Computer Science Department
University of Crete

Exploring honeypot technologies for malware collection and cyber-attack information acquisition

Master's Thesis

Iasonas Polakis

October 2009
Heraklion, Greece

University of Crete
Computer Science Department

Exploring honeypot technologies for malware collection and cyber-attack information acquisition

Thesis submitted by
Iasonas Polakis
in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Iasonas Polakis

Committee approvals: _____
Evangelos P. Markatos
Professor, Thesis Supervisor

_____
Maria Papadopouli
Assistant Professor

_____
Sotiris Ioannidis
Associate Researcher

Departmental approval: _____
Panos Trahanias
Professor, Chairman of Graduate Studies

Heraklion, October 2009

Abstract

The Internet today is plagued by numerous types of malicious software (malware). Most of them proliferate by exploiting vulnerabilities in the operating system or services that run on hosts connected to the Internet. As computer networks expand, and their functionality is applied in many fields, more and more people are starting to use them. The increase of vulnerable computers connecting to these networks has led to a major increase in the number of malware created by malicious users in an attempt to infect as many hosts as possible. As a result, one of the major goals of the Security community, is creating infrastructures that can gather new samples of malware as soon as they are released in the wild, so as to deploy appropriate defense mechanisms.

The goal of this thesis is to explore honeypot technologies as a means of extracting information from malicious network traffic and gathering malware samples. First we examine the benefits of enhancing and extending the architecture and backend of an existing malware collection infrastructure, namely the NoAH project. Second, we present HoneyBuddy, a novel infrastructure that gathers information regarding the increasing number of attacks in instant messaging (IM) traffic. We present a prototype implementation as well as the results from our deployment.

Supervisor: Evangelos P. Markatos

Εξερεύνηση της χρήσης των honeypot τεχνολογιών για την συλλογή κακόβουλου λογισμικού και την απόκτηση πληροφορίας σχετικά με διαδικτυακές επιθέσεις

Πολάκης Ιάσονας

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Περίληψη

Το Διαδίκτυο σήμερα μαστίζεται απο μια πληθώρα διαφορετικών ειδών κακόβουλου λογισμικού. Τα περισσότερα απο αυτά εξαπλώνονται εκμεταλλευόμενα ευάλωτα σημεία στο λειτουργικό σύστημα ή στις διάφορες υπηρεσίες που εκτελούνται στους κόμβους που είναι συνδεδεμένοι στο Διαδίκτυο. Καθώς τα δίκτυα υπολογιστών επεκτείνονται, και η λειτουργικότητα τους εφαρμόζεται σε πολλά πεδία, αυξάνεται και ο αριθμός των ατόμων που τα χρησιμοποιούν. Η αύξηση των ευάλωτων υπολογιστών που συνδέονται σε αυτά τα δίκτυα έχει οδηγήσει στην ραγδαία αύξηση του αριθμού και των ειδών κακόβουλου λογισμικού που δημιουργούνται απο κακόβουλους χρήστες σε μια προσπάθεια να μολύνουν όσους περισσότερους κόμβους γίνεται. Ως αποτέλεσμα, μια απο τις κυριότερες ανησυχίες της κοινότητας που ασχολείται με την ασφάλεια υπολογιστών ειναι να δημιουργεί νέες υποδομές που μπορούν να συλλέξουν νέα δείγματα κακόβουλου λογισμικού μόλις αυτά εξαπολυθούν, για να αναπτύσσει τους κατάλληλους μηχανισμούς άμυνας.

Ο στόχος αυτής της μεταπτυχιακής εργασίας είναι να εξερευνήσει την χρήση των τεχνολογιών honeypot ως μέθοδο εξαγωγής χρήσιμων πληροφοριών απο δικτυακή κίνηση που περιλαμβάνει επιθέσεις καθώς και συλλογή δειγμάτων κακόβουλου λογισμικού. Πρώτα εξετάζουμε τα πλεονεκτήματα που προκύπτουν απο την τροποποίηση της αρχιτεκτονικής μιας υπάρχουσας υποδομής για την συλλογή κακόβουλου λογισμικού, και συγκεκριμένα του NoAH project. Δεύτερον, παρουσιάζουμε το HoneyBuddy μια πρωτότυπη υποδομή που συλλέγει πληροφορίες σχετικά με επιθέσεις που γίνονται σε δίκτυα προσωπικών μηνυμάτων (instant messaging). Παρουσιάζουμε την υλοποίηση καθώς και τα πει-

ραματικά αποτελέσματα.

Επόπτης Μεταπτυχιακής Εργασίας: Ευάγγελος Π. Μαρκάτος

Acknowledgments

I would like to thank my Supervisor, Professor Evangelos P. Markatos, for his continuous support in my work and studies. I want to express my deepest gratitude to Spiros Antonatos (a-i-a) for his invaluable help and guidance that led to the realisation of this thesis.

I would like to thank Sotiris Ioannidis for his helpful insight and all the past and present members of the Distributed Computing Systems Lab, Demetres Antoniades, Manos Athanatos, Michalis Polychronakis, Elias Athanasopoulos, Alexandros Kapravelos, Vassilis Pappas, Nikos Nikiforakis, Thanasis Petsas, Aris Tzermias, Antonis Papadogiannakis, Christos Papachristos, Manolis Stamatogiannakis, John Velegrakis, Andreas Makridakis, Michalis Foukarakis, Giorgos Vasiliadis, Vassilis Lekakis, Giorgos Kontaxis, Eleni Gessiou, Spiros Ligouras for their support and friendship.

I would also like to thank all my friends for their support, love and tolerance. You know who you are. K,A,T,S,I,P,O,F,I!!!!!

Finally, I would like dedicate this work to my parents, George and Karen, and my sister Maria for their endless love and support throughout the years.

Contents

List of Figures

List of Tables

1 Introduction

Many different types of malicious software (malware) plague the Internet today. From worms to viruses and trojans, attackers release numerous attack vectors. Whatever the type, the majority propagates by exploiting vulnerabilities in the operating systems and services running on networked computers. The multitude of vulnerable hosts are susceptible to every new sample released by attackers, even in the cases where users and administrators try to protect their assets by installing intrusion detection systems.

As computer networks become bigger and their use more widespread, the number of computers that get infected increases exponentially. Thousands of new malware instances are discovered on a daily basis [1, 2] by security labs and researchers worldwide, indicating the major increase in network security threats. The speed, by which these threats proliferate, renders the deployment of malware gathering infrastructures a necessity. For end host defense mechanisms such as anti-virus software to be effective against malware, it is vital that anti-virus vendors receive and analyze new threats immediately upon their release. Therefore, security researchers design and deploy systems that automatically gather and analyze malware instances.

The distributed deployment of such systems is vital, and has several advantages. First and foremost, the effectiveness of these systems is improved

as they obtain a "global" and more representative view of the network by monitoring several distributed subnets. Second, a larger number of deployed sensors increases the amount of received traffic and the possibility of encountering new malware variations and types. The ultimate goal of such systems is to minimize the time window between the release of a new malware instance and the moment it is captured. Once it is captured it is analyzed by researchers that create signatures that will update end host defense mechanisms.

A common type of such infrastructures deployed by researchers are honeypot systems. Practically, a honeypot is a computer system set up with the sole purpose of luring attackers. Honeypots do not initiate network connections and, thus, all incoming network traffic is either malicious or the result of network misconfigurations. There are different types of honeypots, based on the level of interactivity they provide to attackers and whether they monitor server or client side attacks. Based on these characteristics we identify four categories of honeypots: low-, medium-, high-interaction and client-side honeypots. Each type of honeypot presents several good characteristics but also a number of drawbacks. Different types of honeypots are suitable for different problems. Therefore, it is vital to select the appropriate honeypot for the extraction of information depending on the specific situation.

An approach that aims to combine the benefits of several types of honeypots is proposed by the Network of Affined Honeypots (NoAH) project. The NoAH architecture proposed the combination of both low- and high- interaction honeypots. The main idea is that depending on the type of incoming attack traffic, it is handled by the most appropriate honeypot. The goal of the NoAH project is to use honeypots to monitor unused portions of the IP address space and interact with malicious traffic which is subsequently analyzed. The architecture of NoAH presents a flexible design for deployment and collaboration of honeypots. Honeypots in NoAH are deployed inside the "NoAH core" the center of decisions. The initial deployment of the NoAH infrastructure utilized honeyd, a low-interaction honeypot, and the Argos high-interaction honeypot for interacting with the malicious traffic received. As part of this EU-funded project, our lab designed, developed and deployed two types of sensors that could be easily deployed at remote locations and forward attack traffic to the NoAH core to be processed by the honeypots. After a long deployment period of the initial NoAH infrastructure, the mo-

tivation to extend and enhance it arose.

The motivation, which led to the realization of this thesis, was twofold:

- We wanted to extend the NoAH architecture in a way that would allow the integration of other honeypots but that would also provide a more stable and easy to maintain NoAH core.

- We wanted to identify a new trend in the way attackers deploy malware, so as to build a honeypot system that can acquire information that is different but complimentary to what the NoAH infrastructure collects.

Initially, the overhead of maintaining Argos, led to the need for a more stable,light-weight and easy to maintain and deploy honeypot. In that light we chose to replace Argos with an alternative honeypot system that would better suit our needs. We chose to use Nepenthes and Amun, two very similar medium-interaction honeypot utilized by many research projects. The first part of this thesis focuses on the NoAH project, its initial architecture and the modifications we performed, as well as an analysis of the attack traffic handled by the new honeypots. We present the attackers that generated the largest volume of attack traffic as well as various characteristics of those attacks.

As new communication channels are used by attackers for propagating their attacks, it is vital for an existing architecture to catch up. The second part of this thesis focuses on the expansion of both the interaction mechanisms of the NoAH architecture, and the amount and type of attack information acquired by it. We develop and deploy HoneyBuddy, as part of the NoAH attack collection mechanisms. HoneyBuddy is an active honeypot infrastructure for detecting malicious activities in IM networks.

HoneyBuddy finds and adds contacts to its honeypot messengers by querying popular search engines for IM contacts or by advertising its accounts on contact finder sites. Our deployment has shown that with more than six thousand contacts we can gather between 50 and 110 malicious URLs per day that belong to 10-15 unique domains. 21% of our collected executable samples were not gathered by other malware collection infrastructures, while 93% of the identified IM phishing domains were not recorded by popular blacklist mechanisms.

1.1 Thesis organization

The rest of this thesis is organized as follows. In Chapter 2 we provide a background on honeypots. In Chapter 3 we present the NoAH architecture. Section 3.1 contains an overview of the Argos, Nepenthes and Amun honeypots, while in Section 3.2 we present the types of sensors deployed for the NoAH project. Section 3.3 presents the techniques used to handle incoming connections and Section 3.4 how we enhanced the existing architecture. In Section 3.5 we present the results from the incoming attack traffic. We continue by presenting the design of HoneyBuddy in Chapter 4. We provide an overview of the different attacks against instant messaging networks in Section 4.1 and the architecture of our system in Section 4.2. In Section 4.3 we analyze the data collected by our infrastructure. We present related work in Chapter 5 and summarize our results and conclude in Chapter 6.

2

Background

A formal definition of a honeypot is a "trap set to detect, deflect or in some manner counteract attempts at unauthorized use of information systems"[1]. Their purpose is to lure attackers. They are non-production systems, which means machines that do not belong to any user or run publicly available services. Instead, in most cases, they passively wait for attackers to attack them. By default, all traffic destined to honeypots is malicious or unauthorized as it shouldn't exist in the first place. Honeypots can also take other forms, like files, database records, IP addresses or e-mails. We can identify four types of honeypots: low-, medium-, high-interaction and client-side honeypots. Each type presents several good characteristics but also a number of drawbacks. No single type is the optimal choice for all situations; each type is suitable for specific problems and complements the other types. For the purpose of placing our work in context, we give an overview of honeypot technology by presenting the different types of honeypots.

[1] `http://en.wikipedia.org/wiki/Honeypot_(computing)`

2.1 Honeypots

Several honeypot designs have been proposed. The two main axes upon which a honeypot is designed is the level of interactivity with the attackers and which side is targeted by the attacks the honeypot will monitor. Concerning the level of interactivity, honeypots can either do simple service emulation (low-interaction), more advanced emulation (medium-interaction) or run real services (high-interaction). As far as the second axis is concerned, we can categorize honeypots as server-side or client-side. The vast majority of honeypots protect the server side. The fundamental difference between the two types is that client-side honeypots search for attackers, instead of waiting to be attacked.

2.1.1 Low-interaction honeypots

Low-interaction honeypots only superficially simulate services. They provide very limited interactivity compared to high-interaction honeypots, but are still useful to lure attackers and gather information at a higher level, e.g., detect network scanning activities. They may emulate parts of an operating system, such as core services and daemons, as well as network applications, like web servers, sshd, FTP servers and more. They can be tuned for interaction with the attacker but lack the ability to trigger previously unknown vulnerabilities of the real services. Low interaction honeypots are more efficient in terms of performance and have less setup and maintenance overhead than high-interaction ones. The most popular low-interaction honeypot is honeyd.

A powerful characteristic of low-interaction honeypots is their ability to enable a single host to claim multiple addresses and run multiple services on each address. Since honeyd, and other similar tools, handles network traffic in a raw fashion, it has the ability to handle a large range of IP addresses. While in high-interaction honeypots having multiple IP addresses is impractical, or even impossible (due to the fact that they need one - physical or virtual - interface per IP address and have to open a socket for each connection they receive), low-interaction honeypots are naturally designed to cope with it.

There are also two practical issues concerning the deployment of low-interaction honeypots that must be considered. The first one is the mapping between physical machines and low-interaction honeypots. Typically, hon-

eyd can emulate a /16 subnet without a significant overhead. For a larger address space, multiple low-interactions should be used. The second one is the security of the machine hosting the honeypot. This machine should exclusively run the honeypot components and no other services. It is a precaution measure to eliminate the risk of being infected because of the underlying operating system.

2.1.2 Medium-interaction honeypots

Medium-interaction honeypots also emulate services but, unlike low-interaction honeypots, they do not manage network stacks and protocols themselves. Instead, they bind to sockets and leave the connection management up to the operating system. In contrast to systems like honeyd, which implement network stacks and protocols, they focus more on the application-level emulation part. The most-well known medium-interaction honeypot is Nepenthes.

2.1.3 High-interaction honeypots

High-interaction honeypots, unlike the two previous categories, do not emulate services. On the contrary, they run services in their native environment. Thus, high-interaction honeypots have the advantage that they are real systems; no emulation is used, no fake services run. Therefore, unknown bugs are still present and can be exploited. By being vulnerable to attacks, they can provide useful information on how previously unknown threats emerge and propagate.

The use of real systems has its drawbacks. First because such a system runs real services and applications, it can be compromised using appropriate real-world attacks. Once compromised, they can be used by the attacker for her purposes, such as participating in worm propagation, DoS attacks, or spamming. Naively blocking outbound communications will not work, if the whole infection phase should be studied. Some attacks involve e.g. code download initiated by the attacked machine. Blaster is such an example. When Blaster infects a host, it then downloads its main executable from previously infected hosts. Second, real systems are not sufficiently transparent for the purpose of monitoring their activity. The internal workings of the operating system and the services may not be readily accessible. Finally, using real systems comes with a performance overhead. While it is reasonable to expect that a single physical machine can support dozens of

low-interaction honeypots, a single real system will require many more resources (at least virtualization software allows multiplexing several ones over a single physical machine). Other research efforts,like Potemkin [52], have focused on bringing the cost down by using aggressive sharing.

A high-interaction honeypot (a real operating system together with services) can be installed inside a virtual machine using virtualization software such as VMware [30], Qemu [37] and Xen [36]. Running a high-interaction honeypot inside a virtual machine offers several advantages. First, multiple high-interaction honeypot instances can be multiplexed on the same physical machine. This is a huge improvement, since most of the honeypots could end up idling, which would waste resources. Second, deployment is significantly easier. Virtual machine environments load the operating system from a disk image. All that has to be done is install and configure the environment once and reuse the image for all high-interaction honeypot instances. Third, recycling operating systems, .e.g. after an infection, is cheap: the environment is ready and the time for starting a virtual machine is much smaller than a typical reboot of a physical machine. The image of the operating system is actually a snapshot of its running state, including its filesystem. Loading the virtual machine with the image will bring up a running operating system within seconds. Finally, virtual machine environments provide the ability to better monitor the guest operating system, in terms of network activity, system calls and resource management.

2.1.4 Client-side honeypots

Recently, exploits that target client applications and especially web browsers have been observed. The WMF and JPEG vulnerabilities [15, 16] have shown how the Internet Explorer browser can be compromised and execute arbitrary code on the victims side. Instead of waiting passively for the attackers to contact them, as we have seen so far, client-side honeypots try to spot locations where malicious content is hosted. Although they are not passive systems, they are still characterized as honeypots as they are non-production systems. Client-side honeypots try to cover the gap of classic detection techniques. According to [28], only 1.5% of IDS signatures are based on client-side attacks, although the number of client-based vulnerabilities rapidly increases over time.

3

Network of Affined Honeypots

In this chapter we will first present the initial architecture and components of NoAH and will then proceed with our modifications. The NoAH project aims to monitor unused portions of the IP address space. The architecture of the NoAH infrastructure was designed in a way that would allow entities other than the project partners to easily contribute IP address space. Thus, the address space covered by the NoAH core can be further extended. Institutes, campuses or organizations can collaborate with NoAH by deploying a "plug" to the NoAH core. This "plug" is actually a tunnel to the NoAH core. All traffic going to the dark space of a collaborating party is tunneled to honeypots in the NoAH core for processing. Replies from honeypots are tunneled back and injected into the party's network. By using tunneling, honeypot deployment on the party's side is not needed and thus the administrative overhead is minimal. Apart from organizations and institutes, simple home users can help NoAH. Home users or small size enterprises can share their black address (or port) space in a similar way as the participating organizations. To do so, they install the honey@home tool.

The NoAH core is not a centralized farm of honeypots. On the contrary, it is a distributed set of honeyfarms that can collaborate. Inside the core, both low- (LI) and high-interaction (HI) honeypots are deployed. Low-

interaction honeypots are used as a traffic filter. Therefore, activities like portscanning can be effectively detected by LI honeypots and stop there. Traffic that cannot be handled by LI honeypots is handed over to HI honeypots. In this case, LI honeypots are used as proxies whereas HI honeypots offer the optimal level of realism. The initial deployment of the NoAH infrastructure utilized the Argos high-interaction honeypot for interacting with the malicious traffic received.

3.1 Backend

The installation and configuration of a honeypot is a manual procedure that requires significant human effort. A honeypot system should relieve administrators from major overheads and run with little maintenance overhead. As there are few solutions for auto-configuration and auto-recovery, a honeypot must minimize the downtime and effort required to keep all its services running and ensure it is not compromised. Due to the high maintenance overhead and stability issues of the Argos honeypot, we decided to replace it with the Nepenthes and Amun honeypots that have the advantage of low maintenance and deployment overhead.

In this section we present a more detailed overview of the honeypots deployed in the backend of the NoAH infrastructure that interact with the traffic forwarded by honeyd. First we provide an overview of the Argos hi-interaction honeypot, and then we present Nepenthes and Amun.

3.1.1 Argos

Argos [47] is a containment environment for worms and manual system compromises. It is actually an extended version of the Qemu emulator that tracks whether data coming from the network is used as a jump target, function address or instruction. To identify such activities, Argos performs dynamic taint analysis[46] (memory tainting). Memory tainting is the process where unsafe data that resides in the main memory or the registers is tagged. All data coming from the network is marked as tainted because by default it is considered unsafe. Tainted data is tracked during execution. For example, if we have an add operation between a tainted register and an untainted one, the result of the addition will be tainted. Before data enters the Argos emulator, it is recorded in a network trace. As Argos has

control of all operations that happen in the guest OS[1], it can detect whenever tainted data is trying to be executed or used as a jump target, e.g. to override a function pointer. When tainted data is to be executed, an alarm is raised and the attack is logged. This log contains information about the attack and specifically registers, physical memory blocks and the network trace. This information is given as input to the signature generation component, which basically correlates information between the memory dump and the network trace using two approaches. The first one locates the longest common sub-sequence between the memory footprint and the network trace. The second one, called CREST, finds the memory location that allowed the attacker to take control of the system. This memory location is found using the physical memory origin and value of the EIP register, that is the instruction pointer register. The value of the EIP register is located inside the trace and then the trace is extended to the left and right. The extension stops when different bytes are encountered. The resulting byte sequence, along with the protocol and port number, is used as a signature. For both approaches, a network trace is useless if data in it is encrypted, for example if it is a HTTPS connection. Latest advances of Argos allow it to correlate memory dumps with unencrypted data, as Argos comes with modified versions of secure socket libraries for some guest operating systems. The signature generation time is linear to the size of the network trace.

The basic advantage of Argos is that it is able to detect without false positives that an automated attack is taking place, regardless of the application under attack or the attack's level of polymorphism. A major drawback of the Argos approach is the performance overhead. An application running in the Argos environment is 20 to 30 times slower than running in its native environment. A large part of this overhead is due to the underlying Qemu[15] emulation. The rest of the overhead is due to memory tainting and the tracking of tainted data. Another major disadvantage is that it has a high maintenance and deployment cost. Furthermore, during its deployment period, we had several stability issues. This lead us to the decision to search for a more stable and easy to deploy component in the NoAH core.

[1]In virtual machine terminology, the guest OS is the operating system running inside the virtual machine while the host OS is the operating system that runs the virtual machine software

3.1.2 Nepenthes

The Nepenthes [19] platform is a system designed to automatically collect malware. Its functionality is based on five types of modules: vulnerability, shellcode parsing, fetching, logging and submission modules. Vulnerability modules emulate the vulnerable services, like a DCOM service or a WINS server. Shellcode parsing modules analyze the payload received by the vulnerability modules and try to extract information about the propagating malware. If such information is found, the fetch modules download the malware from the designated destination and finally the malware instance is submitted to a central service (disk, database, anti-virus company) through the submission modules. The whole process is logged by the logging modules. For the time being, only sixteen vulnerability modules have been implemented for well-known exploits, like a buffer overflow in the Microsoft RPC services, buffer overruns in SQL server 2000 and exploits in the LSASS service. Nepenthes was originally designed to capture malware that spreads automatically, like Blaster or Slammer worms that targeted hosts blindly.

The host running Nepenthes listens to several ports on one or more black (unused) IP addresses. The assignment of these addresses to this host and the creation of virtual interfaces in order to have multiple IP addresses on a single interface must be done by the administrator manually. As the host running Nepenthes listens to many open ports, it is vulnerable to detection. After a connection is established on one of the open ports, the payload of the packets of this connection is handled by the appropriate module. The main restriction here is that for each open port we can only have one vulnerability module. This means that for example we cannot emulate vulnerabilities for both Apache and IIS simultaneously. Vulnerability modules do not provide full service emulation but only emulate the necessary parts of the vulnerable service. When the exploitation attempt has arrived, the shellcode parsing modules analyze the received payload. In most cases, this parsing involves an XOR decoding of the shellcode and then some pattern matching is applied, like searching for URLs or strings like "CreateProcess". If a URL is detected, fetch modules download the malware from the remote location. These modules implement HTTP, FTP, TFTP and IRC-based downloads. However, a shellcode parser can be more complicated. Some malware can, for example, open command shells and wait for commands or bind to sockets.The shell emulation modules of Nepenthes provide command emulation

for the virtual shells. Most shell commands are trivial, like echo or START directives.

The Nepenthes platform has evolved into a distributed network of sensors. Institutions and organizations participate in the mwcollect alliance, where all binaries captured are submitted to a central repository, accessible to all members of the alliance.

3.1.3 Amun

Amun [3] is a medium-interaction honeypot designed to capture autonomous spreading malware in an automated fashion. It operates in a way very similar to Nepenthes, and also bases its functionality on the same five types of modules. The main difference between the two is that Amun may simultaneously run more than one vulnerability module for a specific port, allowing the interaction with a wider range of attacks. Furthermore it is more up-to-date than Nepenthes with a richer selection of vulnerability modules. Finally, Amun is written in Python and therefore allows the easy integration of new features. While we changed the NoAH infrastructure so as to integrate Nepenthes, no changes were necessary to replace Nepenthes with Amun.

3.2 Sensors

In this section we provide a more detailed overview of the sensors deployed for the NoAH project: honey@home and the dark space monitor.

3.2.1 Honey@home

Honey@home [34] is designed simple and lightweight, as it mainly targets typical home users or administrators unfamiliar with honeypot technologies. The functionality of honey@home is to obtain a single unused IP address and forward the incoming traffic to the NoAH core. All the traffic received by the client is tunneled to the NoAH core through a SSL connection. Responses coming from the NoAH core are injected by honey@home into the network so as to reach the originators of the traffic. As honey@home must be as simple as possible, the tunneling process is not performed using third-parties software, like openVPN, but uses a built-in tunneling component (see section 3.3.2) that also supports secure connection. Honey@home clients connect to the communication component (a custom ssl server) of the NoAH infrastructure that only handles traffic from authorized clients that have been verified

by a unique key.

Each honey@home client requests an IP address from the local DHCP server (optionally the user can set a static IP address). Most broadband connection routers, like ADSL routers, assign addresses through a built-in DHCP server that is configured to offer addresses from the block purchased by the user (configuration of the router is performed by the ISP upon connection activation).

3.2.2 Dark Space Monitor

Typically a respectful portion of the IP address space that is allocated to an organization remains unused. This unused IP address space, also referred to as *Dark Space*, can be utilized by the NoAH infrastructure to gather traffic for the NoAH core honeypots. However, the number of honeypots in the NoAH core is limited compared to the size of the dark space. The dark space of a medium-sized organization typically measures a few hundred IP addresses, while for larger organizations it may measure up to thousands of IP addresses. It is unreasonable to expect that for each IP address in the dark space a physical machine will be available to be deployed as a honeypot.

The solution to this size mismatch is to have each physical honeypot handle traffic for a range of IP addresses instead of only one. A single low-interaction honeypot (e.g. honeyd) can be configured to handle traffic for a whole /16 dark subnet, or a high-interaction honeypot can have its network interface configured with multiple IP addresses.

However, honeypot deployment and maintenance is an additional administrative overhead, undesirable to most organizations that do not have the expertise or the resources for such a deployment. Requiring from organizations that wish to cooperate with NoAH to bear this overhead would be impractical.

For this reason, the NoAH architecture enables the redirection of traffic arriving at the dark space of the organization to the NoAH core. By having the traffic redirected and processed in the NoAH core, the cooperating organization can avoid the local deployment of fully fledged honeypots. They will only have to install and run a funnel, which maps all the dark space traffic to a packet forwarding component. This component will be responsible for forwarding packets to and from the NoAH core. Since this forwarding scheme ultimately aims to relieve the cooperating organization from the honeypot related maintenance burden, the forwarding component

itself should require minimal configuration and resources to work.

For the problem of the desired traffic being gathered at one host we use *funneling*. To forward the traffic to the NoAH infrastructure, an extended version of honey@home that handles multiple IP addresses is used, to take advantage of its embedded *tunneling* mechanism. More details about these mechanisms are presented in the next section.

3.3 Handling incoming connections

In this section we present the mechanisms used by the NoAH architecture to handle connections received from the deployed sensors. *Funneling* is used to gather all the connections towards a set of IP addresses on a single machine, *tunneling* is used to forward these connections to the NoAH core, and a *modified* version of honeyd is used to hand-off certain connections to the NoAH backend.

3.3.1 Funneling

The main abstraction of *funneling* is that traffic going to a set of IP addresses will end up being processed by a single machine. The goal is to create a virtual *funnel* that concentrates traffic from portions of dark space on a limited number of honeypots.

When we want a single honeypot to claim traffic for a few tens of IP addresses, it is impractical and resource consuming to do it with means of configuring its network interface for all the addresses. The solution to this inefficiency is arpd. Arpd is a user-space daemon that responds to ARP requests arriving at the network interface of the honeypot. 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. By having arpd reply to these requests, we can effectively direct the traffic for any IPs in the LAN to the host we want, without actually configuring the host network interface for all these addresses.

In order to implement the funnel, the arpd approach is used similarly in the case of local honeypot deployment, so as packets arrive at the dark space they are mapped to the machine running the tunneling software.

3.3.2 Tunneling

As aforementioned, cooperating third parties will act as relay agents: they will forward traffic directed to their dark space to the NoAH core and accordingly the responses sent by the NoAH core back to the original sender of the traffic. A possible solution to achieve this would be to perform address rewriting on the packets, so that they are routed to the NoAH core. This solution has the advantage of being simple and that no modification occurs to the payload of the packets. On the downside, this approach would diminish the dynamics of connections, making detection of certain type of events harder or even impossible. For example, consider an attacker who is scanning the dark space of the cooperating organization. Following the address rewriting approach would require to rewrite all destination IP addresses with the address of the honeypot and inject them in the network. However, in the NoAH core we wouldn't be able to detect this scan as we only see traffic going to a single IP address, the honeypot. Another drawback are communications that use the IP address as data in a higher level protocol. In this case, rewriting the address will not work because addresses will no longer match after the translation.

Therefore, the traffic directed to the dark space must arrive at the NoAH core unchanged. A more appropriate solution to this problem is tunneling. The concept of tunneling is used mainly in virtual private networks. With tunneling, every packet directed to the dark space of the organization is encapsulated in one or more packets and subsequently sent to the NoAH core. Tunneling preserves the single packets as they are received by the tunnel entry-point (no fragmentation is performed by the tunneling mechanism). At the side of the NoAH core, the original packet is decapsulated by the communication component and injected into the loopback interface. The modified honeyd runs on the same physical machine and monitors the loopback and forwards the packets to the Argos backend. The response from Argos follows the reverse procedure: it is encapsulated on the side of NoAH core, decapsulated at the organization and then sent back to the original sender.

3.3.3 Custom ssl server

All traffic received from our deployed sensors is tunneled to the NoAH core and specifically to a host running the tunnel component, a custom ssl server.

The tunnel component has several functions. First, it decapsulates all incoming packets from the network and then injects them in the loopback interface which is monitored by a modified version of honeyd running on the same host. After the attack is processed by the honeypots in the NoAH core, all response packets are forwarded back to the ssl server. Then, the server encapsulates all response packets from the honeypots and tunnels them back to the sensor that received the attack.

3.3.4 Modified honeyd

Honeyd is the most popular low-interaction honeypot. In NoAH it has two functions. First of all it is used as a filtering mechanism. Since high-interaction honeypots are heavily instrumented machines, we need to offload them as much as possible. Low-interaction honeypots are used as a front-end to high-interaction honeypots. Honeyd has the appropriate properties to play the role of the front-end and act as a filtering component. Filtering is defined as any action done to prevent high-interaction machines from being overloaded by packets that will not yield any useful information. As a filtering component, honeyd can absorb any unestablished connections without any overhead. It can also emulate certain services for specific ports.

The second function that honeyd plays in the NoAH infrastructure is to forward all other connections from the deployed sensors to the backend honeypots. The process of connection forwarding is called hand-off. To do so, NoAH utilizes a modified version of honeyd. The hand-off is performed when the connection with the attacker is established. Honeyd opens a socket with Argos and sends all application data received by the attacker through this socket. As application data, we define the TCP (or UDP) payload. To avoid the overhead of creating a new connection for every incoming attack, honeyd sets up a persistent socket with Argos.

The main problem of the hand-off process is that the low-interaction honeypot and the Argos system have different IP addresses. In cases where the application protocol embeds information about IP addresses, as is the case with FTP passive, this mismatch on IP addresses will cause the application to not work properly. To overcome this problem, honeyd replaces all occurrences of its IP address inside packets (either in human readable or binary format) with the IP address of the Argos system. Note that this change cannot be applied generally in any protocol but works well for the protocols we have tested; FTP, HTTP and SSH. Also encrypted protocols

that require IP address information, like secure FTP, cannot be properly handed-off.

3.4 Integration of Nepenthes and Amun

After having deployed the NoAH infrastrucutre for a long period, the need for an easier to maintain and deploy core arose. In an attempt to make the NoAH infrastructure easier to maintain and "tailor" it to our needs we chose to replace Argos with Nepenthes. By replacing the high-interaction honeypot with a medium-interaction one, there would be no more need for the low-interaction honeypot as a filtering mechanism. All incoming traffic could be handled by Nepenthes and, thus, honeyd was redundant. By removing the modified honeyd, we would also lose the component that performs the "hand-off" between the ssl server and the honeypots. As mentioned before, it is unreasonable to dedicate a physical machine for each sensor deployed. Therefore there was need for a new mechanism that would perform the "hand-off" and allow one honeypot to receive and handle traffic from multiple sensors that may have multiple simultaneous connections with attackers. We chose to modify our ssl server and utilize the Unix kernel firewall and network interface, through the iptables, ip route and ifconfig utilities, for these tasks.

For the integration of the Nepenthes honeypot into the NoAH core, we had to modify the ssl server, the component that handles the incoming connections from the deployed NoAH sensors. Specifically, we altered the way the server maps connections from NoAH sensors to the interface where they are "handed-off", and and how it keeps state of these mappings. The ssl server keeps two types of mappings. Each sensor is mapped to a private /16 subnet, a mapping that doesn't change through time, and is also stored in a database. Each IP address monitored by that sensor that receives an attack is dynamically mapped to an IP address of the /16 subnet, a mapping that may change over time. Also the machine that Nepenthes runs on, needs to be specially set-up so as to be able to handle the connections between the attackers and sensors that are forwarded to the NoAH core. Next we present how connections are handled by the new NoAH architecture.

The ssl server runs on a dedicated machine, equipped with two network interface cards (NIC). One is assigned a public IP address, while the other one is assigned a private IP address (10.0.0.1). It receives the encapsulated

packets from all the deployed NoAH sensors, and its role is to decapsulate the packets and perform the "hand-off" operation towards the Nepenthes honeypot. In order to maintain the correct information regarding the targeted IP address monitored by the sensor, the ssl server must maintain a list of mappings from the targeted IP address to a virtual network interface handled by Nepenthes.

Once a sensor first connects to the server, a unique mapping for that sensor is created. All mappings are saved in a database and remain unaltered over time. Since a sensor may monitor up to a /16 subnet, the ssl server maps the sensor to a private /16 subnet (10.xx.0.0) and can, thus, maintain the information of which dark IP address was targeted. Each monitored dark space IP address that receives a connection is dynamically mapped to an IP address of the corresponding private /16 subnet (10.xx.0.yy) which may change over time. All connections received are forwarded to the mapped private IP address on the NIC of the Nepenthes host.

The machine that hosts the Nepenthes honeypot, is also equipped with two network interface cards. One is assigned a public IP address, while the other one is assigned a private IP address (10.0.0.2). We create 251 virtual interfaces on the second interface (from 10.0.0.3 to 10.0.0.253). Each virtual interface will be assigned to one sensor. Thus, each Nepenthes host can handle up to 251 sensors. The NIC with the private IP address has a direct physical connection to the NIC (with a private address) on the machine running the ssl server, for the server to be able to forward packets to that private interface.

On the Nepenthes host, we use iptables to create a prerouting rule for the (network-address-translation) NAT table, so as to have a NAT rule that translates the packages sent from the ssl server (destined to a private IP address 10.xx.0.yy) to a virtual interface (10.0.0.xx). This way we take advantage of the operating system's embedded functionality and are able to forward multiple connections from one sensor without losing any information. Nepenthes will receive all connection information from a sensor on a specific virtual interface, and create the appropriate response packages. The NAT mechanism allows the handling of multiple connections to one sensor by Nepenthes without losing the information of the original destination IP address.

However, it is important to route the appropriate packets from Ne-

penthes, through the private interface and not the public one which is the default. Otherwise, the response packet will be sent to the attacker directly from the Nepenthes host and not the honey@home sensor. Therefore, we create 2 more iptable rules (one for TCP and one for UDP packets) that use the mangle utility and mark all outgoing packets that have a source port that belongs to the ports handled by Nepenthes with a special mark value of 2. Then the ip route utility is used to create a routing policy that forwards all packets with a mark value of 2 to the private interface. Therefore, only packets generated by Nepenthes as responses to attack traffic will be tunneled back through the NoAH infrastructure, and not any other traffic from the host. This allows the Nepenthes download modules to use the public interface to download malware binaries, after extracting the appropriate information from the attack traffic. For the ssl server to be able to receive the packets from the private interface of the Nepenthes host, we run the arp utility on its private interface.

An example of how connections are handled by the new architecture can be seen in Figure 3.1. First, assume a honey@home client that monitors 10 consecutive IP addresses: from "1.2.3.1" to "1.2.3.10". Next an attacker with an IP address of "5.6.7.8" sends an attack packet to the IP address "1.2.3.4" which is handled by our honey@home sensor. Honey@home then encapsulates the packet and tunnels it to the ssl server which has previously authenticated the client. If the server has never received a forwarded packet from the particular client it will create a new mapping from the client to a virtual interface of the Amun host. Here it already has been mapped to "10.3.0.0". The ssl server then checks which IP address has been mapped to the particular IP address monitored by honey@home. Here "1.2.3.4" has been mapped to "10.3.0.20". The ssl server decapsulates the packet and changes the destination IP address to "10.3.0.20" and sends the packet to the private interface host running Amun. The NAT prerouting rule translates the packet destination IP address to "10.0.0.3" which belongs to one of the virtual interfaces monitored by Amun. Amun will process the packet and create a response packet. The iptables rule that marks all out-going packets that have a source port monitored by Amun will initially mark the packet, which will then be routed by ip route to the private interface. From there it will be received by the ssl server since it runs an ARP daemon that answers all arp requests. Next, ssl server will overwite the source IP address based on

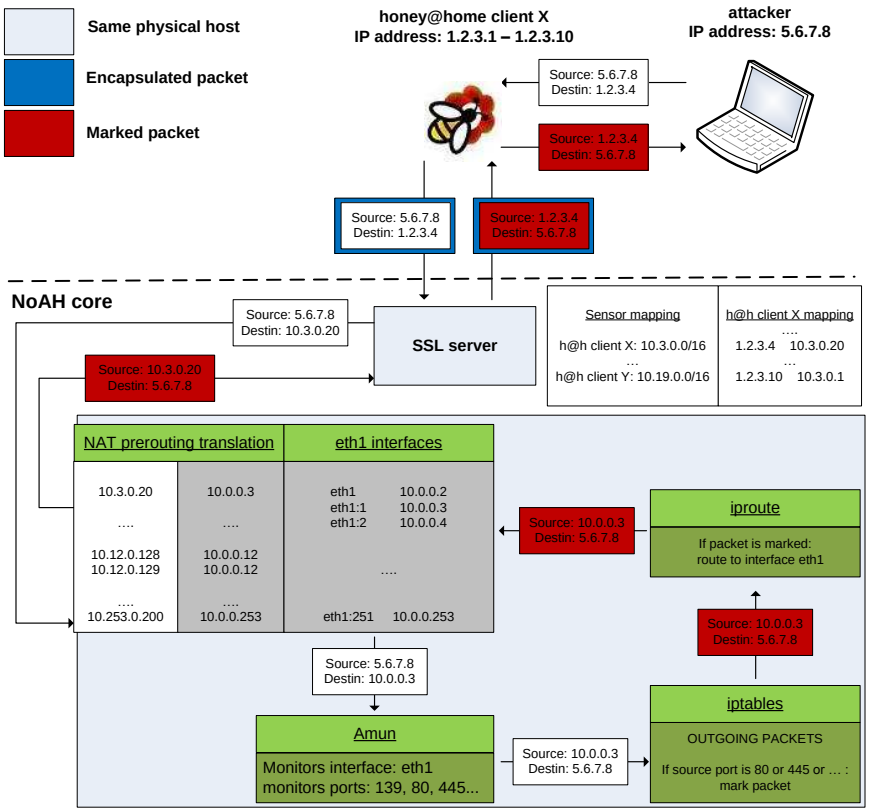

Figure 3.1: Example of a connection being forwarded to the NoAH core, and handled by Nepenthes.

the mappings information, in this case with "1.2.3.4", encapsulate the packet and tunnel it back to the honey@home sensor. The honey@home client will decapsulate the packet and send it to the attacker. The attacker receives the packet which she believes to have been generated by the vulnerable host.

After deploying Nepenthes, and monitoring the results over a period of 4 months, we replaced it with Amun since it offers a larger selection of vulnerability modules and is more up-to-date. Amun is very similar to Nepenthes so there was no need for further changes to the NoAH architecture for Amun to be integrated. All that had to be done was to include the extra ports handled by Amun in the iptable rules. The measurements regarding the received traffic being handled by Nepenthes and Amun are presented in the next section.

The benefits from the modifications we performed are several. First, and foremost, by removing Argos and honeyd and utilizing Amun and the

Unix utilities, our infrastructure is much more stable, robust and scalable. Next, we alleviated the problem of Argos' administration overhead. Finally, the Amun logging modules provide much more information both for known attacks and unknown vulnerabilities.

3.5 Measurements

In this chapter we will present the results from integrating the Nepenthes and Amun honeypots into the NoAH infrastructure. First we will present the traffic handled by the Nepenthes honeypot during its 4 month deployment. Next we will present the statistics of the traffic handled by Amun during its 2 month deployment.

3.5.1 Traffic handled by Nepenthes

The distribution of the conversations handled by Nepenthes are shown in Figure 3.2. During the 4 month deployment period, March to July 2009, Nepenthes handled a total of 498,689 conversations with attackers that targeted the NoAH sensors. The maximum number of conversations handled in one day was 84,697 which occured on the 16th of July.

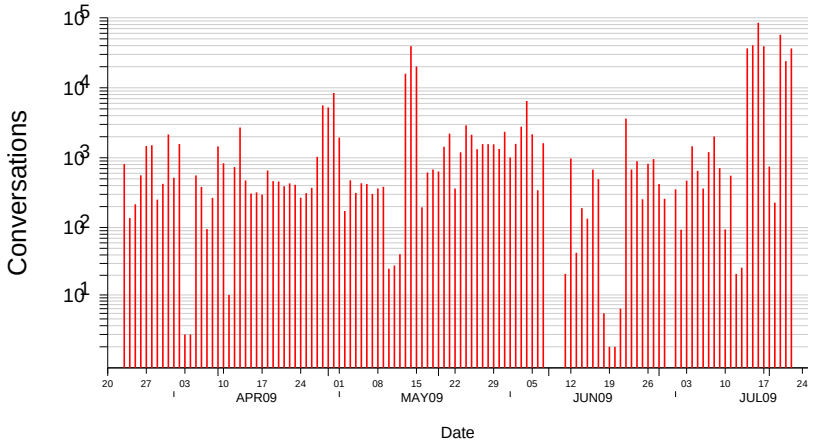

Figure 3.2: Conversations with attackers, handled by Nepenthes.

Table 3.1 presents the top 10 source countries of attackers that initiated conversations with the NoAH sensors. The second column shows the aggregated number of conversations, initiated from attackers placed in that country. For the geographic location of the attackers' IP addresses we used the

Country	# Conversations
Greece	62,918
USA	62,453
Russia	49,076
Turkey	24,279
Germany	21,962
Canada	20,267
Brazil	20,057
China	17,503
Italy	17,080
Taiwan	16,907

Table 3.1: Top 10 source countries of attackers that targeted the NoAH sensors.

IP Address	# Conversations	Country	Days
89.19.15.18	18,855	Turkey	12
72.158.69.42	11,901	USA	2
207.182.157.2	10,358	USA	4
62.1.78.43	10,171	Greece	3
189.144.251.108	5,134	Mexico	2
80.73.6.161	4,814	Ukraine	1
164.77.152.10	4,156	Chile	1
118.173.19.140	3,734	Thailand	1
196.219.191.229	3,535	Egypt	5
67.246.220.245	3,408	USA	1

Table 3.2: Top 10 attackers that targeted the NoAH sensors.

MaxMind [2] database. The aggregated conversations from these 10 countries amount to 62.6% of the total conversations handled by Nepenthes. Furthermore, the results are somewhat unexpected. While we expected Greece and the US to be among the top countries in the list, China ranks 8th which is much lower than expected.

In Table 3.2 we can see the statistics of the top attackers for the whole duration of the Nepenthes deployment period. An interest observation is that while most attackers were active only for a few days, which indicates that the attackers' IP addresses changed afterwards or the machine might have been "disinfected", the top source IP address targeted our sensors on twelve distinct days over a period of 6 weeks.

In Figure 3.3 we can see which ports were targeted the most by attackers. Port 445 was the destination port for over 385 thousand conversations initiated by attackers. This port is technically known as "SMB over IP" and consists a significant threat for users, since attackers can exploit vulnerabilities and silently upload and run any programs of their choosing without the computer's owners ever being aware. Second we can see that port 80, the standard port for websites, attracted over 49 thousand conversations. Websites may have a lot of different security issues. These holes can allow an attacker to gain either administrative access to the website, or even the web server itself. Next, port 139 which is used for TCP NetBIOS connections was attacked more than 33 thousand times. Port 139 is one of the

[2] http://www.maxmind.com/app/geolitecity

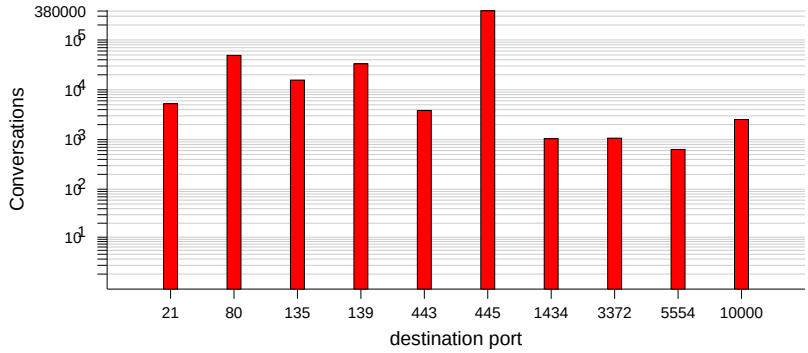

Figure 3.3: Top 10 destination ports that received the most traffic by attackers.

most common ports targeted by attackers, and so we would expect to find it at a higher rank. Attacks targeting port 1434 are most likely due to worms probing for vulnerable Microsoft SQL servers.

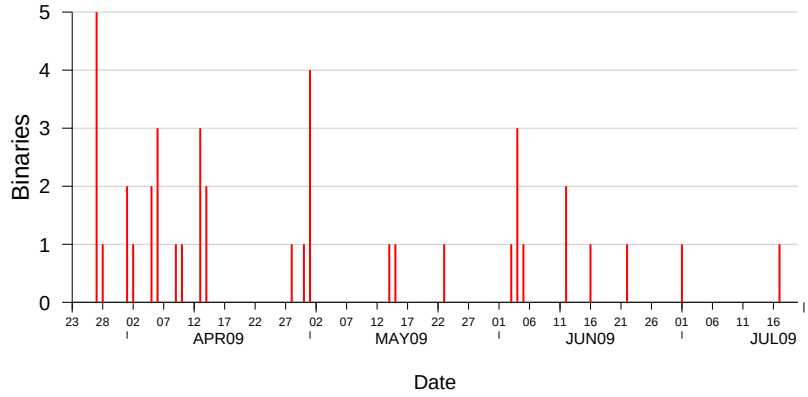

Figure 3.4: Binaries collected by Nepenthes.

In Figure 3.4 we can see the unique malware binaries collected by Nepenthes during the 4 month deployment period. Overall, 41 unique samples of malicious software were downloaded from Nepenthes based on information extracted from the incoming attack traffic. The largest number of unique samples downloaded in one day, was on the 27th of March when 5 different binaries were downloaded by Nepenthes.

Table 3.3 shows the countries of origin of the attackers from which Ne-

Country	# Binaries	Country	# Binaries
Japan	10	Netherlands	1
USA	9	Italy	1
Taiwan	5	Israel	1
Germany	3	Honk Kong	1
France	2	Greece	1
Spain	2	United Kingdom	1
Canada	2	Denmark	1
Venezuela	1		

Table 3.3: Source countries of attackers that sent malware binaries.

penthes was able to extract information and download malware binaries. Attackers traced back to Japan sent the most unique binaries, while attackers from the US sent 9 unique binaries. It is surprising, once again, that no new binaries were received from attackers located in China.

3.5.2 Traffic handled by Amun

The distribution of the conversations handled by Amun are shown in Figure 3.5. During the 2 month deployment period, from the end of July to September 2009, Amun handled a total of 153,082 conversations with attackers that targeted the NoAH sensors. The maximum number of conversations handled in one day was 22,268 which occured on the 17th of September. We must note that during August, a large number of the NoAH sensors were down, resulting in a inaccurate view of the attack landscape for that period.

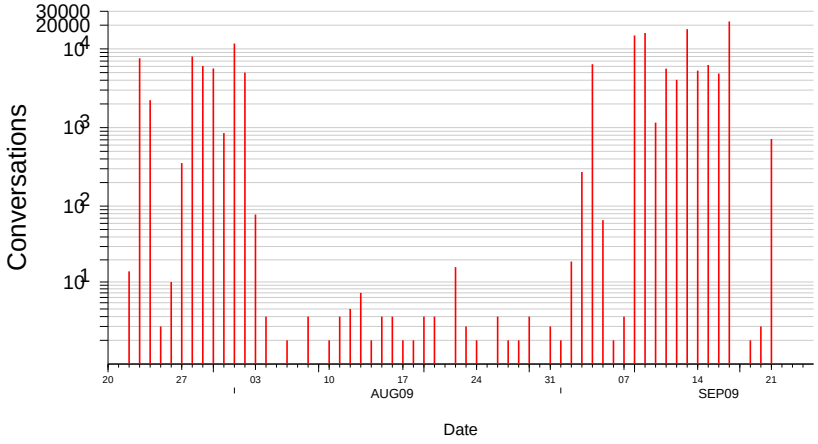

Figure 3.5: Conversations with attackers, handled by Amun.

Country	# Conversations
USA	56,297
Russia	22,807
Taiwan	18,846
Argentina	9,325
Japan	6,656
Malaysia	4,193
Moldova	3,474
Portugal	3,059
Bulgaria	2,663
China	2,362

Table 3.4: Top 10 source countries of attackers that targeted the NoAH sensors, and were handled by Amun.

IP Address	# Conversations	Country	Days
99.14.215.83	14,495	USA	1
173.24.92.207	7,862	USA	2
89.178.247.14	7,279	Russia	1
63.151.109.189	7,260	USA	2
114.137.101.25	5,327	Taiwan	2
122.125.98.156	4,851	Taiwan	1
190.50.56.118	3,798	Argentina	1
96.243.101.211	3,743	USA	3
190.176.42.29	3,626	Argentina	1
12.183.171.197	3,580	USA	2

Table 3.5: Top 10 attackers that targeted the NoAH sensors, and were handled by Amun.

Table 3.4 presents the top 10 source countries of attackers that initiated conversations with the NoAH sensors. The aggregated conversations from these 10 countries amount to 84.8% of the total conversations handled by Amun. Results show that the United States are the country that initiated the largest number of conversations with the NoAH sensors. During this period the number of attacks received from Greece are negligible. China ranks 10th which is again much lower than expected and quite surprising.

In Table 3.5 we can see the statistics of the top attackers for the whole duration of the Amun deployment period. It is interesting to note that all top attackers were active for a few days, and in all cases they were consecutive. This differs from the previous results from the first collection period where the same IP addresses were seen across several non consecutive days.

In Figure 3.6 we can see which ports were targeted the most by attackers. Port 445 was again the most targeted port receiving over 130,000 conversations initiated by attackers. The second most popular port is again port 80 which attracted over 11 thousand conversations. The hierarchy of the popular ports is very similar to that presented in the previous section with the top ports being the same. However, we can identify two new ports. Attack traffic targeting port 2967 exploits a vulnerability in the Symantec antivirus. Attacks with a destination port of 3127 are very popular amongst worms.

In Figure 3.7 we can see all malware binaries collected by Amun during

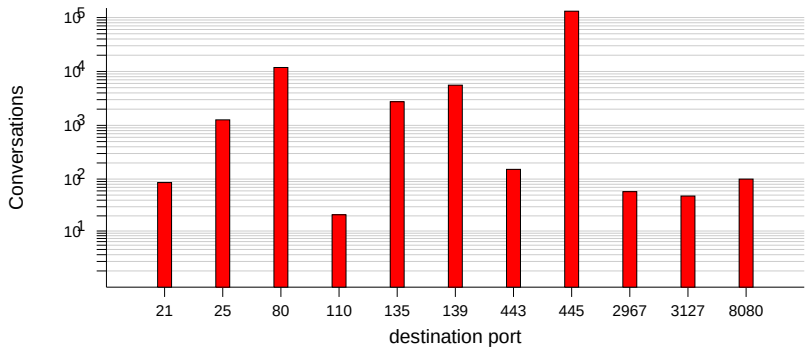

Figure 3.6: Top 10 destination ports that received the most traffic by attackers.

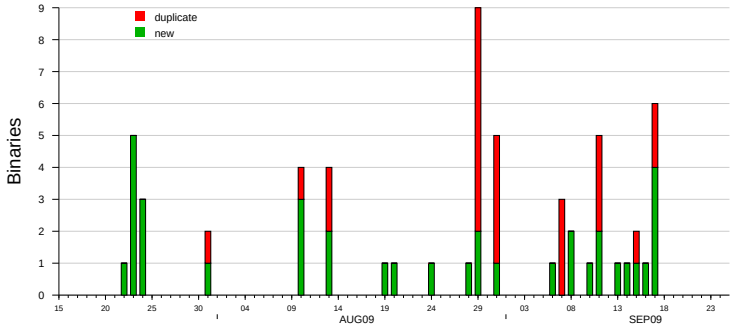

Figure 3.7: Binaries collected by Amun.

the 2 month deployment period. The green portion of the bars indicate the number of malware samples received on that day, that had not been collected by Amun before. Overall, 60 samples of malicious software were downloaded from Amun based on information extracted from the incoming attack traffic, 36 of which had not been collected before. The largest number of unique samples downloaded in one day, was on the 23rd of July when 5 different binaries were downloaded by Amun, while the day with largest number of malware samples overall was on the 29th of August with 9 samples totally.

Table 3.6 shows the countries of origin of the attackers from which Amun was able to extract information and download malware binaries. Attackers traced back to the Republic of Korea sent the most binaries, reaching a total of 27. This comes as a shock, because the Republic of Korea had not sent any malware before, but managed to send a total of 27 samples on 5 different

Country	# Binaries	Country	# Binaries
Republic of Korea	27	Japan	4
China	9	Canada	1
Russia	8	Taiwan	1
Iran	5	Sweden	1
USA	4		

Table 3.6: Source countries of attackers that sent malware binaries.

days. During this period we received 9 malware samples from China, and we also received samples from countries that had not sent malware to our infrastructure while Nepenthes handled incoming traffic, such as Iran.

4

HoneyBuddy

In this chapter we present HoneyBuddy, the active honeypot system we designed after identifying new trends in the way attackers deploy their malicious schemes. Our goal was to design and deploy a system that would acquire information regarding cyber-attacks that use new channels for propagating. This would help in further expanding the attack information and malware samples we acquire from the NoAH project after the integration of the Amun honeypot.

Instant messaging is one of the most popular Internet activities. According to an older survey [10], more than 82 million people in Europe and 69 million people in North America use an instant messenger. A more recent study by Leskovec et al. [41] reveals that the number of MSN messenger (the most popular IM client) users has reached 240 million, with 7 billion exchanged messages per day. Reports estimate over 400 million registered Skype users [23], and 2.1 billion instant messages sent per day by AIM users[5].

The large user-base and the fact that IM is a near real-time form of communication, in contrast to other forms such as e-mail, make IM networks an attractive platform for attackers to launch their campaigns. Attackers either exploit vulnerabilities of the IM client software, or steal account information

through phishing schemes. Once a user account has been compromised, the attack propagates by targeting the victim's contacts. The attack vectors are either file transfers or instant messages that contain URLs of websites controlled by the attacker. As users tend to trust content sent from their contacts, the probability of users accepting the transfer or clicking the URL is higher than in the case of traditional phishing campaigns or malicious websites.

4.1 Attacks on Instant Messaging networks

The high population of IM networks makes them an attractive target for attackers that try to exploit them for malicious purposes, such as spreading malware and scamming. We identify four different scenarios of attacks on IM networks.

Malware infection. Recent malware instances [32] can attach to a victim's instant messaging client and start sending URLs that point to malicious websites, or spread themselves by sending executables. In the most common case the malware instance logs in to the IM network, randomly selects users from the victim's contact list, sends the malicious URLs or files and then immediately logs out. In order to be more appealing to potential victims, the URLs point to domains whose name contains the username of the recipient, for example `http://contact_username.party-pics.com`. All attack campaigns we have detected send messages in English. However, we believe that attackers will soon send localized messages, as is the case with one localized phishing site that we have detected.

Compromised accounts. Attackers can also use compromised credentials to log in as several different users and flood users that are in the victims' contact lists. Many services, like MSN, use unified credentials for e-mail and instant messaging, making life easier for attackers. Attackers can harvest IM accounts either by setting up phishing sites for the service, by planting key-loggers or through social engineering. A relatively known attack campaign is that of websites advertising a service that can reveal to users if someone has blocked them. If the user enters her IM credentials in the website, she is redirected to another domain where nothing happens. Later on, the phishing site owner logs in as the user and sends messages to the victim's contact list.

Exploiting weak privacy settings. Even in the absence of malware infection or stolen credentials, some messengers provide the option to allow incoming messages from people who are not in the user's contact list. We tested the latest client versions of the most popular IM services: MSN live messenger, Skype, Yahoo and AIM. MSN live messenger is the only IM client we tested that has a privacy setting enabled by default that blocks messages from accounts not contained in the contact list. Skype, Yahoo and AIM by default allow anyone to send instant messages to our account, but this setting can be opted-out. Attackers exploit these settings to send unsolicited messages to IM users.

Exploiting client software. IM client software suffers from the problem of monocultures. Once an exploit is discovered, then automatically millions of clients can be infected immediately [31]. While in the case of malware infection exploits take advantage of the IM client to spread, this case involves the attack where the IM client is used to infect the rest of the machine.

4.2 Design and implementation

HoneyBuddy was designed taking into consideration the attack scenarios described in section 4.1. In contrast to previous work, HoneyBuddy does not use modified versions of open source alternatives. It rather uses the latest version of the original clients, the same software most users install. The main reason for this choice is that direct attacks on IM client software will be detected. The basic concept behind HoneyBuddy is to add random accounts to a decoy IM account and monitor the incoming connections. As HoneyBuddy is in fact a honeypot, any incoming communication is by default suspicious. For our prototype we chose the MSN service due to its popularity. However, the design of HoneyBuddy is generic enough to allow the fast implementation of other services as well, like AIM, Skype and Yahoo messengers. Furthermore, MSN live messenger 2009 inter-operates with Yahoo, and is planned to introduce interoperability with Google Talk, AIM and other services, rendering our architecture deployable to all major instant messaging services. All deployed messengers run in a fully patched Windows XP SP3 system.

4.2.1 Architecture

HoneyBuddy has three main components; a harvesting module, a script-based engine that handles the MSN messenger clients and the inspection module.

The harvesting module is responsible for gathering accounts that will later be added to the decoy accounts. All harvested accounts are inserted in CTT files (MSN contact files) that are imported in the messengers and all accounts listed are automatically invited. Another way is to search for e-mail addresses that belong to the @hotmail.com and @live.com domains. Other potential sources are sites where users advertise their MSN account, such as [14]. A more advanced method is to harvest account names from popular social networking sites.

The script-based engine starts the messengers and invites all contacts gathered from the harvesting module. Based on the AutoIt software [6] , we can automatically start the application, import CTT files and invite other accounts to our friend list. The AutoIT software allows the manipulation of the windows of an application the same way a user would manually click, monitor the status of the application and check for new windows (in order to check for incoming messages). When an incoming message comes and includes a request for a file transfer, the engine automatically accepts the transfer. As each messenger can only have a limited number of friends in its contact list, it is preferable to run multiple messengers. For resource efficiency reasons, we used MSN Polygamy [18] in order to run multiple MSN messengers on a single platform without the need of virtual machines.

The inspection module monitors the logs of the messengers for malicious URLs. It additionally checks the default download folder for new file transfers. An interesting finding is that we received URLs and malware in the Hotmail inboxes of our accounts. Thus, we extended the inspection module to also fetch and analyze e-mails, so as to extract URLs and attachments. All malicious URLs are stored in a database and are queried every one hour to check their uptime status.

4.2.2 Contact sources

We used two major sources for finding and adding contacts. The first one was queries for contact files and e-mail accounts belonging to the @hotmail.com and @live.com domains. Simple queries like “filetype:ctt msn” or

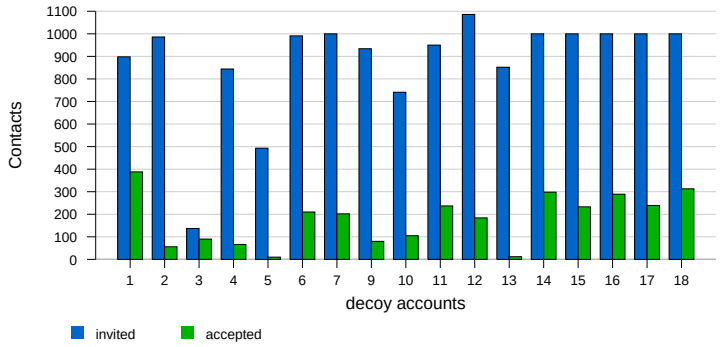

Figure 4.1: Number of friend invitations sent and number of accepted invitations, per decoy account.

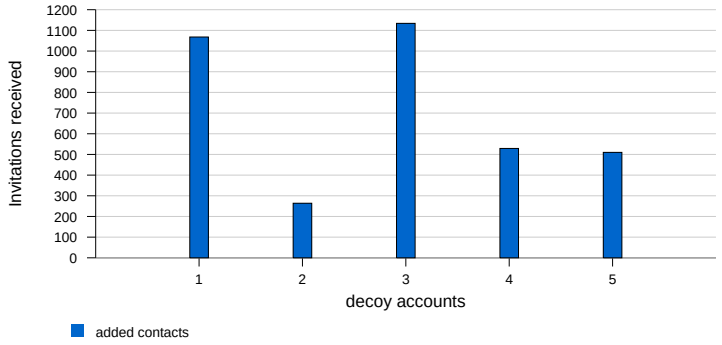

Figure 4.2: Number of invitations our decoy accounts received and accepted after being advertised on messengerfinder.com

"inurl:'@hotmail.com" were able to provide us with thousands of contacts. We also harvested e-mail accounts from other popular sites like `buddyfetch.com`[7], from which we extracted 38,000 hotmail addresses. Overall, we have invited 14,912 contacts to become friends with our accounts. 3,012 of those (20%) accepted our invitation. The exact number of invitations and acceptances per decoy account is displayed in Figure 4.1. The five decoy accounts denoted in Figure 4.1 as decoy accounts 14 to 18, sent a thousand invitations each, to addresses extracted from `buddyfetch.com`. We plan on adding the remaining accounts to our system in the near future. More advanced methods of harvesting [22] can be based on popular social networking sites like Facebook. By crawling such networks, one can collect IM accounts from

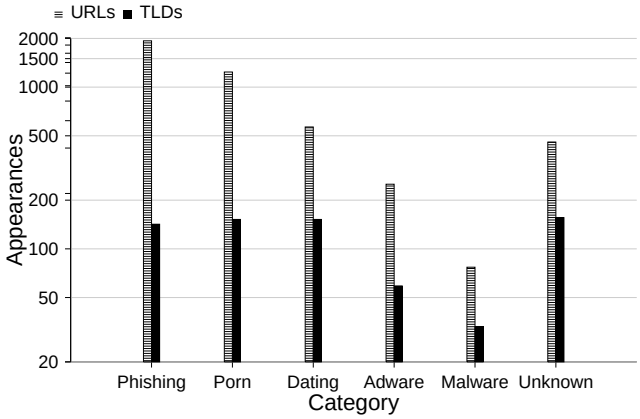

Figure 4.3: Classification of collected URLs

accessible profile pages.

Other potential sources are sites where users advertise their MSN account, such as `messengerfinder` [14]. The `messengerfinder` site contains more than 25,000 active messenger contacts that are advertised by their owners for social networking purposes. We advertised our accounts on this site and instructed our honeypot messengers to accept any friend request. So far, we have added 3,505 contacts while this number increases daily. The exact number of contacts per decoy account is shown in Figure 4.2.

4.3 Collected data analysis

In this section we provide an analysis of data collected by the HoneyBuddy infrastructure, from the 27th of February to the 16th of September 2009, unless stated otherwise. Despite the technical simplicity of our system, we were surprised by the fact that popular defense mechanisms had not detected the majority of our collected data. During the collection period, the HoneyBuddy infrastructure collected 6,966 unique URLs that belong to 742 unique top-level domains.

During the first weeks of HoneyBuddy operation we were able to fetch all URLs through the *wget* tool. However, malicious sites changed their behavior to avoid these fetches. Their pages now serve an obfuscated piece of Javascript code that changes the window location to a URL like `http://www.malicious.com/?key=<randomkeyhere>`. If a user has not visited the page with the key, then all subsequent requests are ignored and eventually

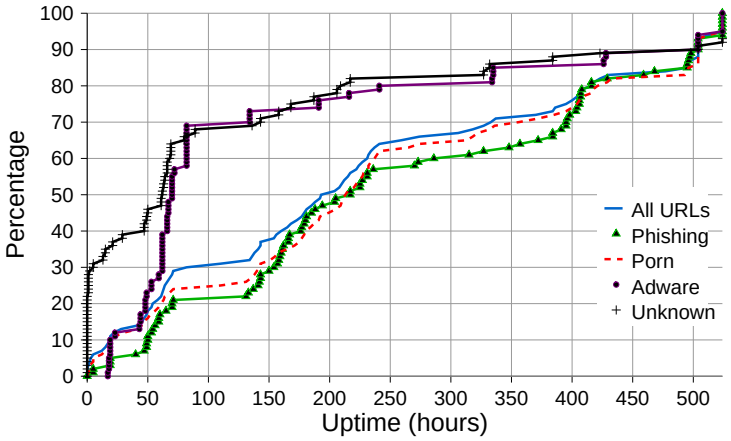

Figure 4.4: CDF of uptime of URLs per category

her IP address is blocked for 24 hours. This behavioral change has forced us to fetch URLs through the Crowbar [9] environment that allows running javascript scrapers against a DOM to automate web sites scraping.

Our first step was to provide a simple classification for those URLs. Our five major categories were phishing, porn, dating, adware [1] and malware. The results are summarized in Figure 4.3. 1,933 of the URLs in 142 top-level domains were phishing for MSN accounts, 1,240 were porn, 567 were dating services and 251 sites were adware. While porn, dating and adware can be considered as harmless, the phishing sites pose a security danger for users. Furthermore, 77 URLs redirected to executable files or to pages that contained a direct link to a ".exe" or ".scr" file. We classify these URLs as malware.

We also spotted several sites that advertise subscription-based services for mobile phones. When the users enter their mobile phone number, they receive a subscription request. Once they subscribe to the service, they get charged for receiving SMS messages. These sites claim to give away free mobile devices to the subscribers of the service or promote quiz games that may attract victims, such as love calculators etc. These sites are highly localized. We visited them from different geographic locations using the Planetlab infrastructure [21] and got different pages in the language of the

[1] We characterize sites that promote third-party addons for the MSN messenger (like extra winks, emoticons etc.) as adware sites

origin country. An interesting fact is that when the site cannot find the geolocation of the user, it redirects her to an MSN phishing site.

Our second step was to analyze the uptime of the collected URLs. The uptime graph can be seen in Figure 4.4. On average, a site is functional approximately for 240 hours (10 days). We also plotted the uptime graph for each category. We notice that porn and MSN phishing sites present much higher uptime than adware and unclassified sites. Half of the MSN phishing sites were alive for up to 250 hours (ten and a half days), while adware present a shorter lifetime of up to 80 hours (three and a half days).

4.3.1 MSN phishing

Attackers try to gather MSN credentials by tricking the user into entering her MSN e-mail and password in a bogus site. These sites falsely advertise a service that will reveal to the user which accounts from her contact list have blocked her. To validate that these phishing sites actually steal user credentials, we created several MSN accounts and entered them into the phishing sites. Each account had one of our decoy accounts as a friend. The decoy account received messages from the stolen MSN accounts that advertised the phishing site. However, the attackers did not change the passwords of any of the compromised accounts.

All phishing sites we visited shared one of three different "looks". We analyzed the source HTML code of all the three "looks" and there was absolutely zero difference among the pages with the same look. This means the phishing pages with the same look had the exact same size and contained the same images and forms. This indicates that the majority of the different phishing campaigns might be deployed by a number of collaborating attackers. We also detected a localized phishing site which contained translated content, a technique used in e-mail spam campaigns[24]. The number of syntactical and grammatical errors revealed that the text translation was done automatically. For the time being, simple string matching for specific text segments is efficient for detecting these sites. Another detection mechanism is to query the various URL blacklists.

We queried the Google blacklist through the Google Safe Browsing API [11] to check if it included the phishing sites we discovered. From the 142 unique top-level domains (TLD) that hosted phishing sites and were detected by HoneyBuddy, only 11 were listed by Google blacklist. That means that 93% of the domains captured by HoneyBuddy were not listed elsewhere,

making HoneyBuddy an attractive solution for MSN phishing detection. The average delay from when our system detected one of the 11 sites until it was included in the Google blacklist was around two weeks, leaving a time window of 15 days for attackers to trick users. Firefox, one of the most popular browsers uses the Google Safe Browsing API as an anti-phishing measure. We also compared our findings with the blacklist maintained by SURBL [26] and URLblacklist.com [27]. SURBL detected only 1 out of the 142 MSN phishing domains (0.7%) and none of the adware domains. None of the phishing or adware sites were listed by URLblacklist.com.

A very interesting fact was that when resolved, all the unique top level domains translated to a much smaller number of unique IP addresses. This fact confirms our initial theory that all these phishing campaigns lead to a limited number of collaborating attackers. To further investigate this behavior, we conducted an experiment for a period of almost two months, presented in Section 4.4.

4.3.2 Malware sample analysis

In this section we provide an analysis of the malware collected by the HoneyBuddy infrastructure, from the 1st to the 31st of March 2009. Our infrastructure collected 19 unique malware samples. We distinguish the malware collected by the HoneyBuddy infrastructure into two categories, the direct malware set and the indirect malware set. We present the two categories and proceed to further analyze the collected samples.

The first category contains malware samples collected either through direct file transfers (uncommon case) or by visiting URLs that were redirected to executable files. In the case of the URLs, the e-mail account of the victim was always appended as a parameter to make it look more realistic. In some cases attackers used popular keywords, like Facebook.

The second category, the indirect one, contains malware samples collected in two types of cases. In the first case, users are presented with a web page that alerts them that they need to download the latest version of the "adobe flash plugin" so as to play a certain video strip, and are prompted to download the installer which is, obviously, malware. In the second case, users are redirected to a page prompting them to install a screen saver. This ".scr" file they are prompted to download and install is a malicious file that infects the machine upon execution.

Due to the small volume of files, we were able to manually check these

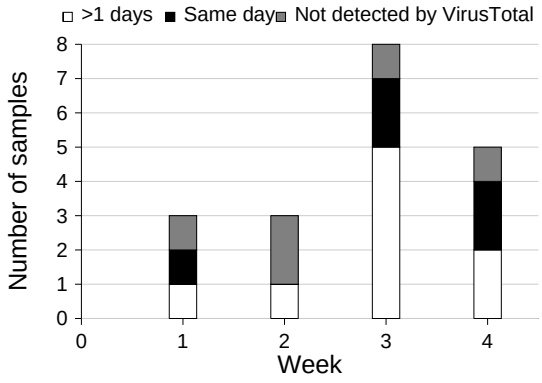

Figure 4.5: Detection delay of collected samples compared to the VirusTotal database. 21% of the samples were previously unseen, while 26% were collected the same day they entered the VirusTotal database.

files using the Anubis analysis center [4]. All of them were characterized as dangerous , while some of them were bots that connected to an IRC C&C server.

In order to verify how original our samples are, we submitted them to the VirusTotal [29] service. VirusTotal is a large malware collection center with the primary goal of providing a free online virus and malware scan report for uploaded samples. Every day VirusTotal receives around 100,000 samples. Four collected samples had not been seen by VirusTotal before, that is 21% of our samples were previously unseen malware instances. Figure 4.5 shows the relative detection delay compared to the date the samples entered the VirusTotal database. The base bar of the stack graph (solid white) shows how many samples were detected with a delay of one or more days, the middle bar (solid black) displays the number of samples that were detected the same day as VirusTotal while the top bar shows the number of samples not included in the VirusTotal database. Five samples (26%) were collected the same day they entered the VirusTotal database, while the maximum detection delay was five days.

We also checked the VirusTotal analysis reports for the collected samples. 42% of the samples were detected by half of the anti-virus engines, while the maximum detection rate was 77%. However, the dates of the analysis reports were one month after the collection date as we did not submit the samples the day they were captured. The one month delay means higher

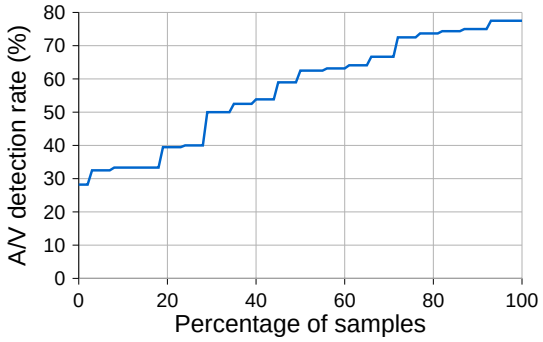

Figure 4.6: Cumulative distribution function of detection rate for collected samples based on VirusTotal reports. 42% of the samples were detected by 50% of the anti-virus engines.

detection rates for the anti-virus engines as they update their signature files daily. Even in that case, it can be observed that there are samples that are recognized by only one third of the anti-virus products. The cumulative distribution function of detection rates can be seen in Figure 4.6.

4.3.3 Mailbox Analysis

In this section we present an analysis of the emails we found in the mailboxes of our decoy accounts. Our analysis focuses on two aspects of the incoming emails. First, whether the body of the email contains any URLs and, second, whether the email contains any attachments. The decoy accounts received a total of 4,209 emails, 403 of which contained a total of 1,136 attachments. The emails contained 5,581 URLs which were passed to our classifier. The goal of the classification was to only identify phishing URLs and URLs that downloaded malware. 26 of the received URLs belonged to phishing domains while 7 downloaded malware samples.

While the majority of the attachments were pictures, several were windows media files and office documents. We checked the VirusTotal database for the MD5 hashes of the files but found no matches. This was expected, since hotmail scans incoming emails for malware and blocks executables. The most interesting attachments were two ".zip" files. Once extracted, the zip files returned a ".lnk" file. Upon inspection, we found that the files were command line scripts that connect to an FTP site and download and execute malicious software. For a more detailed analysis of the file refer to

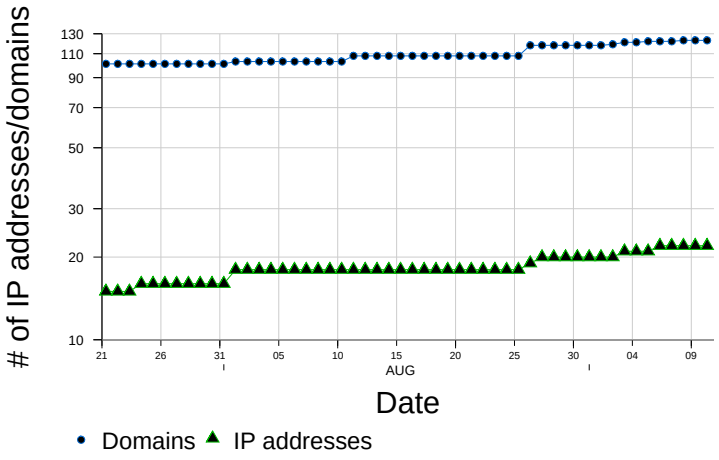

Figure 4.7: Number of distinct phishing domains and the IP addresses they resolve to over time.

this report by F-Secure [12].

4.4 Hosting analysis

The fact that all the top-level domains of the URLs collected from our initial experiment translated to a very small number of IP addresses, urged us to conduct a new experiment that might reveal more information. For a period of 50 days during July and August of 2009, we periodically ran *nslookup* for each of the unique top level domains our system had collected up to that moment, in order to gather more information regarding how and where attackers host phishing and malware-distributing domains. Here we present the results for each category of domains separately and highlight their particular behavior.

The experiment gave us further insight in regards to the small number of IP addresses that host a multitude of phishing campaigns. All top level domains translated to one or two IP addresses, while 98% of them translated to only one. Furthermore, ten of the top level domains belonged to fast-flux networks and translated to a different set of IP addresses each time. In Figure 4.7 we can see that during the first days of the experiment, all 101 top-level domains translated to only 14 different IP addresses. The TLDs that belonged to fast-flux networks are excluded from the graphs. This behavior is consistent throughout the duration of the experiment and as

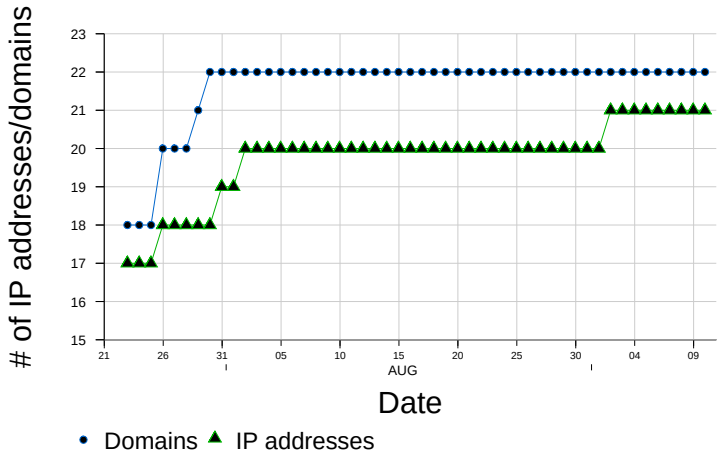

Figure 4.8: Number of distinct malware-distributing domains and the IP addresses they resolve to over time.

new domains were added only a small number of new unique IP addresses appeared.

Next we wanted to track down the country where the domains were hosted. We used the MaxMind[2] database. In Figure 4.9 we can see the breakdown of the percentages of the countries that hosted the top level domains. Honk Kong ranks first hosting 26% of the domains, while the United states follow with 22%. A surprising result is that only 13% of the domains were hosted in China, which is quite lower than what we would expect based on reports [25].

Next we present the results from the experiment regarding domains that distribute malware. Our initial goal was to investigate whether the top level domains of the malware-distributing websites also translate to only a small number of IP addresses.

In Figure 4.8 we present the results from this experiment. We can see that in the case of the URLs that contain malware, the top level domains translated to different IP addresses. Unlike the phishing domains, here each top level domain translated to a different IP address, and only one to three IP addresses overlapped at each moment in time. The IP address that hosted three malware-distributing domains also hosted one of the phishing domains and was located in the United States. None of the other IP addresses hosted

[2]http://www.maxmind.com/app/geolitecity

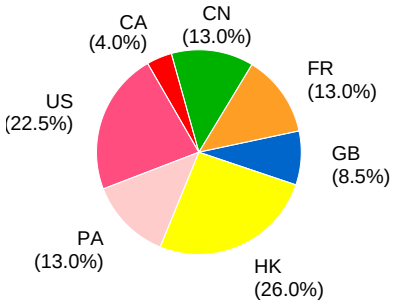

Figure 4.9: Breakdown of countries that host the phishing domains.

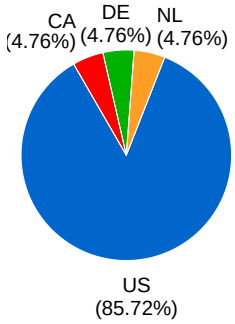

Figure 4.10: Breakdown of countries that host the malware-distributing domains.

both a malware-distributing and a phishing domain. The nslookup operation for all the top level domains returned only one IP address, and only one of the domains belonged to a fast-flux network pointing to a different address each time. Since the IP addresses that host phishing domains are more likely to be blacklisted, this result is not surprising. Another interesting fact is that none of the top level domains is in both of the sets, meaning that none of the domains hosted a phishing site and simultaneously distributed malware.

Similarly to the phishing domains, we wanted to trace the country where the malware-distributing domains were hosted. In Figure 4.10 we can see the breakdown of the percentages of the countries. The United States were responsible for hosting the majority of the domains that distribute malware through IM traffic, reaching almost 86%. The remaining three countries, Canada Germany and the Netherlands, hosted an equal amount of domains. Once again, it is surprising that China did not host any of the domains caught by our infrastructure.

4.5 Attacker profile

In this section we present statistics and observations in an effort to outline the behavior of IM attackers and recognize different strategy patterns.

First of all, in Figure 4.11 we present the number of unique compromised accounts that had sent URLs to our decoy accounts over time. We can see that the plot line follows a sub-linear curve, with almost 100 new accounts

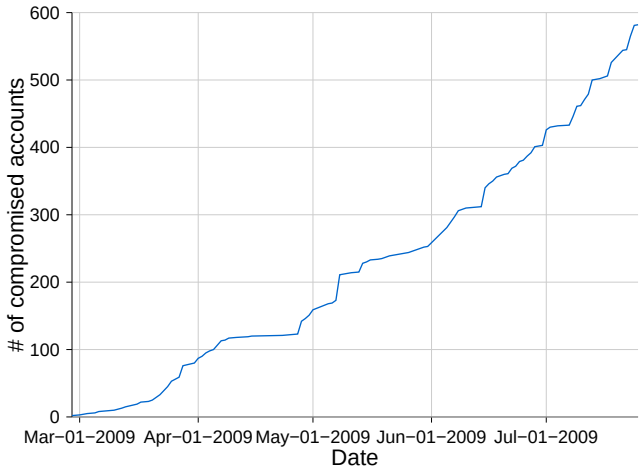

Figure 4.11: Number of compromised accounts that contacted our decoy accounts over time.

contacting us each month. This indicates that over time legitimate users still follow malicious URLs sent by compromised "buddies" and in turn get infected too.

In Figure 4.12 we can see the CDF plot of the number of URLs sent by each compromised account to our infrastructure throughout the duration of the experiment. One should note that approximately 25% of the compromised accounts sent only one URL and 40% up to two URLS. Based on the numbers we can identify one possible strategy that attackers choose to follow.

Even though some of these accounts/hosts may have been dis-infected before sending us another URL, it is improbable that all of them were "cleaned up". Therefore, this might indicate a cautious behavior on behalf of the attackers. With 55% of the compromised accounts sending up to 4 URLs and 75% sending less than 20, it is evident that one strategy that attackers follow is to avoid aggressive spamming behaviors so as not to raise suspicions among the compromised accounts' contacts. Such aggressive behaviors could alert the user and lead to the dis-infection of the account/machine. However, this cautious behavior could also be attributed to technical reasons. If the attack is propagated through a worm that infects the client, then a low rate of worm propagation would be used so as not to trigger antivirus or intrusion detection systems.

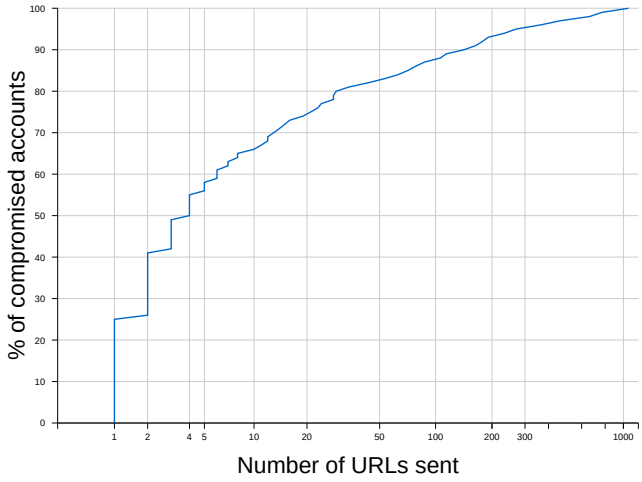

Figure 4.12: CDF of the number of URLs sent by compromised accounts to our decoy accounts.

Furthermore, approximately 12% of the attackers sent at least 100 URLs to our decoy accounts. This aggressive strategy of massively dispatching spam messages, indicates a category of attackers that don't try to remain beneath a certain threshold. This can also be attributed to technical reasons. Specifically, amongst the top ten compromised accounts that sent us the largest number of URLs, we found all the victim accounts whose credentials we had entered in phishing sites. Therefore, the attackers use tools to send messages from these compromised accounts without relying on worms that infect the IM client software. Thus, we can recognize a second more aggressive strategy, where it is not necessary for attackers to adopt a stealthy propagation rate.

Finally, it is interesting to note that attackers send URLs from all the categories, as well as malware, and do not focus on one specific type.

4.6 MyIMhoneypot, a detection service

In this section we present an overview of existing defense measures, and propose a service for the early detection of attacks targeting instant messaging networks. The existing defense mechanisms deployed by instant messaging service providers and other vendors, are insufficient for protecting users from the threats presented in Section 4.1. Anti-virus products that scan files received from instant messaging file-transfers fail to identify all malware used

by IM attackers, as shown by our findings. Anti-virus vendors could provide more up-to-date signatures for IM malware by deploying HoneyBuddy for the early collection of such malware. Pop up messages from IM client software that alert users of phishing, that are triggered by all messages that contain a URL even if it is benign, are ineffective since users tend to ignore warnings that are presented even for well-known benign URLs. We propose that clients should correlate received URLs with blacklists and alert users only when they belong to malicious domains.

We present our client-side mechanism that is orthogonal to existing defense mechanisms; myIMhoneypot, an early detection service that can inform users if their accounts or IM clients have been compromised. IM attacks try to spread through the victim's contact list by sending either URLs or files to the victim's friends. Any user that wants to check if her account is compromised registers with the myIMhoneypot service. Upon registration, the service creates a unique IM honeypot account (for example, a new MSN account that will be used as a decoy account) and informs the user to add that honeypot account to her contact list. As the user will never start a conversation with the honeypot account but an IM attacker will (with great probability), the user can check if something is wrong by visiting the website of the service and checking the conversation logs with her unique honeypot account. If there are entries in the conversation log of her decoy account then there is a strong indication that her IM client or credentials have been compromised.

The reason that a unique IM account must be created per user is twofold. First, if the service has only one or a few honeypot accounts then they can be easily blacklisted (recall that anyone can subscribe to the service, including attackers). The attacker should not be able to distinguish whether a contact is a decoy account or not. The service creates accounts with human-like nicknames. Second, the attacker can try to hack into the service's accounts once she knows the user is a subscriber. Using a unique honeypot per user makes the attacker's life a lot harder. The attacker cannot correlate common friends across accounts and has to try to compromise all the accounts in the user's contact list. Even if she does that, most IM services (at least MSN and AIM) do not keep conversation logs at the server side so she cannot find her spam messages in the logs of decoy accounts.

The attacker could guess the decoy accounts by checking the locally

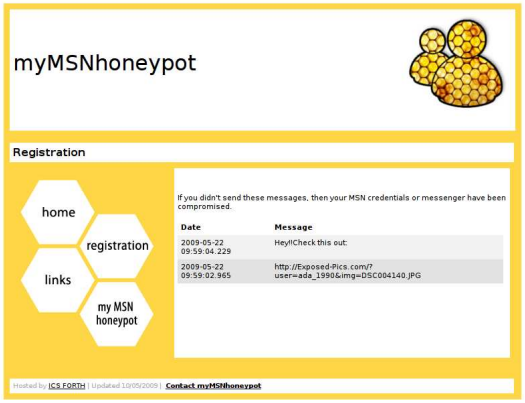

Figure 4.13: A screenshot of the log presented to a user whose IM account has been compromised.

stored conversation logs. Normally, a user will have conversations with all members of her contact list except the honeypot account. Therefore, the attacker could avoid sending messages to accounts for which no conversation logs were found. This attack can be easily circumvented by planting a fake conversation log on the user's side.

The myIMhoneypot service has a limitation. For each registered user, a new IM account must be created in order to be used as a decoy. This process involves the solution of CAPTCHAs [8] which prevents us from making it completely automatic. Although we could claim that MyIMhoneypot is a legal case for laundering CAPTCHAs, we did not implement it for obvious reasons. For the time being, we manually create decoy accounts. However, we propose that this service should be implemented by each IM provider as a means of protection for its users. We implemented a prototype of myIMhoneypot for the MSN platform. We call it myMSNhoneypot and it can be found at `http://mymsnhoneypot.dyndns.org`.

5
Related Work

Collapsar[39] proposes a decentralized architecture composed of a large number of honeypots deployed in different network domains. This approach tries to address the problem of centralized honeypot farms having a limited view of Internet activity. The core idea of Collapsar is to deploy traffic redirectors in multiple network domains and examine the redirected traffic in a centralized farm of honeypots. This approach has the benefit that we can deploy honeypots in many networks without the need of honeypot experts on each network. All the processing and detection logic is done in the centralized honeypot farm, also referred as the Collapsar center. The second part is the front-end of the Collapsar center. It receives encapsulated packets from redirectors, decapsulates them and dispatches them to the honeypots of the Collapsar center. It also takes responses from the honeypots and forwards them to the originating redirectors. Upon receipt, redirectors will inject the responses into their network. In that way, an attacker has the sense that it is communicating with a host in the network of the redirector but in reality she is communicating with the Collapsar center. However, the front-end does more than packet dispatching. Its role is also to assure that traffic from the honeypots will not attack other hosts on the Internet. To prevent such malicious activities, it introduces three assurance models: logging, tarpiting

and correlation. The logging module is embedded in the honeypot's guest OS as well as the log storage in the physical machine's host OS in order to be invisible to the attacker. The tarpiting module throttles outgoing traffic from honeypots by limiting the transmission rate and also scrutinizes outgoing traffic based on known attack signatures. Snort-inline is responsible for performing this task. Snort-inline is a modified version of Snort[49], a very popular intrusion detection system. While Snort is a system that passively monitors traffic, Snort-inline intercepts traffic and prevents malicious traffic from being delivered to the protected network. The correlation module is able to detect network scanning by correlating traffic from honeypots that logically belong to different production networks.

The last part of the architecture is the Collapsar center. The center is a farm of high-interaction honeypots. Honeypots run services inside a virtual machine and have the same network configuration as other hosts in the production network, that is the hosts running the redirectors. The virtual machines used are VMware and UML, with UML being more preferable due to the fact that it is open-source and allows better customization, especially for network virtualization issues.

We can identify two major drawbacks in the approach proposed by Collapsar. The first one is that redirectors need a dedicated machine that communicates with a predefined set of front-ends, imposing administrative overhead for the maintenance of the redirector. Furthermore, it implies a level of trust between the redirectors and the Collapsar center. Once the identity of the front-ends is known, they are susceptible to direct attacks and the redirectors become useless. The second drawback is that traffic redirection adds almost double latency, according to the paper's measurements, which helps attackers in identifying redirectors, e.g. by correlating response times from the redirector and other machines in its production network.

Honeypot farms usually require a large number of physical machines in order to run a few tens of virtual machines. Virtual machines in fact consume a large amount of physical memory and processing power and it is hard to run more than ten on the same physical machine. The Potemkin approach[52] proposes an architecture that overcomes this problem and improves honeypot scalability. The Potemkin architecture is based on two key observations. The first one is that most of a honeypot's processor cycles are wasted idling, as they usually wait for an adversary to connect to them.

The second one is that, even when serving a request, most of a honeypot's memory is also idle.

Based on their design, a virtual machine monitor (VMM) runs on each physical machine. When a packet arrives for a new IP address, the VMM spawns a new virtual machine with that address. This way, we have a virtual machine running only when needed, that is on a per-request basis. However, spawning a new VM for each request is an expensive operation. To reduce this overhead, the *flash cloning* technique is used in the Potemkin architecture. After the the first VM instance boots, a snapshot of this environment is taken. This snapshot is then used to derive subsequent VM images. The process responsible for VM cloning is the cloning manager. It instructs Xen to create a VM based on the reference snapshot. After the VM is created and successfully resumed, the clone manager instructs the guest operating system to change its IP address based on the destination address of the request. During the cloning process the VMM stores packets destined for the VM. After the cloning process is finished, packets are flushed to the VM. Per-request VM cloning solves the problem of wasted processor time spent by a honeypot on waiting for requests. To overcome the problem of large memory consumption, the *delta virtualization* technique is used. The notion behind delta virtualization is that most of the memory pages among VMs are common, for example pages of operating system, and thus can be shared. This technique follows the copy-on-write approach for pages that need to be changed by a VM.

Leurre.com[13] is a distributed honeypot environment that operates a broad network of honeypots covering around 30 countries. Honeypots run a modified version of honeyd and emulate three different operating systems; two from the Windows family (98 and NT server) and Redhat 7.3. Traffic and security logs are retrieved daily and stored into a centralized database. Apart from logs, raw traffic is also analyzed, mainly to derive information about attackers and specifically IP geographical location, DNS names, OS fingerprinting and TCP stream analysis. In the Leurre.com terminology, a single host running the modified honeyd is called a platform. As honeyd emulates three operating systems, each platform needs 3 dark IP addresses to listen to. These IP addresses are consecutive and each emulated OS is assigned to listen to one of them. The reason for listening to consecutive IP addresses is to identify attackers that scan subnets. If all three emulated

OSes are contacted by an attacker, it is a strong indication of scanning. Participants in the Leurre.com project need to deploy a platform and in return they are granted access to the centralized database.

Provos et al. in [35] propose an architecture that combines the scalability of low-interaction systems with the interactivity of high-interaction ones. The architecture consists of three components: low-interaction (or lightweight) honeypots, high-interaction honeypots and a command and control mechanism. The role of low-interaction honeypots is to filter out uninteresting traffic. Connections that have not been established (the 3-way handshake was not completed) or payloads that have been seen in the past are part of the uninteresting traffic. Low-interaction honeypots maintain a cache of payload checksums. If the payload of the first packet (after connection is established) has not been seen in the past, it is considered interesting. According to the measurements of the paper, around 95% of the packets with payloads have been observed in the past. All interesting traffic is handed off to high-interaction honeypots. The hand-off mechanism is implemented by a specialized proxy. Once a packet is marked as interesting, the proxy establishes a connection with the back-end and replays this packet. Next, packets of the "interesting" connection will be forwarded to the back-end by the proxy. The honeyd system is used as the main core of the low-interaction honeypots. The high-interaction honeypots run on VMware and form the back-end of the architecture. The back-end is set up in a way so as not to be able to contact the outside world. Instead of blocking or limiting the outgoing connections, traffic generated by these honeypots is mirrored back to other honeypots. As long as there are uninfected machines, the infection will spread among the honeypots, allowing the capturing of exploits and payload delivery. However, this architecture does not work for malware that downloads its code from an external source, like a web site. To detect whether high-interaction honeypots are infected, their network connections are monitored as well as changes in their filesystem. The virtual machines of infected honeypots are returned to a known good state, through the snapshot mechanism of VMware. The command and control mechanism aggregates traffic statistics from low-interaction honeypots and monitors the load of high-interaction ones. It also analyzes all data from virtual machines to detect abnormal behavior such as worm propagation. The proposed hybrid infrastructure looks similar to the infrastructures

proposed by Collapsar and NoAH. However, this approach focuses more on filtering the interactions before they reach the high-interaction honeypots and additionally the backend architecture is fundamentally different as in this approach mirroring is performed.

Architectures presented so far use honeypots as non-production systems, residing in different network domains than production systems and listening to unused IP address space. Shadow honeypots[33] propose a different approach for detecting attacks that couples honeypots with production systems. The architecture consists of three components: a filtering component, a set of anomaly detectors and shadow honeypots. The filtering component blocks known attacks from reaching the network. Such a component can be either a signature-based detector, like Snort, or a blacklist of known attack sources. The array of anomaly detectors, each one running with different settings in respect to their sensitivity and configuration parameters, is used to classify which traffic is suspicious. The traffic that is characterized as anomalous is forwarded to the shadow honeypots. Their main role is to offload the shadow honeypots as much as possible by forwarding only the traffic that may include an attack.

Shadow honeypots are cloned instances of production servers that are heavily instrumented so as to detect attacks. Two types of shadow honeypots can be identified: loosely-coupled and tightly-coupled. Loosely-coupled honeypots are deployed on the same network of the protected server, running a copy of the protected applications but in a different machine and without sharing state. However, the effectiveness of loosely-coupled shadow honeypots is limited to static attacks that do not require to build state at the application level. Tightly-coupled shadow honeypots run on the same machine as the protected applications and share their state. Shadow honeypots, as stated before, are instrumented versions of protected applications. The instrumentation allows the accurate detection of buffer overflow attacks and is based on the pmalloc() concept, as described in [50]. Pmalloc() is a replacement for malloc(), the standard memory allocation routine, and it works as follows. Before and after an allocated memory block is requested by pmalloc(), read-only memory pages are placed. If an overflow attack is going to take place, it will try to write on the read-only pages and an exception will be thrown. The exception is caught by the pmalloc() routine, indicating the presence of an attack. (note: the concept of pmalloc() was

extended to include statically allocated arrays). The major drawback of this instrumentation approach is the requirement of the application's source code.

When the shadow honeypot detects an attack, its state is rolled back to what it was before the attack and the malicious content is not forwarded to the normal application. It also informs the anomaly detectors about the attack so they can tune their detection models for better performance. If the shadow honeypot does not detect any attack, the request is handled to the normal application. Again, the anomaly detectors are informed that this was not an attack so they can update their models. Shadow honeypots can be also used to protect the client from client-side exploits, such as the buffer overflow in the JPEG handling routine of Internet Explorer. As an example, an instrument copy of Mozilla Firefox can handle web requests. According to the paper, the overhead of the instrumented version is around 20%.

The iSink architecture[55] aims at monitoring large unused IP address space, such as /8 networks. Although this work focuses on measuring packet traffic, we will study its design and properties, which are related to honeypot infrastructures. The design model of iSink is to respond to traffic that goes to unused IP address space. However, as iSink deals with large address spaces, a scalable architecture is needed. The authors considered four systems that can be used as responders: Honeyd, honeynet, LaBrea and ActiveSink. ActiveSink is a framework written by authors using the Click router language[40]. This framework includes several responders, such as ICMP, ARP, Web, SMTP, IRC and NetBIOS responders. Additionally, responders for MyDoom and Beagle backdoors were also implemented. ActiveSink responders are stateless, and still accurate, in order to achieve a high degree of scalability. Even for complex protocols, it is possible to construct a response by looking at the last request packet. Furthermore, there is need for interacting with the attacker up to the point where an attack is detected. For example, if an attack is taking place on the fifth step of a very long conversation, there is no need to emulate further than this step. The four systems were tested along five main criteria: configurability, modularity, flexibility, interactivity and scalability. Honeynet was discarded because of low configurability and medium scalability. LaBrea, on the other hand, has high scalability but very low configurability, modularity and flexibility.

Honeyd presents high configurability and flexibility but medium scalability. ActiveSink, finally, is highly scalable, configurable, modular and flexible, with the only drawback depending on its interactivity with responders (medium interactivity according to authors). The performance of the iSink architecture was evaluated using TCP and UDP packet streams at rates up to 20,000 packets per second. Each packet was a connection attempt. iSink didn't suffer from losses at any rate for both protocols. The system was also deployed in four class B networks and one class A network. The amount of traffic received in these networks was large. The iSink node for the class A network received between 4,000 and 20,000 packets per second.

Xie et al. propose HoneyIM [54], a system that uses decoy accounts in users' contact lists, to detect content sent by IM malware. HoneyIM can be deployed in a enterprise network and alert network administrators of malicious content, provide attack information, and perform network-wide blocking. HoneyIM has a limited view of the IM attack landscape due to its passive architecture and enterprise deployment. To overcome these disadvantages, HoneyBuddy is an active architecture that constantly adds new "buddies" to its decoy accounts, transcending the narrow confines of an enterprise level deployment, and monitors a variety of instant messaging users for signs of contamination. Furthermore, the use of pidgin [20] prevents their system from detecting attacks that exploit vulnerabilities in dominating instant messaging software such as the MSN live messenger [17].

Trivedi et al. address the problem of instant messaging spam (spim) and how to utilize honeypots to extract network and content characteristics of spim[51]. They set up an open SOCKS proxy that only allows outbound connections to IM servers. The analysis of the collected data reveals several characteristics of spim campaigns. An interesting result is that advertised URLs lead to a small number of websites, something that is confirmed by our findings. However, there are several major differences with our work. While they focus on spim campaigns, our honeypot detects all types of instant messaging threats mentioned in section 4.1, and also handles malicious file transfers. Furthermore, they propose a passive architecture that waits for spimmers to connect to their open proxy while our system actively broadens its view by connecting with a diverse and wide-spread set of IM users. Finally, their approach will not work with encrypted instant messaging traffic, such as Skype traffic.

Mannan et al. conduct a survey and provide an overview of threats against instant messaging users and existing security measures[45]. Several scenarios of attacks against IM users are presented, as well as the weaknesses of default security and privacy features provided by IM client software. They conclude that existing public and enterprise IM systems fail to provide sufficient security and protect users from existing IM threats. Hindocha[38] provides an overview of several IM clients and protocols, threats to instant messaging like worms and trojans, and issues regarding IM blocking.

Liu et al. [43] propose an architecture, for detecting and filtering spim, that incorporates widely deployed spam-filtering techniques and new techniques specific to spim based on the analysis of spim characteristics. In follow-up publications [44, 42], the authors focus on instant messaging worms. In [44] worm propagation is modeled and traced through multicast event tree tracing, while in [42] a formal IM worm modeling based on branching process is presented. Williamson et al. [53] apply virus throttling as a mitigation measure against viruses and worms that spread through instant messaging. They explore how several throttle parameters delay propagation without interfering with normal traffic.

Provos et al. [48] follow a different approach than ours for locating URLs that distribute malicious content. They actively scan a large number of URLs to locate malicious actions and focus only on drive-by downloads, while we passively collect URLs from spam messages in IM traffic. A very interesting fact is that their findings show that there is a difference between the domains of the frontend servers that contain URLs that exploit vulnerabilities in users' browsers or plugins, and the domains of the backend servers that distribute the malware. However, our results based on URLs collected by the HoneyBuddy infrastructure do not reveal any such frontend servers. All malware samples were downloaded from the same domain without redirection to a different domain. This highlights a different approach to malware distribution between drive-by downloads and phishing campaigns.

6
Summary

We have explored the use of honeypot technologies for acquiring information regarding cyber-attacks and collecting samples of malicious software. This thesis is separated in two parts.

Initially, we focused on the NoAH project and redesigned its architecture so as to integrate the NoAH and Amun honeypots. The benefits of this integration were twofold. First, the amount and type of information we acquired based on the incoming attack traffic better matched our needs. Second, by removing Argos and (modified) honeyd, and using utilities of the Unix operating system, our infrastructure became more stable and efficient. Upon the integration of Nepenthes, for a period of four months, we monitored all incoming traffic destined to our deployed NoAH sensors and extracted a series of interesting information and statistics. We then replaced Nepenthes with Amun, a more up-to-date medium interaction honeypot very similar in design that provides more fine-grained information regarding the incoming traffic. No further changes to the NoAH architecture were necessary for this integration. We then presented characteristics of the incoming attack traffic.

Next we presented our efforts in identifying a trend in the channels through which attackers deploy phishing campaigns and propagate malware. That lead to the design and implementation of HoneyBuddy, an active hon-

eypot infrastructure designed to detect malicious activities in instant messaging services. HoneyBuddy automatically finds user accounts that belong to a supported IM service and adds them to its contact list. Our system monitors decoy accounts for incoming messages and file transfers, and extracts suspicious executables and URLs. The suspicious data gathered by HoneyBuddy is correlated with existing blacklists, and malware collection center databases. Despite the simplicity of our system, deployment for the MSN service showed that 93% of the identified phishing domains were not listed by popular blacklist mechanisms. Furthermore, 21% of collected malware samples were also not listed by other infrastructures. These findings confirm that existing security measures of instant messaging services are insufficient, and also indicate the effectiveness of our system as a complementary detection infrastructure.

We further inspected the top level domains that host the phishing URLs and found that they translate to a very small number of IP addresses suggesting the existence of a large network of collaborating attackers. On the other hand, domains that distribute malware do not follow the same tactics and translate to a different set of IP addresses. We located domains that belong to fast flux-networks in both cases, however they are more common in the case of the phishing domains, which have a higher probability of being blacklisted. Based on the results from the analysis of the IM attacks we caught, we provided a profile of the attackers and their spamming strategies. An interesting aspect of IM attacks that could not be measured by our infrastructure was how successful an MSN phishing campaign can be. To get an estimation, we deployed our own *benign* campaign and found that almost 12% of the users followed the URL and 4% ran the executable it redirected to.

We also deployed myMSNhoneypot, a prototype implementation of a service that is open to the public and creates dedicated IM honeypots for users. This service provides an early alerting mechanism for users whose IM accounts or clients are compromised. It provides decoy accounts for users that register with the service to add to their contact list. A message from the user to a decoy account is an indication that the user's credentials or IM client are compromised, as the user would never initiate a conversation with the decoy contact.

Bibliography

[1] 15 million new malware types discovered in 2008. http://www.v3.co.uk/vnunet/news/2232004/malware-types-hit-million.

[2] 52 percent of new viruses only last 24 hours. http://www.pandasecurity.com/homeusers/media/press-releases/viewnews?noticia=9804.

[3] Amun honeypot. `http://amunhoney.sourceforge.net/`.

[4] Anubis: Analyzing unknown binaries. `http://anubis.iseclab.org/`.

[5] AQABA Search Engine Demographics. `http://http://www.aqaba-sem.com/search_ed.htm/`.

[6] AutoIt. `http://www.autoitscript.com/autoit3/index.shtml/`.

[7] BuddyFetch. `http://buddyfetch.com/`.

[8] CAPTCHA: Telling Humans and Computers Apart Automatically. `https://captcha.net/`.

[9] Crowbar. `http://simile.mit.edu/wiki/Crowbar`.

[10] Europe surpasses north america in instant messenger users, comscore study reveals. `http://www.comscore.com/press/release.asp?press=800`.

[11] Google safe browsing api. `http://code.google.com/apis/safebrowsing/`.

[12] H1N1 Shortcut Malware. `http://www.f-secure.com/weblog/archives/00001738.html`.

[13] Leurre.com honeypot project. *http://www.leurrecom.org/* .

[14] MessengerFinder, Find people online. `http://messengerfinder.com/`.

[15] Microsoft Security Bulletin MS04-028. `http://www.microsoft.com/technet/security/bulletin/MS04-028.mspx`.

[16] Microsoft Security Bulletin MS06-001. `http://www.microsoft.com/technet/security/bulletin/MS06-001.mspx`.

[17] Msn messenger. `http://messenger.live.com/`.

[18] MSN Polygamy. http://www.softpedia.com/get/Internet/Chat/Instant-Messaging/MSN-Messenger-7-8-Polygamy.shtml/.

[19] Nepenthes. http://nepenthes.mwcollect.org/.

[20] Pidgin, the universal chat client. http://www.pidgin.im/.

[21] Planetlab, an open platform for developing, deploying and accessing planetary-scale services. http://www.planet-lab.org.

[22] Scraping facebook email addresses. http://kudanai.blogspot.com/2008/10/scraping-facebook-email-addresses.html.

[23] Skype Fast Facts, Q4 2008. http://ebayinkblog.com/wp-content/uploads/2009/01/skype-fast-facts-q4-08.pdf.

[24] The state of spam a monthly report august 2007. http://www.symantec.com/avcenter/reference/Symantec_Spam_Report_-_August_2007.pdf.

[25] StopBadware Blog : China Hosts Majority of Badware Sites. http://blog.stopbadware.org/2008/06/24/china-hosts-majority-of-badware-sites.

[26] Surbl. http://www.surbl.org.

[27] Urlblacklist.com. http://www.urlblacklist.com/.

[28] Using honeyclients to Detect New Attacks. http://www.synacklabs.net/honeyclient/Wang-Honeyclient-ToorCon2005.pdf.

[29] Virustotal, online virus and malware scan. http://www.virustotal.com/.

[30] Vmware homepage. *http://www.vmware.com/* .

[31] Vulnerability in PNG Processing Could Allow Remote Code Execution. http://www.microsoft.com/technet/security/bulletin/MS05-009.mspx.

[32] W32.Bropia. http://www.symantec.com/security_response/writeup.jsp?docid=2005-012013-2855-99\&tabid=2.

[33] K. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of the* 14^{th} *Usenix Security Symposium*, Aug. 2005.

[34] S. Antonatos, K. Anagnostakis, and E. Markatos. Honey@home: a new approach to large-scale threat monitoring. In *WORM '07: Proceedings of the 2007 ACM workshop on Recurring malcode*, pages 38–45, New York, NY, USA, 2007. ACM.

[35] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos. A hybrid honeypot architecture for scalable network monitoring. In *CSE-TR-499-04*.

[36] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Visualization. In *Proceedings of the Symposium on Operating Systems Principles (SOSP '03)*, Oct. 2003.

[37] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[38] N. Hindocha. Threats to instant messaging. *Symantec Security Response*, 2003.

[39] X. Jiang and D. Xu. Collapsar: A VM-Based Architecture for Network Attack Detention Center. In *Proceedings of the* 13^{th} *USENIX Security Sumposium*, pages 15–28, August 2004.

[40] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. In *ACM Transactions on Computer Systems 18(3)*, pages 263–297, Aug. 2000.

[41] J. Leskovec and E. Horvitz. Planetary-Scale Views on a Large Instant-Messaging Network. In *Proceedings of WWW 2008*, April 2008.

[42] Z. Liu and D. Lee. Coping with instant messaging worms - statistical modeling and analysis. pages 194–199, June 2007.

[43] Z. Liu, W. Lin, N. Li, and D. Lee. Detecting and filtering instant messaging spam - a global and personalized approach. pages 19–24, Nov. 2005.

[44] Z. Liu, G. Shu, N. Li, and D. Lee. Defending against instant messaging worms. In *In Proceedings of IEEE GLOBECOM 2006*, pages 1–6, 2006.

[45] M. Mannan and P. Van Oorschot. Secure public instant messaging: A survey. In *Proceedings of the 2nd Annual Conference on Privacy, Security and Trust (PST'04)*, pages 69–77.

[46] J. Newsome and D. Dong. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proceedings of the* 12^{th} *ISOC Symposium on Network and Distributed System Security (SNDSS)*, pages 221–237, February 2005.

[47] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an Emulator for Fingerprinting Zero-Day Attacks. In *Proceedings of ACM SIGOPS Eurosys 2006*, April 2006.

[48] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[49] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of USENIX LISA*, November 1999. (software available from *http://www.snort.org/*).

[50] S. Sidiroglou, G. Giovanidis, and A. Keromytis. A Dynamic Mechanism for Recovering from Buffer Overflow Attacks. In *Proceedings of the 8th Information Security Conference (ISC)*, pages 1–15, Sept. 2005.

[51] A. Trivedi, P. Judge, and S. Krasser. Analyzing network and content characteristics of spim using honeypots. In *Proceedings of the 3rd USENIX SRUTI*, 2007.

[52] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *Proceedings of the twentieth ACM symposium on Operating systems principles (SOSP)*, pages 148–162, 2005.

[53] M. Williamson, A. Parry, and A. Byde. Virus throttling for instant messaging. In *Virus Bulletin Conference*, pages 38–4, 2004.

[54] M. Xie, Z. Wu, and H. Wang. HoneyIM: Fast detection and suppression of instant messaging malware in enterprise-like networks. In *Proceedings of the 2007 Annual Computer Security Applications Conference (ACSAC'07)*.

[55] V. Yegneswaran, P. Barford, and D. Plonka. On the design and utility of internet sinks for network abuse monitoring. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.