exploited in the past and provide useful information about previously unknown vulnerabilities and attack vectors. Using real systems as HI honeypots has some major drawbacks such as the fact that the systems can be compromised and used for malicious purposes by the attacker. Therefore, high interaction honeypots are usually installed inside a virtual machine using virtualization software like VMware [11], Qemu [7], Xen [10] or Bochs [1]. Using virtual machines for HI honeypots has many advantages such as easier deployment, reuse of the same operating system image by many honeypots, rollback capabilities in case the honeypot gets infected and running multiple honeypots on top of the same hardware. Finally, containment environments like Argos [44] or other tainting environments can be used in order to provide additional details for the internals of the system when an attack is in progress.

Vmware [11] , Qemu [7] and Bochs [1] are some popular virtual ma-

chines that are used for the setup of high interaction honeypots. They provide virtualization of the hardware and hide the underlying operating system details. The virtualization only provides the means for creating multiple instances of any operating system and running them on different machines and operating systems. Therefore, in order to monitor the activity of the high interaction honeypots, other tools has to be used as well like Sebek [8], Systrace [45], Wireshark [30] and others. Thus, the aforementioned virtual machines should be deployed in conjuction with tools that provide useful information about the state of the honeypot like network activity, system call traces, disk and file accesses. The information that denotes the state of a honeypot are vital for the analysis of possible attacks it may receive.

The main drawback in using real operating systems or systems running over virtual machines is that they can get compromised and manual actions are then needed in order to rollback to the previous state. Moreover, as we mentioned earlier third party tools are needed in order to closely monitor and track any changes in the state of the HI honeypot. Argos [44] is a virtual machine but also acts as a containment environment. Argos is built upon a fast x86 emulator and is able to track network data throughout execution to identify their invalid use such as jump targets, function addresses, instructions, etc. Argos does not allow these “tainted” data to be executed and thus the HI honeypot running over Argos can not be compromised but instead a memory dump of the “tainted” data is produced and the process that was about to be compromised is restarted. So using Argos as HI honeypots overcomes the major problem of real systems or systems running over the classic virtual machines which is that they can be compromised. On the other hand Argos is less effective than the usual virtual machines in terms of performance.

2.2 Distributed Honeypot Architectures

Attacks no longer utilize generic global propagation methods but have adopted more sophisticated host specific propagation techniques we need efficient systems for the fast detection and interception of previously known and new attacks. In this context, a single honyepot could provide limited or no protection at all against attackers. Distributed honeypots architectures should be deployed to increase our detection potentials even against targeted attacks. Many research projects have deployed multiple sensors globally aiming to collect statistics about attack traffic, provide early detection about novel attacks or collect various malware samples. The most well known distributed honeypot architectures are discussed in the remainder of this section.

The Honeynet [3] project is a non-profit research organization targeting to raise the awareness of the threats and vulnerabilities that exist in the Internet, to provide information to secure and defend against attacks as they arise and to provide tools that can be used for the protection and detection of new and existing attacks. Honeynet project architecture consists of a central gateway named "Honeywall" [16], and a network of honeypots. Honeywall processes the traffic targeted to their honeypot network from the rest of the Internet and reversely. The honeywall also controls outbound connections of the honeypot network and captures the network data that is transmitted. The high interaction honeypots that are used by Honeynet are real systems without any emulation. In addition, Sebek tool [8] is used to instrument and record the honeypots' system calls.

The LUERRE.COM project [18] has deployed a large number of low interaction honeypot sensors all over the world. These low interaction honeypots run a modified version of honeyd [46] that emulates three different operating systems (Windows NT, Windows 98, and Redhat linux), each one running on three different IP addresses. These low interaction honeypots gather sta-

tistical data about the traffic destined for the to honeypots like number of unique IP addresses sources, distinct ports attacked and specific flags that could help the research community to identify backscatter or other similar activities. Periodically, the collected data gets anonymized and send to a central database server where are stored along with some meta data. The meta data that are stored contain geographical location information about IP addresses of the attackers, passive OS fingerprinting of attackers and the method the attacker used to scan the honeypots "sequentially" or "parallel". The database can then be queried in order to derive some meaningful statistics from the stored data.

The Honeystat architecture [36] aims to detect and analyze zero-day worms along with previously unknown attack vectors. Also the authors of this work are aiming to apply statistical analysis on the worm behavior. Honeystat architecture suggests having a central server and multiple distributed Honeystat nodes. The Honeystat nodes comprises of honeypots that run multiple emulated operating systems over VMware. A set of 32 IP addresses is assigned to each of the emulated operating systems. There are three Honeystat events that denote that a honeypot has been compromised: *MemoryEvent* when a buffer overflow has occured which is detected by buffer overflow detection tools like stackguard [35] , *NetworkEvent* when the honeypot starts producing outgoing traffic and *DiskEvents* when the honeypot tries to alter certain key files of the operating system. If a node gets compromised all data is captured and sent to the central "Analysis Node" for further analysis of the attack details.

Potemkin [42] architecture introduces a prototype implementation of a large honeyfarm system that is capable of emulating thousand of hosts in parallel. The main building block for high interaction honeypots in potemkin is XEN [10], a virtual machine monitor that uses para virtualization. The authors also introduce the use of flashing cloning and delta

virtualization to enhance performance and manage to emulate as many virtual hosts on top of few real machines. Recycling is used the virtual honeypots so to minimize the consumption of resources by the honeypots. Moreover a gateway component that is contained in the architecture that attracts inbound traffic and redirects the outbound traffic to another honeypot of the honeyfarm producing a realistic environment for the attacker.

M. Bailey et al in [34] describe a honeypot architecture consisting of a set of low interaction and high interaction honeypots. In the context of this hybrid architecture low interaction honeypots are used to filter out the *harmless* traffic such as plain SYN scans or unestablished TCP connections or payloads that have been recorded in the past. A hand off mechanism redirects interesting traffic to the high interaction honeypots which interact with the attacker. The hybrid architecture also contains a control component that is used to collect and present traffic statistics from the low interaction honeypots and analyzes the received data for abnormal behavior. Many ideas proposed in this architecture are also present in the NoAH architecture.

Collapsar [48] aims at the deployment and management of a vast number of coordinating high interaction honeypots that may be spread across various network domains. Collapsar consists of a redirector, a front-end component and the virtual honeypot component. The redirectors are sensors that are deployed in various network domains and are used to redirect the traffic to the honeypot component through the Collapsar front end. The front-end is a gateway to the Collapsar center. It receives packets from redirectors which are then dispatched after some processing to the intended virtual honeypots in the Collapsar center. The redirector is implemented as a virtual machine based on the User-Mode Linux (UML [37]). Inside Collapsar center, a large number of virtual honeypots exist. These honeypots process the traffic received from distributed redirectors and respond ac-

cordingly. The high interaction honeypots that exist in the Collapsar center are built on either VMware or UML.

2.3 Spam Honeypots

E-mail spam grows every year and reached 80% of email traffic, for the last quarter of 2005, as the Messaging Anti-Abuse Working Group report [31] suggests. Spammers used to utilize "open relays" [23] to send unsolicited messages without leaving any traces behind. Open relays usually are SMTP servers that can be used by anyone on the Internet to send email messages. Because, open relays have became more and more rare the attackers have turned their attention to "open proxies" [22]. Open proxies allow anyone from the Internet to use them in order to connect to a server e.g. web, email, ftp. Therefore, taking advantage of this functionality, the identity of the client using an open proxy is hidden. In 2003 Spammers turned to a new method for sending bulk e-mail the "spammer viruses". These viruses usually spread through e-mail messages which prompt the user to open a specific attachment that contains the virus. Once the virus has compromised the PC, tools get installed which can be used for sending spam e-mails like open proxies or other back doors that can be used by attackers to send unsolicited e-mail messages. Known spammers' viruses are Sobig [26] and [13].

In order to detect and contain as much of this bulk traffic that circulates the Internet as possible, the "Spam honeypots" have been introduced. Spam Honeypots are usually machines that run software which can be used by spammers to send bulk e-mails like open mail relays and open proxies. If an attacker tries to connect to a honeypots all traffic sent is captured and no spam e-mails are sent. Moreover, analysis of captured data can produce useful information about the spammers. Some well known open mail relay honeypots, that are used in order to trap spammers and drop the unso-

licited e-mail that are trying to be sent, are Jackpot [17] and spamhole [25]. A known Open proxy honeypot project that aims at the deployment of many distributed open proxy sensors is the "WASC Distributed Open Proxy Honeypots" project [29]. Another project that aims at the creation of lists that contain IP addresses of spammers, crawlers or spiders is the "Project Honey Pot" [24]. Project Honey Pot is based on a distributed network of decoy web pages, that website administrators can include on their sites in order to gather information about robots, crawlers, and spiders. Besides the decoy web pages their honeypot sensors also contain a spam trap that can be used to trap spammers and blacklist their IP addresses for a short period or permanently.

3

NoAH architecture

Network of Affined Honeypots (NoAH) aims to develop and deploy an infrastructure of cooperating honeypots that will be able to gather and analyze information about Internet cyberattacks. The infrastructure will also be able to produce early warning for previously unknown attacks, so that appropriate countermeasures can be taken to counter these new attacks. The NoAH architecture [4] comprises of a NoAH core, where the Low and High interaction honeypots reside and various *funneling* and *tunneling* mechanisms that are used to redirect traffic to NoAH core from various distributed sensors or participating organizations. Figure 3.1 illustrates the basic components of NoAH architecture. We can observe that both Low and High interaction honeypots are utilized but for distinct purposes. Tunneling

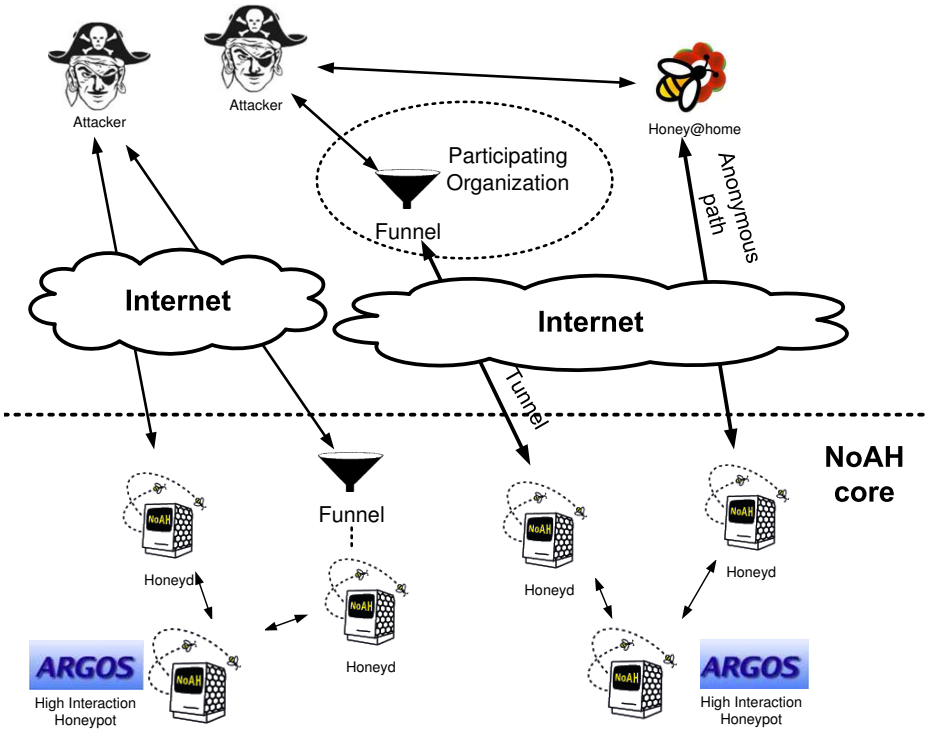

FIGURE 3.1: The overall architecture of NoAH. There are three entry points to the system: Through Low interaction honeypots, through the dark space of cooperating organization or through Honey@Home tool that runs on homes/small enterprises' dark space. At the two latter cases the traffic going to dark space is redirected to NoAH core.

and funneling mechanisms have been proposed by the NoAH architecture for the achievement of two major goals. *Funneling* serves the collection of traffic targeting the dark IP addresses space of an organization in an automated way. *Tunneling* is used to propagate the traffic collected from the Honey@Home tool or the funneling mechanism into the NoAH core. Finally the Honey@Home tool [47] has been introduced to allow a single user to donate a spare IP or port to NoAH infrastructure from its home PC or local network. The following sections describe each of the NoAH components in

more details.

3.1 Tunneling and Funneling

Generally a broad portion of the IP addresses of a network that is allocated to an organization remains unused. This unused IP address space is usually referred to as *Dark Space*. NoAH aims in utilizing the dark space of the participating organizations or the small enterprises by collecting the traffic bearing towards their dark space and redirect it to NoAH core for further processing. Since we can't afford a honeypot listening to every IP address of the dark space we need a mechanism to collect traffic from a larger portion of the dark space to a small set of honeypots. This can be achieved through the NoAH *funneling* mechanism. Moreover, since the traffic bears towards different dark spaces scattered across many organizations, we need a mechanism to redirect this traffic to NoAH core over a secure channel. For that purpose a *tunneling* mechanism has been introduced.

The *Funneling* mechanism will be used by organization or medium enterprises which have numerous IP addresses spare. In a nutshell *funneling* allows NoAH to redirect all traffic destinated to multiple addresses to a single machine where it can be processed. Thus we can create multiple *funnels* that collect traffic from various portions of dark space from different participating organizations to a small set of honeypots. Funneling is a two step process. First, the traffic arriving to the dark space is redirected to the honeypot location within the participating organization by configuring the organization's router(s) appropriately. Secondly, honeypots claim the dark IP addresses that are configured to funnel. This can be achieved either by configuring the honeypots to listen to all the dark IP addresses that are responsible for or by using the *arpd* tool.

Statically configuring every honeypot to listen to certain dark IP addresses is a time and resource consuming process, so in NoAH the *arpd*

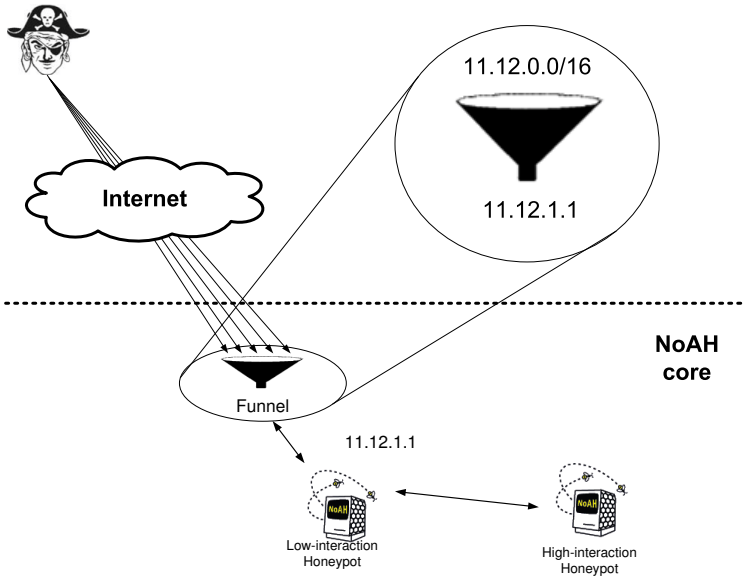

FIGURE 3.2: Funneling IP addresses from 11.12.0.0/16 subnet to a single low-interaction honeypot

daemon is used instead. *Arpd* is a user-space daemon that responds to ARP requests arriving to the network interface of the honeypot for the IP addresses that is configured to respond. ARP requests are broadcast packets used to discover which machine (more specifically which MAC address) has a specific IP address. For a given IP address in a LAN, the LAN gateway directs the traffic to this IP to the host that replied to the corresponding ARP request. Under normal circumstances, it is the operating system that takes care of responding to ARP requests. Having arpd reply to these requests, we can effectively direct the traffic for any IPs in the LAN to the particular host we want, without actually configuring the host network interface for all these addresses.

Funneling is presented in Figure 3.2 where we can see that traffic targeting the /16 11.12.0.0 subnet is funneled to a single IP address 11.12.1.1. Arpd replies to every ARP request for any of the IP addresses in the /16

11.12.0.0 subnet. Therefore, the LAN gateway forwards the traffic for this IP range to the honeypot. Cooperating organizations will act as relay agents that forward traffic arriving to their dark address space, to NoAH core. Sequentially, the responses from NoAH core will be transmitted back to the agent and from there to the original sender. In order to achieve this kind of functionality, one could rewrite the packets' destination addresses so that to be correctly routed to NoAH core and rewrite accordingly the responses sent by the NoAH core back to the original sender. This technique although its simple and straight forward, it has some major drawbacks like disrupting the dynamics of the connections e.g a scan to a number of IP addresses in the dark space will appear to the honeypot that it targets a single IP; also protocols like FTP that have information about the destination address, within the TCP payload will not work correctly under this schema because the translated address will not match the one that appears in the TCP payload.

Consequently, we need packets that arrive to dark space to be forwarded to the NoAH core unaltered. *Tunneling* was introduced into NoAH architecture to overlap that particular problem. Tunneling works in the following way; every packet that is received to the dark space of a particular organization or any other virtual private network is encapsulated into one or more packets and subsequently sent to the NoAH core. There, the packet is decapsulated and injected to the network where it can be processed by the low and high interaction honeypots. The response from the honeypots follows the reverse procedure: it is encapsulated inside the NoAH core and decapsulated at the entry sensor node where the request packet was initially received. We should note that tunneling fully preserves the packet and does not alter its contents throughout the encapsulation/decapsulation process.

Figure 3.3 presents how tunneling works in the funneling context. A network funnel concentrates every packet that arrives to the dark space

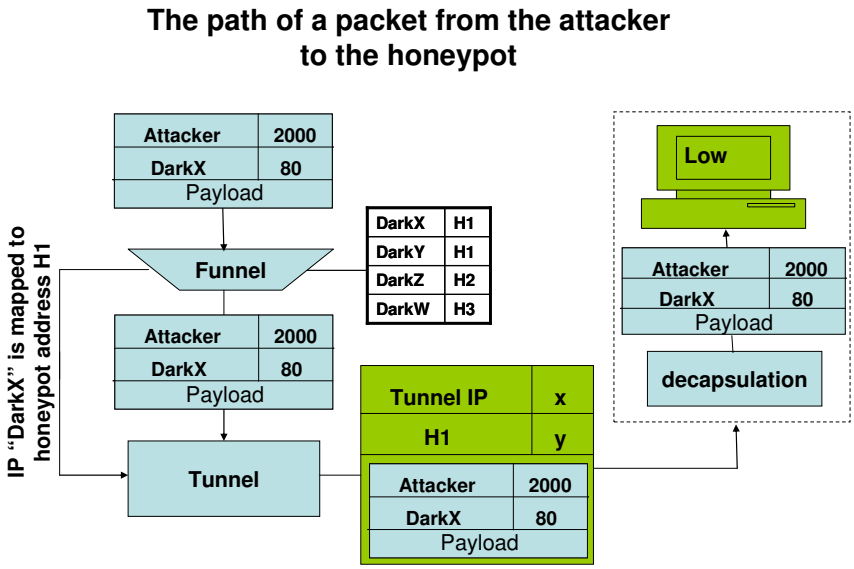

FIGURE 3.3: Funneling and tunneling. A packet from the attacker destined to darkX IP address is tunneled to honeypot H1.

monitor and then it is encapsulated and sent over the tunnel encrypted using the SSL protocol [40]. When it arrives at NoAH core, it gets decapsulated and injected into the network where it is processed by the low interaction honeypots. The low interaction honeypots can either process the request locally or forward it to one of the high interaction honeypots. Therefore, a packet from the attacker that is destined to IP DarkX in this range will be delivered to the tunnel component, that has the responsibility to forward the packet to the honeypot with IP H1. The tunnel component encapsulates the original packet to a packet destined to IP H1 and sends it to the Internet. As the destination IP of this packet is H1, it will be routed normally from Internet routers and will finally reach the low interaction honeypot at IP

H1. There it will be decapsulated and the honeypot will receive the original attack packet unmodified. The responses from the honeypots will follow the extact same procedure for the reverse path though.

3.1.1 Honey@Home

In order to expand the visibility of NoAH into more unused IP space, that home users and small enterprises can provide, the Honey@Home tool [47] was proposed. With the vast propagation of broadband Internet, most home users and small enterprises can easily acquire more than one IP addresses which usually remain unused. Especially enterprises of small or medium size usually have a cluster of four or eight IP addresses which are not fully used. Like other "@home" approaches Honey@Home is aiming in utilizing the "idle" IP space of a home user or small enterprises. Honey@Home is designed to be simple and lightweight so that it can be used by any typical home user. Honey@Home has been implemented to be non-pervasive and it runs in the background processing the traffic arriving on the dark space. Honey@Home tool works both on Windows and Linux operating systems.

Honey@Home [47] works in a similar way as the case of dark space monitoring with funneling and tunneling concepts. The difference between funneling and honey@home tool is that we only have a very limited number (usually just a single) of IP addresses to monitor in the case of Honey@Home whereas the dark space monitored using the funneling concept can expand to several thousands unused IP addresses. Similarly to the funneling case, all the traffic received by the Honey@Home is tunneled to NoAH core over an SSL connection. Responses from NoAH core are send back to Honey@Home client where are injected to the network so that they can reach the originator of the traffic. The tunneling component is embedded inside Honey@Home and thus, there is no need for installation and configuration of third party software. Figure 3.4 presents the overall design of the Honey@Home sensor. An IP address is claimed through the DHCP server and all traffic destinated

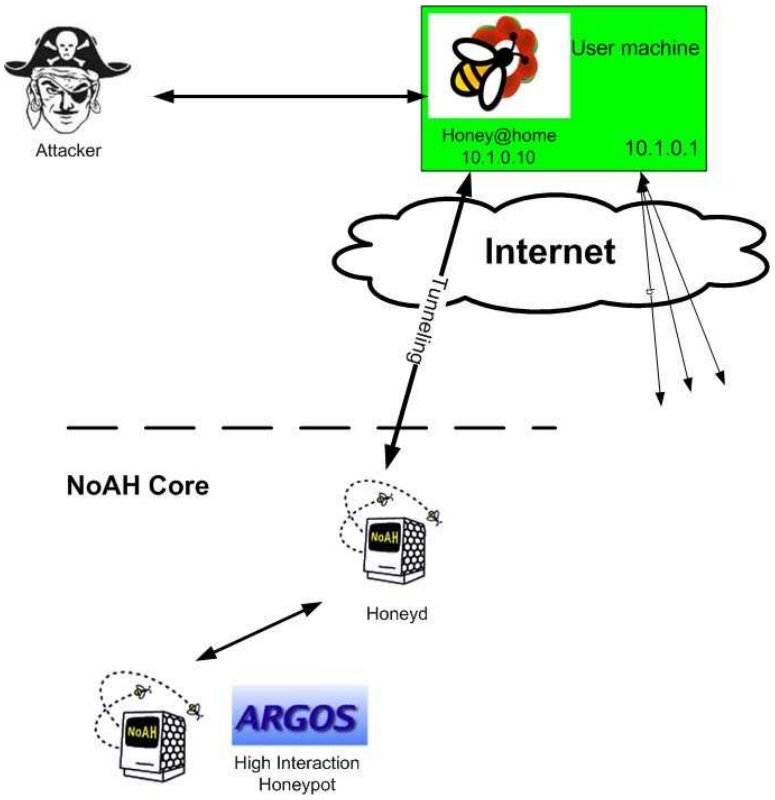

FIGURE 3.4: An overview of the design of Honey@Home. An IP address is claimed by Honey@Home which runs on the user's machine and redirects all the traffic received to that IP to the NoAH core the responses follow the same path back to the attacker.

to that IP address is forwarded to the NoAH core over an SSL tunnel. The part of the Honey@Home that is located within the honeypot network is usually referred as SSL Server or Honey@Home Server and is responsible for collecting traffic from the various Honey@Home sensors decapsulating and injecting packets in the network where low and high interaction honeypots reside.

Every Honey@Home client requests an IP address from the local network using DHCP. If such a server is not running in the network Honey@Home

can be configured to acquire an IP address statically. The majority of the ADSL routers that are used have a built-in DHCP server which is configured to offer the IP addresses that the user has purchased. Thus every Honey@Home client requests a new IP address upon start and it releases it at the time it exits. We should note that Honey@Home is a lightweight process since it does not process the traffic it receives even if that traffic is naive, the honeypots of the NoAH core will process it and determine whether it requires further investigation or can be dropped safely. In order to reduce the traffic arriving to NoAH core, Honey@Home can be configured to white list certain ports to not forward traffic arriving to these ports. By default all the traffic collected by Honey@Home is forwarded to the NoAH core. Honey@Home can also work in a NAT'ed environment but it will only be able to receive traffic originating from the hosts behind NAT and the traffic that NAT is configured to forward to the machines.

Since everyone can download and install Honey@Home we expect that attackers will do the same. Revealing the information about the location of honeypots can result to a DDOS attack against our NoAH core rendering it unable to react. Hiding the identity of the Honey@Home clients is also crucial as attackers could cause an indirect DOS attack to the NoAH core or could manipulate the Honey@Home clients so that they send fake information to the NoAH core or could black list all the Honey@Home sensors rendering the NoAH core blind to ongoing attacks. In order to hide honeypots from the attackers using TOR network [38] has been proposed. Tor is based on onion routing [39] anonymous networks where a message enters the network from an end node, gets encrypted by several layers of encryption and then it is propagated from node-to-node in a pre-established path. Each node can decrypt, "pill off", only one layer of encryption. When the message arrives to the final node the last layer is removed and the packet gets fully decrypted and sent to its destination the responses use the same

path back to the originator. Since no node of the network have the knowledge of the whole path it can not reveal who is communicating with whom. Figure 3.5 presents an overview of how tor works and how the packet is decrypted while it travels from the one node to the other. Then at the final node it gets decrypted and sent to its final destination.

3.2 Low Interaction Honeypots

Low interaction honeypots are software that can emulate a home PC, workstation or a sophisticated server. Usually low interaction honeypots can also emulate the TCP stack of any operating system so it can appear to the attackers as any of the operating systems that is emulated. The most widely used tool for the creation of low interaction honeypots is Honeyd [46], which is a small daemon that creates virtual hosts in a network. It can be configured to emulate that arbitrary services are running on a variety of different systems over multiple IP addresses. Each emulated service is a script that is loaded by Honeyd when it starts; moreover a user could create her own script and emulate any service proprietary or not. Low interaction honeypots of NoAH will be principally based on honeyd because of its functionality, modularity and portability. Honeyd can run both on Windows and Linux environments but in NoAH the Linux version of Honeyd is mainly used due to its performance eminence over the Windows one.

Figure 3.6 presents an overview of how Honeyd works. Every incoming packet is dispatched to the correct protocol handler. The handler performs the TCP stack emulation and propagates the packet to the emulated services' scripts e.g packets with destination port 80 are handed over to the script that emulates a web server, packets for port 23 to a script that emulates telnet etc. All outgoing packets are initially processed by the personality engine in order to mimic the behavior of the configured network stack. With the personality engine of Honeyd, one could configure each of

the virtual machines to a certain behavior eg “this IP address belongs to a virtual machine running Windows 2000 operating system”. We should remind that low interaction honeypots only emulate services and cannot be compromised since real services are not running.

As we have mentioned earlier, low interaction honeypots can emulate a specific set of application and services. Therefore, we need to provide a suitable configuration for the emulated machines in the network before it is initiated. Since using the same configuration on multiple honeypots is feasible, one could record the services that typically run on variety of operating systems upon start and use them as a guideline to provide configurations for our low interaction honeypots. For example one could record which services are enabled in a fresh Windows XP machine and emulate the same services on our low interaction honeypots. Moreover, one could create arbitrary network configurations with a mix of different operating systems appearing to belong in the same network. While creating a configuration we need to make sure that it does not appear anomalous eg “a Linux machine running IIS” or “a /24 subnet in which every machine runs Microsoft SQL server”. Typical views of services running on a subnet can be obtained by actively scan several live subnets.

3.3 High Interaction Honeypots

High interaction honeypots are usually ordinary machines that run real services or systems running on virtual machines or containment environments. Inside NoAH core there will be a number of high interaction honeypots that will run a variety of operating systems and services. There are a number of virtual machines that can be used to host operating systems and different services. Most virtual machine environments do not provide logging facilities like tracking system calls, network activities or disk accesses. So tools like Sebek [8] can be used to enhance plain virtual machines with logging

capabilities.

Since NoAH aims into receiving, analyzing and providing reaction to new attacks, we expect our High interaction honeypots to receive a large number of cyberattacks and other malicious activity. Therefore, in NoAH Argos [44] containment environment will be used for building our high interaction honeypot. Argos provides buffer overflow attack detection and protection by tracking the propagation of network data inside the system and producing an alert when these "tainted" data are about to be executed thus the high interaction honeypot can not be compromised. When an attack is detected, both a memory dump of the attack vector is produced along with a dump of the network activity which triggered the attack.

3.4 Interaction between Low and High Interaction Honeypots

As we have described previously in NoAH the various funneling, tunneling and honey@home components will redirect traffic from various distributed dark address spaces to the NoAH core. Inside NoAH core the traffic will first be processed by the low interaction honeypots and if it is considered "interesting" it will be forwarded to the appropriate high interaction honeypot for further processing. Therefore, only low interaction honeypots can communicate with the high interaction ones. As we mentioned in Section 3.3 we will use Argos containment environment in order to reduce the maintenance and damage costs when a honeypot gets attacked and thus in this way the availability of the NoAH core increases drastically. In addition all packets that arrive to the NoAH core are recorded to a pcap trace for offline processing.

Honeyd will be the primary low interaction honeypot inside the NoAH core. The scripts that honeyd uses to emulate services like Web, FTP or any other server can interact with the attackers up to a certain level and

past this level they are unable to send valid responses back to the attacker. Thus sophisticated attacks that need a series of steps before the actual exploit takes place can not be carried out to low interaction honeypots. A example of this kind of attacks is the well know exploit on Windows RPC service [19], which needs more that ten message exchanges before sending the actual exploit. Creating scripts that emulate such complex protocols is not applicative. Also considering the number of available Internet services along with the different versions, make the creation of a huge number of scripts for every service impractical to maintain, test and run. Therefore high interaction honeypots are needed so that the actual services can run and fully interact with the attackers.

Low interaction honeypots in the NoAH core will be used as lightweight responders. The responders will be able to identify the "uninteresting" traffic and will not forward it to the high interaction honeypots. Therefore the traffic resulting to the high interaction honeypots for processing will be reduced. The architecture is presented in Figure 3.7, where it can be observered that all traffic is first processed by the low interaction honeypots, then only the dialogs that can not be processed by the low interaction honeypots are handed off to the high interaction ones so to provide realistic responses back to the attacker. Attackers can not access high interaction honeypots directly and since Argos containment environment is used our high interaction honeypots can not be compromised and thus contaminate their private subnet.

3.4.1 "Lightweight Proxy" Honeypots

In every conversation with an attacker, the initial part of it is performed by a low-interaction honeypot. The low interaction honeypots inside NoAH core act as lightweight proxies for performance and traffic reduction reasons. The proxies are modified instances of honeyd which listen to specific ports based on a particular configuration. The operating systems and the services

running on these lightweight proxies are mapped to real services running to the high interaction honeypots. For example if a low interaction honeypot emulates a Windows XP Sp1 machine with an Apache server running, then the port 80 on the low interaction honeypot is mapped to a specific high interaction honeypot that runs Windows XP Sp1 with an Apache server running on it serving real pages.

These lightweight proxies in NoAH core try to reduce the traffic that arrives to the high interaction honeypots by dropping connections to ports that the corresponding services are not running on the high interaction honeypots. Also lightweight responders are able to detect TCP SYN scans by examining the first three packets of the conversation. If these packets do not comply with the standard TCP three-way handshake the connection is dropped. When the three-way handshake has been completed correctly with the attacker, the connection must be handed off to the appropriate high interaction honeypot. At this point the low interaction honeypot becomes a relay proxy and all data in application level are relayed to the high interaction honeypots and vice versa. This relaying of packets between the attacker and the high interaction honeypots continues until the termination of the communication.

In Figure 3.8 an example of a connection handoff is illustrated. Firstly, the attacker sends a TCP SYN packet to the low interaction honeypot with a certain destination port X, if port X is open a SYN/ACK is sent back to the attacker and the honeypot awaits to receive the next packet. If the next packet that arrives from the attacker is not an ACK then the connection is considered to be scan activity or malformed request and therefore dropped. If the third packet is received we have reached the “zero” point. At this “zero” point the low interaction honeypot connects with the high interaction honeypot running the requested service. After the connection has been established, the low interaction honeypots acts as proxy between the

attacker and the high interaction honeypot. We don't expect the attacker to notice the extra delay since the both low and high interaction honeypots will be located to the same local high speed network.

The previously described architecture, allows NoAH to accomplish a number of goals. To begin with only a a small number of high interaction honeypots needs to be maintaied since the major portion of the traffic will be filtered out by the low interaction honeypots. Moreover, by using honeyd we can map several "virtual" machines which run the same operating systems and similar services to a single high interaction honeypot. Finally, it is easy to add a new service to the whole system since we don't need to write any new complex scripts that emulate the specific service for the low interaction honeypots. It is only needed to install the specific service to the high interaction honeypot and set the appropriate mapping at the lightweight responders.

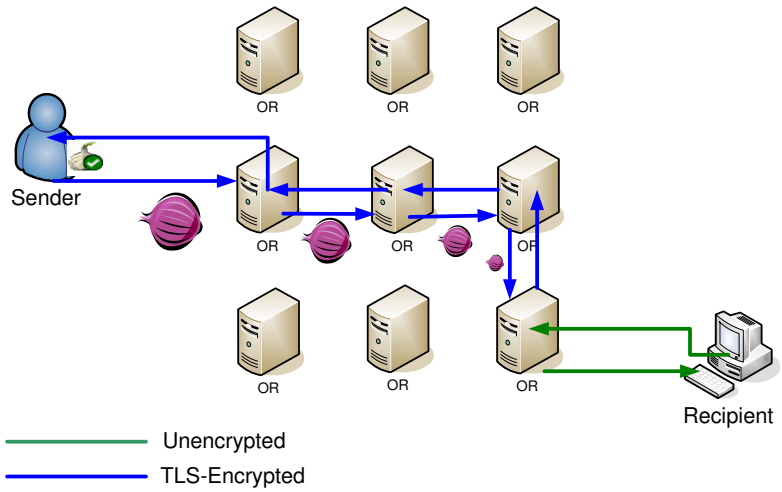

FIGURE 3.5: An overview of how Tor works. The client establishes a path of onion routers and sends "onions", messages encrypted with public keys of all path's routers. At each router the onion is piled off -decrypted by router's public key- and forwarded to the next router. The last router has fully decrypted content and communicates directly with recipient through a standard TCP/IP connection.

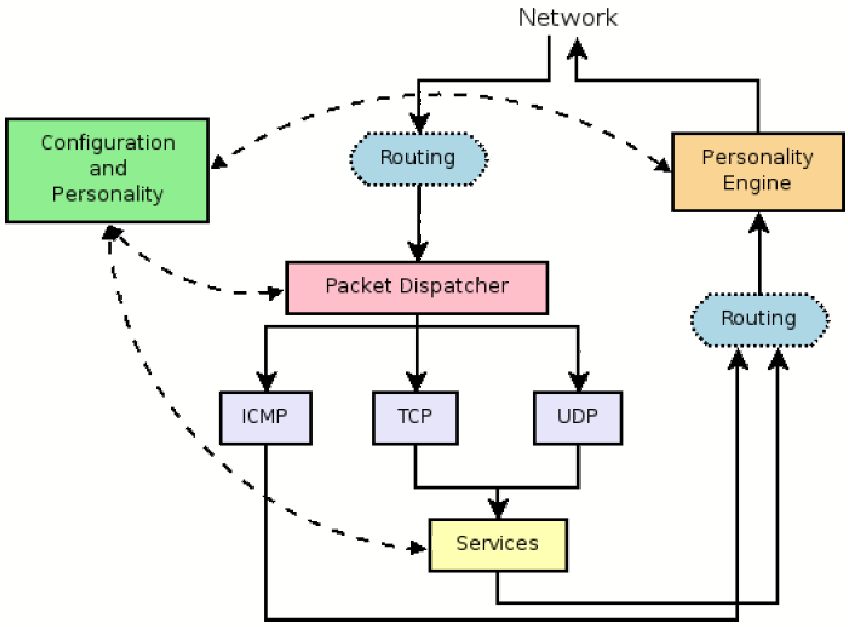

FIGURE 3.6: The diagram gives an overview of the Honeyd's architecture

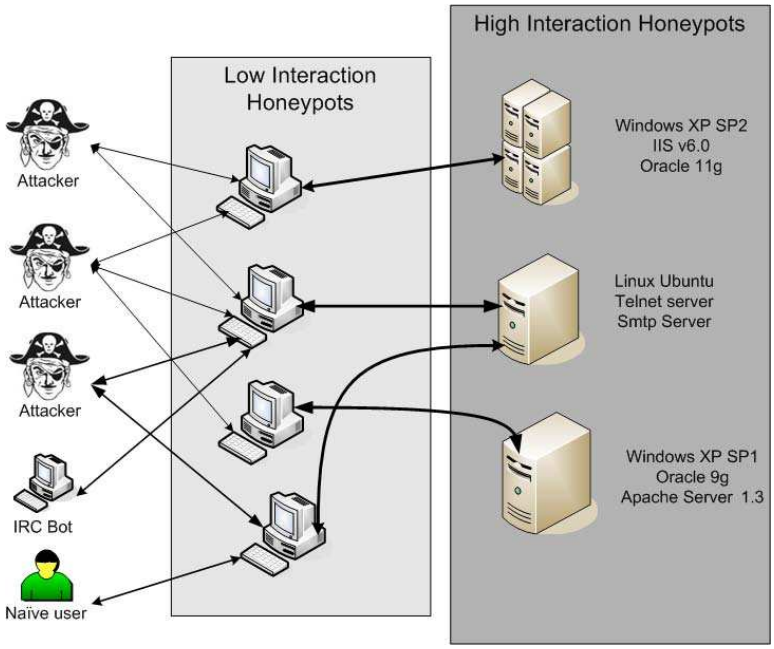

FIGURE 3.7: Cooperation between low and high interaction honeypots. Low-interaction honeypots are accessible by attackers, while high-interaction ones are placed in a private subnet. The traffic from attackers arrive to the low interaction honeypots which in turn uses the services on high interaction honeypots to provide real responses to the attackers.

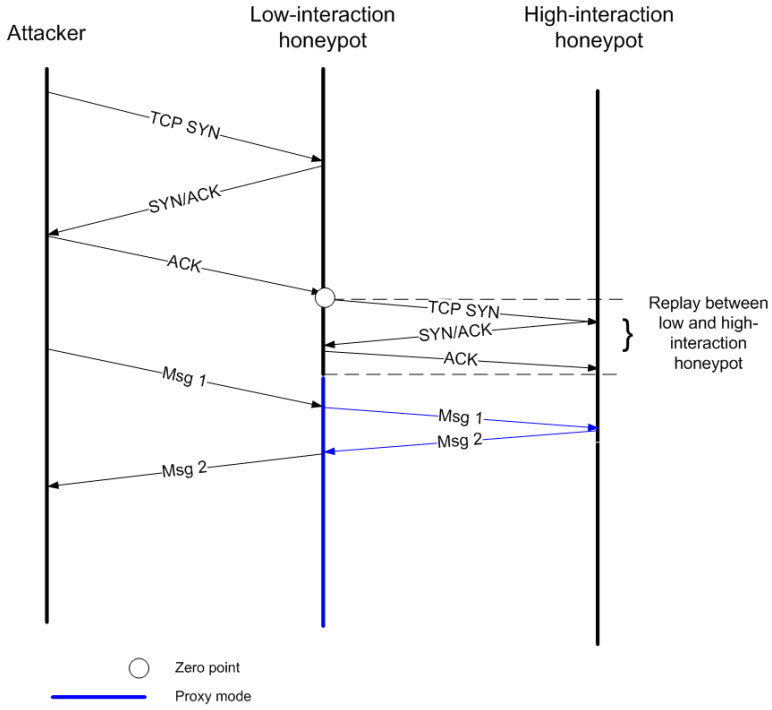

FIGURE 3.8: An example of a connection handoff. After a correct three-way handshake, low interaction honeypot becomes a proxy between the attacker and the high-interaction honeypot

4

Evaluation Methodology

In this chapter we will describe the methodology that will be used in order to evaluate each of the Noah components and the NoAH architecture as a whole. The evaluation will be based on a series of experiments for each of the components that is currently part of the NoAH infrastructure.

4.1 NoAH components

First we will outline the various NoAH components that are in our best interest. A brief overview of each component is presented here, for further information refer to Chapter 3 or to article [4]. There is a number of inter-operating components in the NoAH architecture. We are going to describe the methodology for evaluating each of these components and examine the following parts of the NoAH architecture: Windows Honey@Home, Linux

Honey@Home, SSL Server, Honeyd and Argos containment environment.

Honey@Home is a lightweight tool that monitors one or a limited number of unused IP addresses on a home user's network or on a small enterprise's network and interacts with the honeypots of NoAH core in order to respond to the received traffic. Currently, there are two versions of Honey@Home one for Linux and one for Windows Operating systems. Both versions share the same principles and work in a similar way but they must be evaluated in a way that the results are not affected by the different OS they operate on.

SSL Server is a tool used for collecting the data from every live Honey@Home sensor, decapsulating and injecting packets to the network where the various honeypots are located. As the name suggests the connection between every Honey@Home sensor and SSL Server is encrypted using Secure Socket Layer protocol. The data collected by SSL Server are then been processed by the low interaction honeypots of NoAH.

Honeyd is an open source computer program that allows a user to set up and run multiple virtual hosts on a computer network. These virtual hosts can be configured to mimic several different types of servers, allowing the user to simulate an infinite number of computer network configurations. The low interaction honeypots within NoAH are based on a modified version of Honeyd which is able to hand-off the connections we receive from the attackers to the high interaction honeypots. Moreover, Honeyd is configured to act as a lightweight responder in order to filter out port scans to our targeting infrastructure and connections to services that are not available on our high interaction honeypots.

In the context of NoAH architecture, Argos containment environment is used as the high interaction honeypot. Argos is a full and secure system emulator designed for use in honeypots. Using dynamic taint analysis, it tracks network data throughout execution and detects any attempts to

use it in an illegal way. By using Argos we secure our high interaction honeypots from getting compromised and thus if an attack against our honeypots succeeds it will not spread further.

Each and every component should be tested appropriately in order to identify its potential performance. Also, end to end tests should be performed in order to identify possible bottlenecks or other misbehaviors in the system. The methodology proposed should provide us a clear view of the behavior and performance of each component along with an overall view of the capabilities of our system. Based on the previous remarks the following sections will provide the main skeleton for the experiments that should be conducted for the better evaluation of the NoAH architecture.

4.2 Honey@Home and SSL Server Evaluation Methodology

The entry point to the NoAH architecture are Honey@Home tool and the funneling component. Any traffic going to certain ports or IPs on a host or network running Honey@Home is redirected to the NoAH core as we have previously described. Thus Honey@Home should be working correctly even in the presence of failures. Also, the performance of Honey@Home is an important factor affecting the performance of the whole architecture and should be examined thoroughly. Honey@Home can not be examined as a standalone tool since it redirects the traffic it receives to the SSL server through a SSH connection. So we need to test it as part of the NoAH infrastructure. Keeping the previous remarks in mind we are going to conduct a series of experiments.

In order to test the correctness of the system we will conduct a series of transactions using different protocols like HTTP, SSH or other well known protocols. Firstly we will replace Argos with an HTTP server and perform a series a request using wget tool [14]. The requests will be done to a

Honey@Home (both for Windows and Linux) client which in turn will direct the requests through the NoAH architecture to the HTTP server. We should validate that both the requests and responsed arrived correctly to the server and to the wget client respectively. Thus, what is in our best interest is the number of the correct requests completed. This can be done by comparing received by the wget client with the data that were hosted in the server side. Except from HTTP we should also try other protocols that are not plain text protocols like SSH. Therefore, we can setup a SSH server replacing Argos and using scp tool in order to transfer files over the network.

By using the same setup as before, we can measure the throughput of the part of the system that contains Honey@Home, SSL server and modified Honeyd. Since we will be able to measure the performance of Honeyd as a standalone tool, the latter measurement will give us a rough estimation of the performance of Honey@Home and SSL server as a whole. So the experiments that we will conduct are the following. Firstly we will replace Argos with an apache2 HTTP server and we will perform a number of transactions using wget tool through the Honey@Home-SSL server-modified Honeyd path. We are interested in the mean traffic rate that the system can sustain during these transactions. Secondly we will also use the nuttcp tool [21] in order to measure the throughput of the system. This will be done by replacing Argos with the nuttcp receiver and using the nuttcp transmitter to transmit data over the system. During the execution of the experiments we will also keep track of the CPU usage of Honey@Home and SSL server this will provide useful information for the further evaluation of the system.

Figure 4.1 presents the setup that will be used to perform experiments on the honey@home, SSL server and Honeyd part of the NoAH architecture. As it is shown the attacker has been replaced with a client and the high interaction honeypots with a real server. The requests from the client arrive

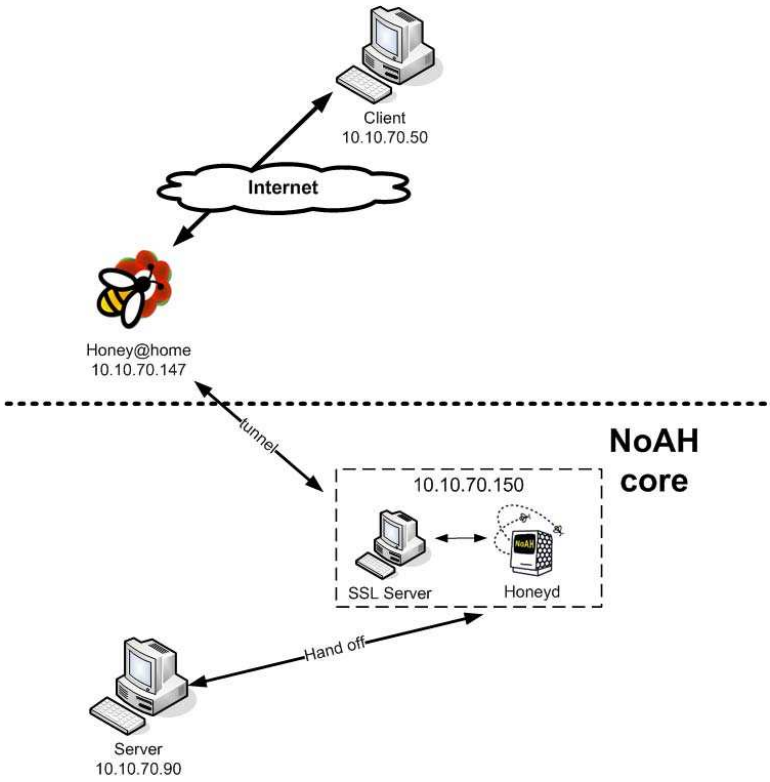

FIGURE 4.1: A test setup in which requests to the server are performed through the honey@home, SSL server and Honeyd. The server and the client can communicate using the same protocol.

to the IP address that honey@home is responsible to monitor and to a port that is open on the server. The request is then encapsulated and send over the SSL tunnel to the SSL Server there it gets decapsulated and sent to the low interaction honeypot. After the three-way handshake has been completed properly a connection is opened with the real server and from that point on all packets from the client are relayed to the real server and the responses from the server follow the reverse path back to the client.

Finally, in order to evaluate Honey@Home and SSL server in the presence of failures, we will establish a connection between the SSL server and Honey@Home. Then randomly we will either send a KILL signal to

Honey@Home or SSL server or drop the connection. This will help us understand how our system responds to failures and whether it will be able to restablish the connection once it is lost. Note that the experiments described previously must be conducted both for Windows Honey@Home and Linux Honey. This will enable us to do a quantitative comparison between the two versions of Honey@Home.

4.3 Modified Honeyd Evaluation Methodology

Honeyd is a well known tool used for representing single or a network of "virtual" hosts running a number of different services. Honeyd is used in the NoAH architecture as the low interaction honeypot. As we have mentioned earlier we have modified Honeyd (mhoneyd) by adding two new functionalities. Firstly, modified Honeyd is able to filter out SYN scan attacks. Thus if Honeyd receives a SYN, to a port that is configured to listen, it sends back a SYN/ACK and waits for the next packet if the next packet is not RST or FIN it hands off the connection to the HI honeypot. Therefore in addition to the scan filter, we have added a smart "hand off" mechanism is introduced. The "hand off" mechanism redirects connections that can no longer handle to the high interaction honeypots or any other server on the Internet.

Adding the previous mechanisms to Honeyd may impose some additional costs to the system. Thus we need to examine what is the maximum throughput that the modified Honeyd can achieve while it hands off numerous connections to a HI honeypot or to some real server. The first experiment that we will conduct will be for testing the mhoneyd as a standalone tool. So we will configure mhoneyd to redirect traffic, destinated to a certain port (e.g. 8080), to a web server. In this way all traffic from the client to the server and in the reverse path is redirected through mhoneyd. Since we expect mhoneyd to be the bottleneck in this experiment we need all three components to be connected to the same high speed network e.g.

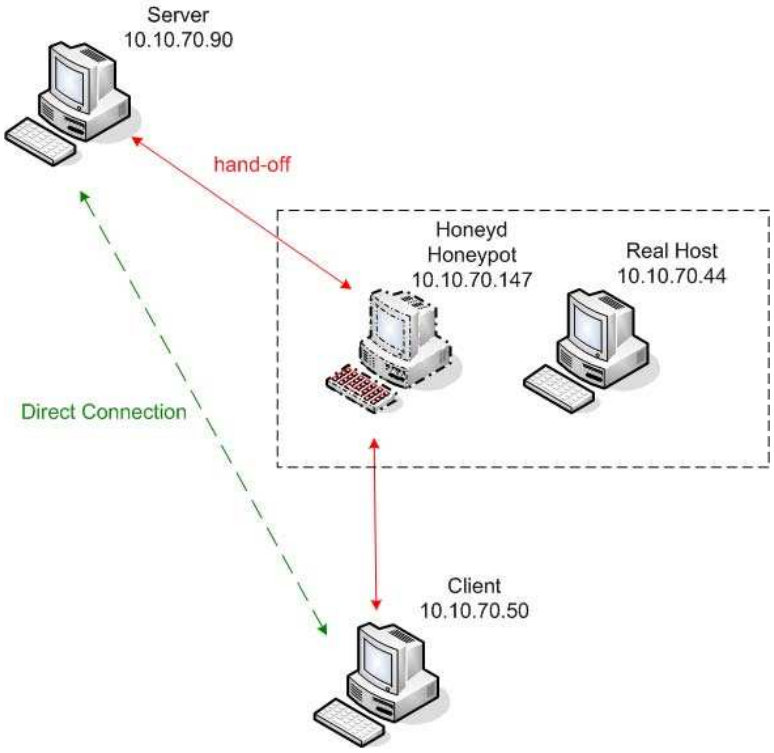

FIGURE 4.2: Setup for testing the overhead imposed by the handoff mechanism. A honeypot based on modified Honeyd is running on a real machine and the requests from the client are handed off to the real server.

100Mbps or 1Gbps. Then we will conduct a series of HTTP transactions using the wget tool. We shall keep track of the time needed for a transaction to be completed and we will be able to produce the mean traffic over all the transactions. The results will indicate what is the maximum throughput that mhoneyd can sustain. We can use different protocols in order to test mhoneyd but since the handoff is done in the socket layer we expect similar results as in the HTTP case.

The experimental setup to examine the hand off mechanism is illustrated in Figure 4.2. A client will produce a number of requests to the server both directly to the server IP address and indirectly through the low interaction

honeypot. The time needed to serve these requests will help us calculate the overhead imposed by the handoff mechanism to the system. The server and the client can communicate using any protocol since the hand off has been implemented to be protocol independent.

With the knowledge of what is the maximum throughput of mhoneyd, we can proceed to experiments adding more NoAH components to the set up, like Honey@Home and SSL Server. The methodology is described in the previous section where experiments for the path client-Honey@Home-SSL Server-mhoneyd are considered. The results from these experiments will help us pinpoint where the "bottleneck" in our system, if any, exists and what is the maximum traffic that the whole path can sustain.

4.4 High interaction honeypot Evaluation Methodology

As we have mentioned the high interaction honeypots that will be used in NoAH core will be running over Argos containment environment. Argos is based on Qemu virtual machine with some additional functionality that allows it to track the tainted data and disallows them from getting executed once the exploit has succeeded. Since Argos will be hosting the high interaction honeypots of NoAH architecture its performance is crucial and affects the whole NoAH architecture. Therefore, the effectiveness of Argos in intercepting attacks against our honeypots is critical because we don't expect our honeypots to be compromised and used for malicious purposes by the attackers or automated malware.

The performance of Argos will be evaluated through a series of experiments that will provide information about the slowdown of programs running over Argos and Qemu in comparison to the original hosts. Since we expect multiple operating systems and services to be running on our honeypots we should have multiple setups in order to test the stability and

coherence of the Argos' performance. The performance metrics that are in our best interest are the execution time for CPU intensive programs or requests per second and the throughput in Mbps for programs serving requests over the network like Web, FTP servers. A comparison between hosts and programs running over Argos, Qemu and on native system will reveal the overhead introduced by Argos.

Furthermore, an end-to-end experiment is needed to identify the maximum performance that NoAH architecture can sustain. The end to end experiment could be performed both with Argos and a native system as high interaction honeypot. This comparison will provide useful results in whether we can somehow "hide" the overhead of Argos inside the whole architecture. In other words it will reveal if Argos is the bottleneck in our system. This can be done by running an apache server on the high interaction honeypot, which in our case will be running over Argos or the native system and perform a series of requests for variant sized files; the same experiment can be repeated for different services like FTP, SSH, Netbios etc. The time per request and the throughput in Mbps will be our major metrics in this experiment.

Another aspect of Argos that needs to be examined is the effectiveness into averting cyberattacks. To determine the effectiveness of Argos we will use the metasploit framework [27] as the attacker. Metasploit framework allows us to test and exploit a remote machine with an already known exploit. We will conduct an experiment where we will use and send multiple exploits directly to the high interaction honeypot and we shall measure the number of the successful interruptions by Argos. We will perform the same experiment but instead we will use the NoAH infrastructure to redirect the attacks to Argos. In order to justify that Argos really throttled the attacks we will use the feedback produced by metasploit framework along with the logs that Argos produces.

5 Results

In this chapter we will present the experimental results for the various NoAH components along with some end-to-end measurements that provide us an overall view of the architecture capabilities. The experimental evaluation will be based on the remarks presented in section 4. The results suggest that by using the NoAH architecture, one is able to receive and detect the cyber attacks effectively. Moreover, the components has been designed to be as lightweight as possible in order to minimize the overall overhead and enhance the performance of the whole architecture.

5.1 Testbed

In this section we will describe certain systems which will be used for the various experimental set ups in the remainder of the chapter. Therefore,

for every experimental set up in the following sections we will refer here for the description of these hosts.

The machines that will be used are the following. A server that will be used to mainly host the high interaction honeypots and other various application servers running Debian Linux, 2.6.18 kernel version with an Intel Xeon with 3.0GHz CPU with 5GB of main memory we will refer to this machine as [Server42] . A machine running Debian Linux, 2.6.20 kernel version with an Intel Core 2, 2.40 GHz CPU and 2 GB of main memory this machine will mainly be used for running the low interaction honeypots and the SSL server, we will refer to this machine as [NoAH machine] . A machine that is used as client and as a honey@home sensor runs Debian Linux, 2.6.12 kernel version with an Intel Xeon, 2.80GHz CPU and 512MB of main memory we will refer to it as [H@H Linux client] . Moreover, two machines that were mainly used as clients, an Intel Xeon with 2.40 GHz and 512 MB of main memory and an Intel Pentium 4 with 2.80 GHz CPU and 1 GB of main memory. On both Linux clients Debian Linux with 2.6 kernel version was running, we will refer to one of machines as [Linux client] . Finally, we used a number of machine running Windows XP SP2 both for honey@home sensor but for clients two we will refer to on of them as [Windows client] .

The network topology will be presented in the following sections since it highly depends on the experimental set up and it is not fixed throughout the experiments.

5.2 High Interaction Honeypots Evaluation

Argos will be our main tool for the high interaction honeypots inside NoAH architecture. Argos is a containment environment based on the Qemu virtual machine. Services running on virtual machines have inferior performance compared to services running on a real machine. In addition to

that, we await that Argos tainting mechanism will add some additional performance costs to the running services. By definition, the operating system of the machine on which the virtual machine runs is called host OS, the operating system that runs inside the virtual machine is called guest OS.

For the computation of the extra overhead that Argos introduce to a system, we will conduct a series of requests using the apache benchmark [12] tool to an Apache 2.0 server running over a vanilla Linux , a Windows XP SP2 running on Qemu virtual machine and a Windows XP SP2 running over Argos containment environment. We downloaded a filed sized 10MB 1, 10, 100 and 1000 times with the use of apache benchmark tool. The results are illustrated on Figure 5.1, where the X axis presents the times that each file was downloaded and the Y axis presents the overall time in seconds for each set of downloads used for that set to complete. The machine hosting Qemu, Argos and Apache server was [Server42] , the machine hosting the apache benchmark client was [NoAH machine] . The machines were connected to the a 1Gbps network. As the results suggest, the apache server running on top of Qemu serves about 35 times slower compared to the apache running on vanilla Linux and 55 times slower while running on top of Argos. Although results seem discouraging, considering that the the services running on a high interaction honeypots are not bandwidth intensive, we could sustain a large number of clients with the 15Mbps traffic rate that an Apache server running on Argos can provide.

Towards to further understanding Argos' behavior when different operating systems is hosted, we installed three different operating systems, Debian Linux, Windows XP SP2 and Windows XP SP2 in safe mode with only command prompt active. We ran Argos over [Server42] machine with each of the operating systems and recorded the CPU consumption over time at host level when each of the operating systems were idle. Figure 5.2 presents the results for a period of 2000 seconds where we can observe that when

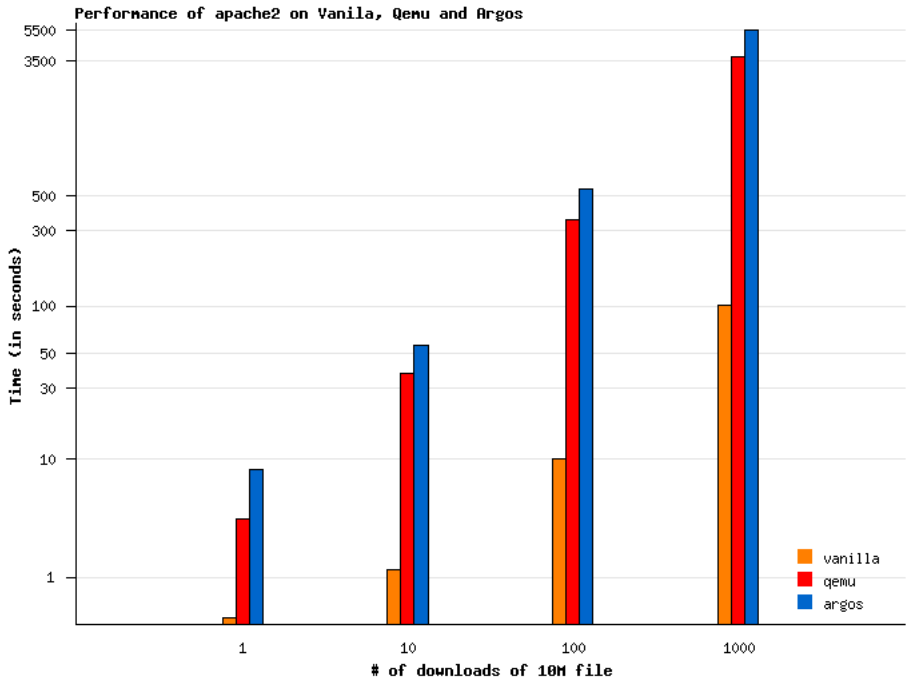

FIGURE 5.1: Download time in seconds for a file sized 10M and for various number of sequential downloads

Windows XP SP2 runs we have almost constant 100% CPU usage whereas on the Linux and Windows XP SP2 in safe mode the CPU usage drops at some point. We suspect that this point is when the booting of the guest OS has completed. We suspect that due to the real time requirements of an operating system like Windows XP SP2, constant requests are preformed by the guest OS, for example to retrieve information about the real time clock or to access the driver of the graphic card etc.

Results suggest that Argos can be used for building high interaction honeypots even though it imposes significant costs to application and services that run on top of it. Moreover, since we expect to have a number of high interaction honeypots in NoAH core, we can use load balancing

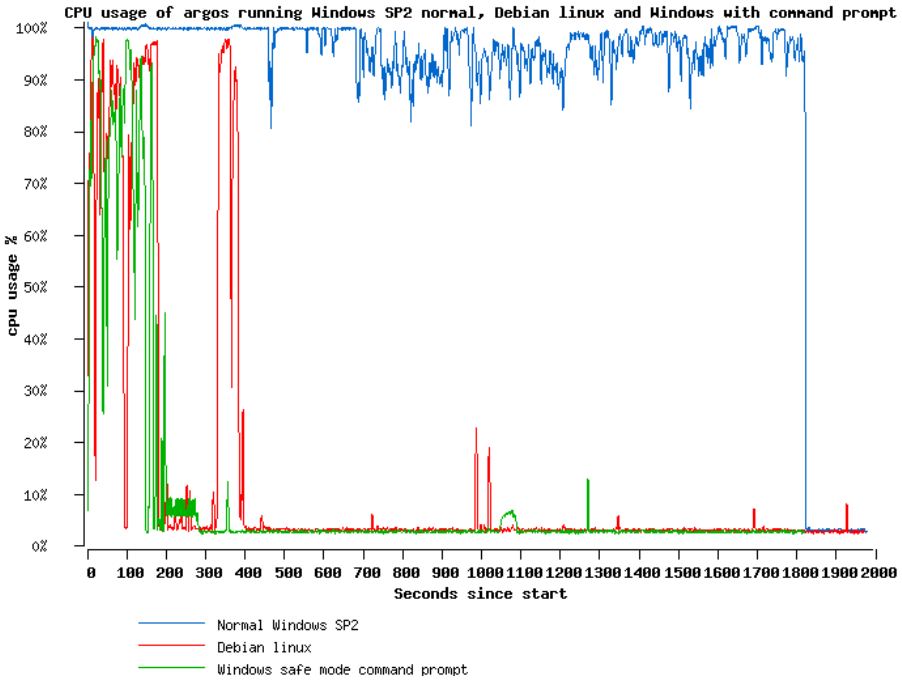

FIGURE 5.2: CPU usage of Argos while running for three different guest Operating Systems

techniques to be able to cope with the increasing number of honey@home sensors and funneling components. Finally, there are some issues that need further investigation like 100% usage of Argos when the guest OS is Windows XP SP2.

5.3 Hand Off Mechanism Evaluation

As we described previously, the low interaction honeypots inside NoAH core will be working as lightweight responders which redirect only the necessary traffic to the high interaction honeypots for further processing and drop packets that belong to scan or other benign activity. As we have stated earlier after the initial TCP hand shake between the attacker and the low interaction honeypot has been completed, the low interaction honeypots

becomes a proxy that redirects all the traffic from the attacker to the appropriate high interaction honeypot and vice versa. This “hand off” mechanism imposes additional cost to the the whole path between the high interaction honeypots and the attacker. In this section we will calculate the overhead introduced by the “hand off” mechanism and the overhead when both “hand off” and funneling are active.

In order to measure the costs that “hand off” introduces to the system we performed a series of experiments based on two different setups. The experiments took place in two subnets which were connected through a 100Mbps link. Also, their internal capacity of the subnets was 100Mbps. Figure 4.1 presents the setup for the hand off experiment. The client was running on [Linux client] machine, the server was running on [Linux client] machine and the low interaction honeypot was running on [NoAH machine] machine. We used the client to perform a series of "HTTP GET" requests and (Secure CoPy) "scp" requests to the honeypot. Figure 5.3 presents the testbed used for calculating the overhead in the case where both handoff and funneling are activated. The client, the server and the handoff machine were set up as described previously whereas the funneling host was running on [H@H Linux client] and located at a different subnet. The honeypot was located at the 10.10.70 subnet and was receiving traffic from the 10.10.71 subnet using the funneling mechanism. In both cases we conducted the exact same set of experiments. The results are presented in the remainder of the section.

We used two different sets of experiments in order to determine the total overhead introduced to response time of the server. The first set consisted of a series of "HTTP GET" requests for three files of different sizes 10K, 100K and 1000K. We requested each one of these files 500, 1000, 2000 and 5000 times and recorded the total time for serving these requests. The second experiment used the same setup but we used (Secure CoPy) “scp”

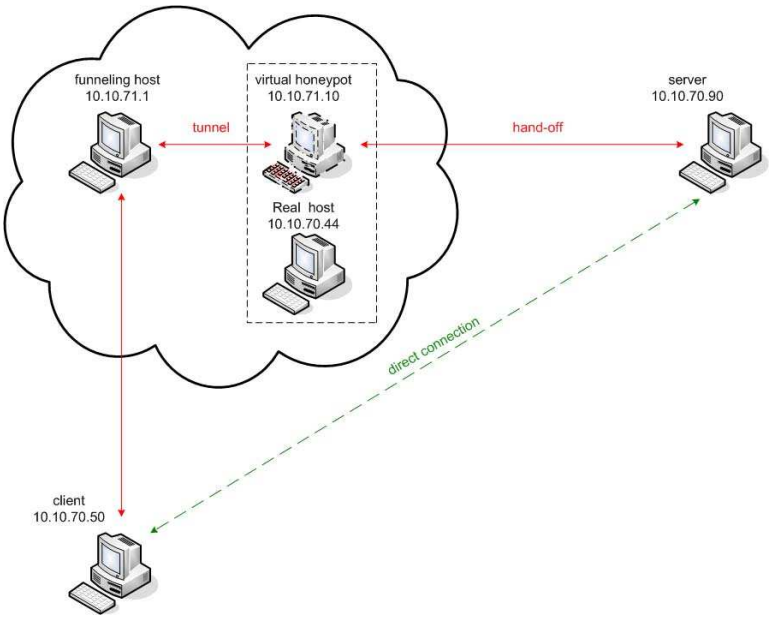

FIGURE 5.3: The experimental testbed for measuring the overhead of both funneling and tunneling mechanisms. The traffic received at the funneling host is redirected through a tunnel to the low interaction honeypot which hands it off to the real server.

instead of HTTP requests. These requests were served from the server in three different manners: a) Directly from the server as presented in Fig. 4.1 and Fig. 5.3 denoted with the discontinuous line b) Through the handoff mechanism which is presented in Fig. 4.1 with a red line and c) Thorough both funneling and handoff mechanisms as presented in Fig. 5.3 with the continuous red line. The results of these experiments are presented in the following tables.

Table 5.1 illustrates the server response time for a number of HTTP requests along with the overhead introduced by handoff and the hand off and funneling mechanisms. We discover that the overhead introduced by the handoff mechanism remains is same regardless the number of the request performed. Furthermore, an additional overhead of about 20% is intro-

Number of HTTP Requests	Real Server (sec)	Hand Off (sec)	Hand Off and Funneling(sec)	Overhead of Hand Off(%)	Overhead of Hand Off and Funneling (%)
500	11,17	17,95	20,37	60,70	82,36
1000	22,35	35,93	40,73	60.76	82,23
2000	44,69	71,86	81,39	60,79	82,12
5000	111,75	179,84	203,67	60,93	82,25

TABLE 5.1: Response time for numerous HTTP Get requests for a file of 100KB size. The table presents the response time when retrieving the file from the real server and from the real server through the "hand off" and Funneling mechanisms

duced by the funneling mechanism. These findings are also confirmed in Table 5.2 where we can see the introduced overhead for performing different numbers of "scp" transfers when the handoff mechanism is activated. But, in this case the additional overhead introduced by the funneling increases from 20% to 29%. This probably happens due to the increased complexity of the Secure CoPy protocol in contrast to the plain text HTTP protocol.

Results suggest an overhead of about 60% in the case of hand off and for plaintext protocols and about 10-13% in the case of binary protocols like ssh. Furthermore, the funneling mechanism adds about 20-29% overhead to the whole system when it is activated. Since we will use Argos as our main high interaction honeypot, we expect Argos to be the "bottleneck" of our system and thus the performance of the hand off is sufficient for our purposes.

5.4 End-to-End Evaluation

In this section we will provide experiments that were conducted while using the whole NoAH architecture. These experiments will either use the NoAH architecture "as is" or will alter a small fraction of its components e.g.

Number of SCP transactions	Real Server (sec)	Hand Off (sec)	Hand Off and Funneling(sec)	Overhead of Hand Off(%)	Overhead of Hand Off and Funneling (%)
500	73,36	82,91	101,93	13,01	38,95
1000	146,69	162,32	203,95	10,65	39,03
2000	293,41	323,96	407,90	10,41	39,02
5000	732,82	811,63	1019,96	10,75	39,18

TABLE 5.2: Response time for numerous SCP transaction for a file of 100KB size. The table presents the response time when retrieving the file from the real server and from the real server through the "hand off" and Funneling mechanisms

replace the high interaction honeypot with a real server, or change the attacker with an application client. The results presented here report that it is feasible to use NoAH architecture to detect attacks effectively.

5.4.1 Honey@Home Evaluation

Honey@home tool is a lightweight forwarder, which can be installed by any user who has a spare IP or port range, and can claim a dark IP address and forward all the traffic it receives to NoAH core. The traffic received is encapsulated and sent over a SSL tunnel. When it reaches the NoAH core it gets decapsulated by the SSL server component and injected to the network for further processing by the low and high interaction honeypots. Towards understanding the performance of honey@home we conducted an experiment based as described in Chapter 4.

Figure 3.4 illustrates the setup we used. We used the NoAH architecture where the attacker and the HI honeypot had been replaced by the nuttcp [21] transmitter and receiver respectively. The modified Honeyd was also configured to redirect the data to the correct port at nuttcp receiver. The [H@H Linux client] machine was used as the nuttcp client and [Linux

Buffer size (MB)	Windows Honey@Home Throughput(Mbps)	Linux Honey@Home Throughput(Mbps)
1	13,68	36,78
2	15,36	38,28
10	14,53	27,08
20	13.32	23,66
40	14,25	23,52

TABLE 5.3: The mean transfer rate as produce by nuttcp tool for numerous buffer sizes. The traffic was redirected through Honey@Home, SSL server and modified Honeyd to the nuttcp receiver.

client] machines was used as server, the [NoAH machine] hosted both the modified Honeyd and the SSL server, [Windows client] was used as the Windows honey@home sensor and [Linux client] machine was used as the Linux honey@home sensor. All machines were connected to a 100Mbps network. The nuttcp transmitter was sending buffers of different sizes using TCP. We performed the same experiments both for Linux and Windows Honey@Home. The mean transfer rates reported by the nuttcp receiver for Windows H@H and different send Buffer sizes over 10 runs are presented at Table 5.3

We tried to pinpoint the bottleneck in each experimental setup. Thus, we were keeping track of the CPU usage while the transfer was taking place. In the case of Linux Honey@Home, we identified that the machine running mHoneyd and SSL server was the bottleneck of the system, since these two processes were consuming about the 70% of CPU and the rest 30% was system time so the machine was at 0% idle during the transfer. In the case of the Windows Honey@home the bottleneck of the whole path was honey@home it self as it consumed 100% CPU usage when the tranfer was active.

The results suggest that the honey@home - SSL server - Mhoneyd path can sustain traffic up to 38 Mbps when the Linux honey@home is used and up to 15 Mbps when the Windows honey@home sensor is used. The performance of the Windows Honey@Home needs to be examined further in order to determine its extensive CPU usage. Overall, we can safely use the whole path to redirect traffic from the various honey@home sensors to the NoAH high interaction since we expect the traffic from each sensor not to exceed tens of Kbps or less .

5.4.2 End-to-End measurements

In this section we will describe a series of experiments that used the NoAH architecture as is without altering any of its components. We only used one Windows and one Linux Honey@Home sensor in our experiments. The set up was as follows. One Windows XP SP2 and one Windows XP SP1 operating systems were running over two Argos processes which were located inside [Server42] machine. The modified Honeyd along with the SSL server were located in [NoAH machine] . The Honey@home sensors was either run on top of [Linux client] machines or [Windows client] machine. Finally, the attacker or the application client, depending on the experiment, was running on top of [Linux client] machine.

Firstly we performed a number of attacks against the rpc service of the Windows XP SP1 machine through the NoAH infrastructure, using the metasploit [27] framework. The attacks was launched against honey@home which redirected the traffic to SSL server, which in turn forwarded the traffic to low interaction honeypots that finally reached the Argos machine and provided the appropriate responses back to the attacker. Figure. 5.4 presents the output of each of the NoAH components along with the attacker console. As it is shown, a connection was first established between the honey@home sensor and the SSL Server (in this case is is called “Honey@Home Server”). Then an attack against the rpc protocol was

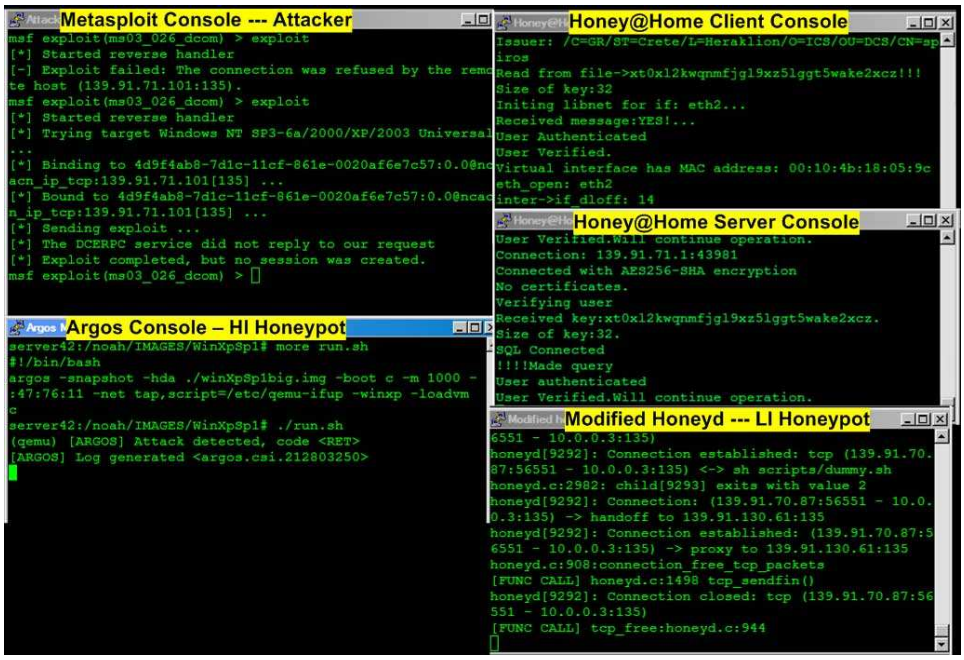

FIGURE 5.4: An attack against NoAH infrastructure is launched by the metasploit framework. NoAH's high interaction honeypots receive the attack and are able to deflect it effectively.

launched by the metasploit framework as it is shown in the "Metasploit Console". The traffic had reached the modified Honeyd machine which in turn redirected the attack traffic to Argos that managed to detect the attack and log it, as it is presented in "Argos Console". The metasploit framework reported that "Exploit completed, but no session was created" which is exactly what we expected since Argos is able to detect the attack at the time the attack data are about to be executed and therefore the exploit has succeeded but it can not be executed any further and return a command shell to the attacker. We launched tens of attacks against both our Windows and Linux honey@home sensors and Argos was able to detect and deflect each of the attacks, producing the appropriate attack logs.

We have concluded that NoAH architecture is up and running correctly

thus we can proceed to an end to end estimation of the traffic rate that the whole system can sustain. We installed Apache2 server to the Windows XP SP2 guest OS and performed ten sequential downloads, using the wget tool, of a text file of 10MB of size both from the Windows and Linux honey@home clients. The traffic rate that we were able to achieve through both versions of honey@Home was *15Mbps*. As we have previously shown at Section 5.2, 15Mbps is the maximum traffic rate that we can get while downloading from a server running inside a Windows guest OS on top of Argos.

The experimental results suggest that we are able to deploy NoAH architecture and effectively detect previously known attacks or previously unknown attacks that can be detected by Argos. We should further deploy NoAH by adding new OS's running on top of Argos. Moreover, we need to test NoAH under large scale condition where thousands or tens of the thousands of honey@home clients are concurrently connected to NoAH infrastructure sending Mbps of traffic to the NoAH core.

6

Conclusions and Future Work

This thesis provides a detailed overview of the *Network of Affined Honeypots* architecture. Based on standard network technologies, NoAH provides a complete framework for the construction of honeypot networks. Moreover, the building blocks of NoAH are designed to be flexible and easily expandable. The network of honeypots that NoAH aims to deploy will provide detection of known attacks and act as an early warning system for previously unknown attacks.

NoAH is comprised of a NoAH core and multiple distributed sensors that redirect unsolicited traffic to the NoAH core. The core of NoAH consists of both low and high interaction honeypots. The low interaction honeypots act as lightweight proxies that aim to reduce the traffic that arrives to high

interaction honeypots by filtering out "uninteresting" traffic, like portscan activity making feasible to monitor large IP address space. There are two types of monitoring technologies used in NoAH, honey@home tool and various funneling and tunneling mechanisms. Organizations can participate to NoAH, without the maintainance cost of honeypots, by installing funnels that gather traffic from a large portion of black space and tunnel all traffic destinated to that black space, to NoAH core. The honeypots located in NoAH core will reply to that traffic providing the illusion to the attacker that a real service is contacted. Honey@Home is a non-pervasive tool that can run on a home PC and claim traffic that is destinated to an unused IP address, like an extra address obtained through DHCP. This traffic is directed to NoAH core which sends back responses to the user's network and from there to the attacker.

As the results suggest, NoAH can be easily deployed and used effectively for the detection of attacks. NoAH was able to detect and intercept attacks launched against the honey@home sensor aiming services that were running to the high interaction honeypots. We conducted a series of experiments on services running on top of high interaction honeypots and concluded that although the latency when performing a transaction through NoAH infastructure is increased, but it is not prohibitive for largely deploying and expanding NoAH.

NoAH is largely under deployment and many features are still under implemention. Therefore further experiments must be performed. One such feature is the use of tor, which will be used connecting honey@home sensor and low interaction honeypots anonymously. We expect that there will be additional overhead to the whole NoAH infrastructure which needs to be counted and understand in depth. Moreover, NoAH needs to be tested with a large number of unused IP addresses are monitored and redirected to the NoAH core. This real time stress testing will provide useful information

about the fault tolerance and robustness of NoAH.

Overall, the architecture of NoAH combines various network components and technologies to allow the construction of a robust honeypot network. It is mainly based on a core of honeypots but that can be extended to any organization and home uses. The extensibility of NoAH allows us the creation of large-scale honeypot farm, able to monitor distributed address space in a controlled environment. Futher work is still needed for the better understanding and evaluation of the new features that are mounted to NoAH architecture

Bibliography

[1] Bochs IA-32 Emulator Project. http://bochs.sourceforge.net/.

[2] F-Secure . http://www.f-secure.com/.

[3] Honeynet Project . http://www.honeynet.org/.

[4] Honeypot Node Architecture . http://www.fp6-noah.org/publications/deliverables/D1.1.pdf/.

[5] Honeytrap . http://honeytrap.mwcollect.org.

[6] LaBrea: "Sticky" Honeypot and IDS . http://labrea.sourceforge.net/.

[7] QEMU homepage. http://fabrice.bellard.free.fr/qemu/about.html/.

[8] Sebek homepage . http://www.honeynet.org/tools/sebek/.

[9] Symantec . http://www.symantec.com/.

[10] The Xen virtual machine monitor . http://www.cl.cam.ac.uk/Research/SRG/netos/xen/performance.html.

[11] VMware homepage . http://www.vmware.com/.

[12] ab - Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/2.0/programs/ab.html.

[13] F-Secure Virus Descriptions : Mydoom. http://www.f-secure.com/v-descs/novarg.shtml.

[14] GNU Wget. http://www.gnu.org/software/wget/.

[15] Google Hack Honeypot. http://ghh.sourceforge.net/.

[16] Honeywall CDROM. http://www.honeynet.org/papers/cdrom/index.html.

[17] Jackpot Mailswerver. http://jackpot.uk.net/mailswerver/.

[18] LEURRECOM.org Honeypot Project. http://www.leurrecom.org/index.php.

[19] Microsoft Security Bulletin MS03-026. http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx.

[20] Mwcollect alliance. *http://www.mwcollect.org* .

[21] nuttcp - network performance measurement tool. http://linux.die.net/man/8/nuttcp.

[22] Open proxy. From Wikipedia, the free encyclopedia . http://en.wikipedia.org/wiki/Open_proxy.

[23] Open rail relay. From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Open_mail_relay.

[24] Project Honey Pot. http://www.projecthoneypot.org/index.php.

[25] spamhole - The Fake Open SMTP Relay. https://sourceforge.net/projects/spamhole/.

[26] Symantec report on W32.Sobig.F@mm. http://www.symantec.com/security_response/writeup.jsp?docid=2003-081909%-2118-99.

[27] The Metasploit Project. http://www.metasploit.com/.

[28] The nmap network scanner. http://www.nmap.org/.

[29] The WASC Distributed Open Proxy Honeypot. http://www.webappsec.org/projects/honeypots/.

[30] Wireshark: The world's foremost network protocol analyzer. http://www.wireshark.org/.

[31] MAAWG Issues First Global Email Spam Report. `http://www.maawg.org/news/maawg060308`, 2006.

[32] P. Baecher, T. Holz, M. Kotter, and G. Wicherski. The Malware Collection Tool (mwcollect).

[33] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *9th International Symposium On Recent Advances In Intrusion Detection, RAID06, Hamburg, Germany, September 20-22, 2006, Proceedings*, Lecture Notes in Computer Science 4219. Springer, 2006.

[34] Bailey, M. and Cooke, E. and Watson, D. and Jahanian, F. and Provos, N. A hybrid honeypot architecture for scalable network monitoring. Technical report, CSE-TR-499.

[35] C. Cowan, C. Pu, D. Maier, H. Hinton, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. *7th USENIX Security Conference*, pages 63–77, 1998.

[36] David Dagon and Xinzhou Qin and Guofei Gu and Wenke Lee and Julian Grizzard and John Levin and Henry Owen . HoneyStat: Local Worm Detection Using Honeypots . In *Proceedings of the Recent Advance in Intrusion Detection (RAID) Conference 2004*, September 2004.

[37] J. Dike. A user-mode port of the Linux kernel. *Proceedings of the 2000 Linux Showcase and Conference*, 2(4), 2000.

[38] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. *Proceedings of the 13th USENIX Security Symposium*, 2, 2004.

[39] D. Goldschlag, M. Reed, and P. Syverson. Onion Routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):39–41, 1999.

[40] K. Hickman. The SSL protocol. *Netscape Communications Corp., Feb*, 9, 1995.

[41] C. Leita, K. Mermoud, and M. Dacier. ScriptGen: an Automated Script Generation Tool for Honeyd. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)*, December 2005.

[42] Michael Vrable and Justin Ma and Jay Chen and David Moore and Erik Vandekieft and Alex Snoeren and Geoff Voelker and Stefan Savage. Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP)*, October 2005.

[43] J. R. of New Zealand Honeynet Project, R. M. of Chicago Honeynet Project, B. E. of Chicago Honeynet Project, and M. M. of German Honeynet Project. Know your Enemy:Web Application Threats, "Using Honeypots to learn about HTTP-based attacks". `http://honeynet.org/papers/webapp/`.

[44] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. *Proceedings of the 2006 EuroSys conference*, pages 15–27, 2006.

[45] N. Provos. Systrace-Interactive Policy Generation for System Calls. *Na internet `http://www.citi.umich.edu/u/provos/systrace/`, July*, 2002.

[46] N. Provos. A Virtual Honeypot Framework. In *Proceedings of the 12th USENIX Security Symposium*, pages 1–14, August 2004.

[47] K. G. A. Spiros Antonatos and E. P. Markatos. Honey@home: A new approach to Large-scale Threat Monitoring. In *5th ACM Workshop on Recurring Malcode (WORM 2007)*, November 2007.

[48] D. X. Xuxian Jiang. Collapsar: A VM-Based Architecture for Network Attack Detention Center. In *Proceedings of the 13th USENIX Security Symposium*, pages 15–28, August 2004.