Honey@home: A New Approach to Large-Scale Threat Monitoring

S. Antonatos, E. P. Markatos
Institute of Computer Science
Foundation for Research and Technology, Hellas
PO Box 1385, Heraklion, Crete, Greece
{antonat,markatos}@ics.forth.gr

K. G. Anagnostakis
Cryptography and Security Department
Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore
kostas@i2r.a-star.edu.sg

ABSTRACT

Honeypots have been shown to be very useful for accurately detecting attacks, including zero-day threats, at a reasonable cost and without false positives. However, there are two pressing problems with existing approaches. The first problem is that timely detection requires deployment of honeypots in a large fraction of the network address space, many organizations cannot afford. The second problem is that attackers are evolving, and it has been shown that it is not difficult for them to identify honeypots and develop blacklists to avoid them when launching an attack.

In response to these problems, we propose a new architecture that enables large-scale deployment at low cost, while making it harder for attackers to maintain accurate blacklists. The Honey@home architecture relies on communities of regular users installing a lightweight honeypot that monitors unused addresses and ports. Because it does not require the static allocation of valuable chunks of network address space, and considering the success of other community-based approaches such as seti@home, our approach is well-suited for creating a large-scale honeypot infrastructure at low cost. Since participation in the system is dynamic as users come and go, it becomes harder for attackers to maintain accurate blacklists.

In this paper we discuss the current design of the Honey@home architecture, a preliminary implementation and describe the design issues that we faced especially with respect to infrastructure robustness, the challenges we have to deal with and the effectiveness of our approach.

Categories and Subject Descriptors

C.2.4 Computer-Communication Networks [**Distributed Systems**]: Distributed applications

General Terms

Security

WORM'07, November 2, 2007, Alexandria, Virginia, USA.

Keywords

Honeypots, Honey@home, Network Security, Packet Forwarding

1. INTRODUCTION

Due to the increasing level of malicious activity seen on today's Internet, organizations are beginning to deploy mechanisms for detecting and responding to new attacks or suspicious activity, called Intrusion Prevention Systems (IPS). Since current IPS's use rule-based intrusion detection systems (IDS) such as Snort [23] to detect attacks, they are limited to protecting, for the most part, against already known attacks. As a result, new detection mechanisms are being developed for use in more powerful reactive-defense systems. The two primary such mechanisms are honeypots [22, 15, 31, 25, 4, 10] and anomaly detection systems (ADS) [29, 28, 13, 19]. In contrast with IDS's, honeypots and ADS's offer the possibility of detecting (and thus responding to) previously unknown attacks, also referred to as *zero-day attacks*. Other approaches, like Dshield [3], try to correlate logs gathered from multiple points and create a summary of most attacked ports and most popular attacking sources. However, such systems are able to detect the source and destination of attacks, not their content.

Honeypots have been shown to be very useful for accurately detecting attacks, including zero-day threats, at a reasonable cost and without false positives, unlike IDS's and ADS's. However, there are two pressing problems with existing approaches. First, the effectiveness of honeypots heavily depends on the unused IP address space they cover. Unused IP address space can be found in almost every organization, institution and public body due to underutilized or even totally empty subnets. However, the deployment of honeypots requires both administrative expertise and dedicated resources that many organizations cannot afford. The second problem is that attackers are evolving, and it has been shown that it is not difficult for them to identify honeypots and develop blacklists to avoid them when launching an attack. Although there are approaches to harden the identification of honeypots, recent work has shown that it is relatively straightforward for attackers to detect the placement of certain types of sensors [12, 24].

In response to these problems, we propose a new architecture that enables large-scale deployment at low cost, while making it harder for attackers to maintain accurate blacklists. The Honey@home architecture relies on communities of regular users installing a lightweight honeypot that mon-

itors unused addresses. Because it does not require the static allocation of valuable chunks of network address space, and considering the success of other community-based approaches such as seti@home, our approach is well-suited for creating a large-scale honeypot infrastructure at low cost. Since participation in the system is dynamic as users come and go, it becomes harder for attackers to maintain accurate blacklists. Users only need to install a lightweight daemon that runs in the background and is responsible for grabbing an unused IP address, forwarding the traffic of that space to a honeypot core and injecting the responses coming from that core in order to respond to the attackers.

In this paper we discuss the current design of the Honey@home architecture, a preliminary implementation, the design issues that we faced especially with respect to infrastructure robustness, and discuss detectability and effectiveness issues.

2. HONEY@HOME DESIGN

Honey@home is designed simple and lightweight, as it mainly targets on typical home users or administrators unfamiliar with honeypot technologies. In a nutshell, Honey@home forwards traffic to unused IP addresses or unused ports to a honeypot farm and sends the answers provided by the honeypots back to the attacker. It is a software package that needs no special configuration to run and can run on both Unix and Windows platforms. It is also non-intrusive as it runs in the background with minimal CPU, memory and network overhead. Our measurements have shown that Honey@home client requires less than 2% CPU utilization and nearly 10MB of main memory for Windows versions. Similar requirements apply for the Linux version as well.

2.1 Design requirements

Honey@home is a software client that is accessible by everyone, including malicious users. With that in mind, we need to fulfill three major requirements:

- The location of honeypots must remain hidden. If honeypots become known, attacker can try to manually compromise them, evade their detection mechanisms or flood them with junk traffic, forcing them to waste their time on useless traffic instead of serving Honey@home clients.
- The identity of Honey@home clients must also remain hidden. Once it is known, the attacker may blacklist them and make them "blind" during the time of a real attack outbreak. Honey@home should be undetectable – or at least very hard to detect – by attackers. We discuss about detectability issues in Section 4.
- Attackers must be prevented from automatically installing Honey@home in the machines they own. In case that the attackers own a botnet, they must be forced to setup Honey@home manually to each one of the bots.

These challenges make Honey@home a more complex architecture than a simple packet forwarder. We address all of the above issues in Sections 3 and 4. Before we proceed to the challenges, we describe the architecture thoroughly at Section 2.2. Apart from the three main requirements, we also need to address the issue that malicious traffic must not be able to infect any part of the architecture, namely Honey@home clients, honeypots and intermediate components.

2.2 Core architecture

Every Honey@home client is responsible for a single unused IP address (unused IP addresses are also referred as *dark*) or the unused port space of the machine it is installed on. All the traffic received by the client is tunneled to the centralized honeypots of Honey@home core through the Tor anonymization network[16]. The Tor network provides all the desired anonymity for both Honey@home users and honeypots. Details about Tor are provided in Section 3. Responses coming from the core are injected by Honey@home to the network so as to reach the originators of the traffic. The architecture of Honey@home is represented at Figure 1. Honey@home clients are connected through an SSL connection to the *SSL server* component. The SSL connection is passing through the TOR anonymization network. The server component handles the client connections and it is responsible for validating users and forwarding their packets to honeypots. Users are validated by supplying a key to the SSL server. They key is obtained after the user registers at the official website of Honey@home. All registration information and user keys are stored in a MySQL database. After the user is validated, her packets are sent to honeypots and responses from honeypots are sent back to the user.

We run both low- and high-interaction honeypots to handle user's packets. *Honeyd*[22] is used as low-interaction honeypot. Honeyd is a very popular and lightweight system with many interesting properties, such as network stack emulation. We use honeyd as a mechanism to filter out uninteresting traffic, such as TCP connections that do not complete the three-way handshake or attacks that can be easily emulated, for example SSH brute-force attacks. All the other traffic is forwarded to high-interaction honeypots. The forwarding is performed by a hand-off mechanism[11] we implemented inside honeyd. Honeyd creates a new connection with the high-interaction honeypot and sends all the application content it receives. The handoff is based on the destination port. For example, if an attacker wants to connect to port 445 or 139, the connection is forwarded to a high-interaction honeypot emulating the Windows XP operating system. As the choice of high-interaction honeypot is static, this means we may loose attacks for services that can run on multiple platforms. Examples of such services are web servers (Linux Apache or Windows IIS?), SMB sharing (Linux SAMBA or Windows sharing?) and many more. However, our choice is currently made based on popularity of applications in terms of users and attack instances. For our prototype system, we emulate a limited number of services and applications and more specific Linux telnet daemon, Microsoft Internet Information Server, MS-SQL server and the default Windows services. We intentionally run unpatched versions of applications and services so we can observe attacks taking place. [1]

We chose Argos emulator[21] for high-interaction honeypot. Argos is using memory-tainting techniques and is able to track both known and unknown exploits. Argos is based on the idea that code coming from the network should never be executed. Once data from the network are treated as ex-

[1]We plan to run fully patched versions in our production system so as to capture only fresh attacks.

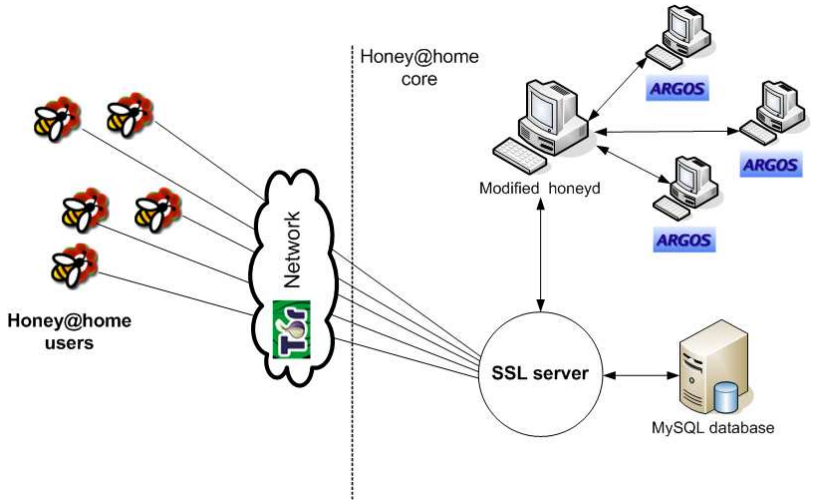

Figure 1: The Honey@home architecture

ecutable code, Argos raises an alert containing all relevant information about the attack and vulnerable application is restarted. As only the vulnerable application is restarted and not the whole virtual machine, the downtime of the vulnerable application is minimal. As Argos detects the attack before its code is executed, we have a core that is hard to infect. The only way to infect an Argos honeypot is to trigger an exploit on the underlying Qemu emulator [5] but this issue is beyond the scope of our work. In contrast with other approaches, like Honeynets[4], that use bandwidth control and extrusion detection to mitigate effects of infection, our core is able to run with minor administration overhead. To improve the scalability of our architecture, we run multiple Argos systems in the core, each one being responsible for specific applications. As Argos has inherently slow performance ,around 30x slowdown of applications, we need multiple instances in order to serve all clients. We also consider using traditional load balancing techniques to share load for the same target applications (like in Web server farms) and offload Argos by emulating known attacks through systems like Scriptgen[20]. Honeyd is not a bottleneck as it can serve thousands of TCP requests per second, around 2000 requests according to [22].

2.3 Unused IP address space monitoring

Each Honey@home client is requesting an IP address from the local DHCP server (optionally it can be set to listen to a static IP address). Most broadband connection routers, like ADSL routers, organizations and institutes use DHCP servers to assign addresses. Every time Honey@home client starts, it requests an address from the local DHCP server. The main advantage of this approach is that user does not need to statically set an IP address, which may be even hard for him to find. Honey@home client can be stopped or started any time without interrupting the normal operation of user's network. Upon the client exit, the clients informs the DHCP server that the IP address is released. The procedure of how a Honey@home obtains an IP address is shown at Figure 2.3. The client first creates a pseudo-interface with a random MAC address and broadcasts a DHCP request (left part of the Figure). When the DHCP response is received, client configures itself to wait for packets destined

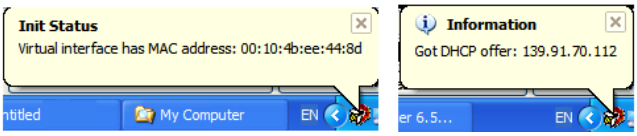

Figure 2: Honey@home client in action: Creating an pseudo-interface (left) and getting an IP address through DHCP server (right)

for the given address, 139.91.70.112 in our example (right part of the Figure).

The Honey@home client can be configured to claim an IP address statically or by using a BPF filter. Static allocation and BPF filter features are mainly targeted for more advanced users. For example, an administrator can setup a single Honey@home client in an unused subnet and set the BPF filter to cover all the addresses of that subnet. In that way, the unused subnet becomes utile in a few steps. As long as the client runs, the subnet monitored by the Honey@home client will contribute to the overall infrastructure.

As a short notice, Honey@home clients also receive legitimate traffic, such as broadcast ping or SMB queries. This traffic has to be white-listed and the decision can be made either locally or at the core. The Honey@home client can be configured not to forward traffic sent to specified ports. Additionally, Honey@home client can be configured to forward traffic sent only to a specific set of ports. By default, all traffic destined to the unused address claimed by Honey@home is sent to core. As Honey@home captures traffic directed to an unused IP address and not the packets destined for the actual IP address of the host running the tool, there are no privacy concerns. All traffic destined to unused addresses is by default suspicious. The only legitimate traffic that is destined for unused addresses, according to our traffic traces, is attempts to connect to peer-to-peer ports. As peer-to-peer programs tend to have some sort of memory, like host caches of Gnutella, it is observed sometimes that some external hosts try to reconnect to an address that was used in the past, participated in a peer-to-peer network, but

is not used anymore and thus claimed by Honey@home. If the user is concerned about privacy issues, she can configure the client not to forward traffic directed to these ports.

Honey@home can work even behind NAT. Hosts behind NAT cannot accept incoming connections from external hosts, only for ports that are explicitly forwarded to them through the router setup. For this case, Honey@home clients can automatically configure the local router to forward specific ports to the physical machine on which client is running using the UPnP protocol[7]. UPnP provides an API to configure the router to forward packets for specific address and ports and is supported by the majority of routers. However, modern routers have UPnP disabled by default for security issues as malware can also use it and make infected machines act like servers. For those users that are not privileged to change the router configuration, the Honey@home client is limited to capture suspicious traffic that is generated by internal infected hosts, for example local scans.

2.4 Unused port monitoring

Regular home users can usually claim one IP address. However, they can also contribute to the honeypot infrastructure by offering their unused port space. Ordinary users usually run a few applications that listen to specific ports. These applications include messengers that bind a port for incoming file transfers and peer-to-peer clients. It is, thus, rare to find typical users that run a web server to their machine or an SMTP server.

Based on this observation and given that already infected hosts keep searching for new victims, it is expected that we may see connections destined to unused ports of the user. Honey@home client can be configured to listen to all unused port space (default) or to a set of ports specified by the user. It examines all incoming traffic (through the *pcap* library) and searches for connection attempts on the unused port space. If such an attempt is found then the traffic going to this port is forwarded to the core. Periodically, Honey@home client scans the host to obtain a list of used ports, similar to the way the netstat tool does, and update the list of unused ones.

The major concern around unused port monitoring is possible deniability. As an example, a user was running a peer-to-peer application for several hours ago and then disconnected from the peer-to-peer network. However, as such systems tend to have a sort of "memory", incoming connections to the ports of his peer-to-peer application may still be arriving. Honey@home client sees that ports previously used by peer-to-peer applications are unused and forwards their traffic despite the fact that this traffic is legitimate. We are currently investigating these issues and we are experimenting on hold-on timers. Hold-on timers remember that a port was used k hours ago and even if it is currently unused, it is not forwarded. Another solution is port sampling, that is forward traffic from only a few unused ports. By default, Honey@home clients do not forward traffic destined to known sensitive ports, like Gnutella and eMule.

3. DEALING WITH CHALLENGES

In this Section, we provide details on how challenges described in 2.1 can be dealt with and specifically how we can hide honeypots (Section 3.1) and prevent automatic installation of Honey@home clients (Section 3.2). Detectability issues of Honey@home clients will be discussed in Section 4.

3.1 Hiding honeypots

Hiding honeypots is essential for the viability of the architecture. The reason is twofold. First, we want to protect the core honeypots from immediate exposure. Although, we cannot protect them from Denial-of-Service attacks (the attacker can easily do DoS on the anonymization network), we make hard for the attacker to manually compromise our honeypots or use techniques that evade the deployed detection mechanisms. For Denial-of-Service attacks, the only measure we could take is not to accept clients that send data above a certain rate. By default, each Honey@home clients sends a few kilobytes per minute and it is easy to detect misbehaving clients. Second, we envision Honey@home as an architecture that will act as a feeding mechanism for other systems as well. Currently, our prototype implementation is based on honeyd and Argos system but our architecture is flexible enough so other systems can plug in as well. A direct exposure of honeypots to users would possibly provide hints for detection mechanisms used in the core.

3.1.1 Anonymous routing

Before we proceed on the approach we use for hiding honeypots, a brief introduction to anonymous routing is provided. The goal of anonymous routing is to protect the privacy of both the sender and the recipient of a message, providing protection for eavesdropping in parallel. Messages are transferred from sender to recipient through an anonymization network and none in the middle of the path can identify the role of each participant (sender, receiver or forwarder).

The anonymization scheme we selected is onion routing, a technique based on a set of dedicated servers that route packets anonymously. Clients simply participate in the network without having to share their resources or help the network route packets. Onion routing uses dedicated software-based routers, called onion routers, that perform the anonymization process. We have also considered peer-to-peer anonymization schemes but they can lead to immediate exposure of Honey@home clients as they become the entities who form the anonymization network. Additionally, to the best of our knowledge, there are no widely deployed peer-to-peer anonymization networks.

The advantage of onion routing is that it is not necessary to trust each cooperating router. The concept behind onion routing is the routing onion. Routing onions are used to create paths through which many messages can be transmitted. In order to create a path, the client at the head of a transmission randomly selects a number of onion routers and generates a message for each one, providing them with symmetric keys for decrypting messages, and instructing them which router will be next in the path. Each of these messages, and the messages intended for subsequent routers, is encrypted with the corresponding router's public key. This provides a layered structure, in which it is necessary to decrypt all outer layers of the onion in order to reach an inner layer. As each router receives the message, it "peels" a layer off of the onion by decrypting with its private key, thus revealing the routing instructions meant for that router, along with the encrypted instructions for all of the routers located farther down the path. Due to this arrangement, the full content of an onion can only be revealed if it is transmitted to every router in the path in the order specified by the layering.

Tor is the most popular and widely used implementation

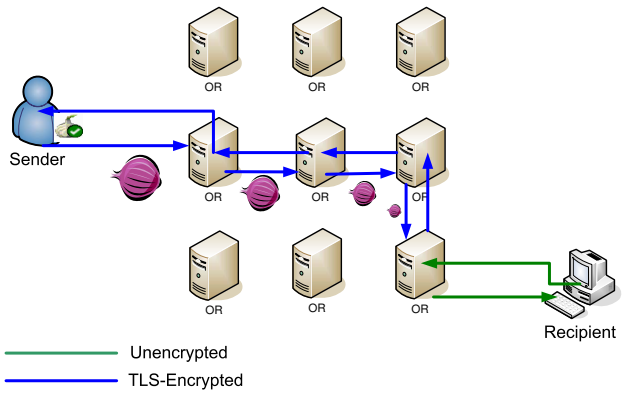

Figure 3: An overview of how Tor works. Client establishes a path of onion routers and sends onions, messages encrypted with public keys of all path's routers. At each router the onion is piled off - decrypted by router's public key- and forwarded to the next router. The last router has fully decrypted content and communicates directly with recipient through a standard TCP/IP connection.

of onion routing and is used for anonymizing any application that uses the TCP protocol. The Tor network has several thousands users and around 400 onion routers are deployed. Figure 3 illustrates how Tor works. At the sender side, user installs a SOCKS proxy that is used by the applications. This specialized proxy connects to Tor network to create an onion routing path. All communication between sender and onion routers and among routers is done using the TLS protocol (blue line). At the server side no deployment is needed. The last router of the path communicates with the recipient using the standard TCP/IP protocol (green line). Sender sends onions which are piled off along the path. The last router receives an onion with non-encrypted content and forwards it to the recipient. The response of the recipient follows the same path backwards.

3.1.2 Using Hidden Services

The way Tor was described earlier assumes that every sender knows the address of the recipient. This is not the case for our system, where address of honeypots must remain hidden. Tor offers a functionality called "hidden services", that permits the recipient to hide its address. Hidden services work as follows. Initially , the recipient gets a descriptor for its hidden service from a centralized service lookup server. This descriptor is a DNS-like name, in the form of "xyz.onion", where the .onion domain can be resolved only inside the TOR network. Afterward, it creates onion paths to several introduction points. An introduction point can be any onion router. It then advertises the descriptors of introduction points and addresses to service lookup repository. The client only needs to know the service descriptor. For example, when a client browses a hidden web server it types to her browser a URL like "http://xyz.onion", where xyz.onion is the service descriptor. When clients lookups for xyz.onion at the directory server, a set of introduction points is returned. The client creates an onion path to a "rendezvous point" and requests one of the introduction points to connect to a server. The introduction point passes the request to the server (remember that introduction point and server are connected through an onion path) along with the address of the rendezvous point. If the server wants to connect to the client, it creates a path to the rendezvous point and then the point mates the two paths. We have implemented Honey@home core as a hidden service. Honey@home clients only know the service descriptor of the core and will connect to it using a "xyz.onion" name. We use two safety features to protect our honeypots from attacks against TOR. First, the entry TOR node from honeypots to the introductory point is a trusted node, maintained by Honey@home developers. This measure protects our infrastructure from Sybil attacks, where attackers users flood the TOR network with malicious servers so they can control the entry and exit nodes of TOR users. Second, we use SSL connection over the TOR network to ensure that even if the trusted router of the path is compromised, it cannot inspect the contents of the communication or understand that is a Honey@home client-Honey@home core communication.

3.2 Preventing automatic installations

In the case that an attacker owns a botnet, she can automatically install Honey@home clients in all bots if no measure for preventing automatic installation is taken. Massive deployment of Honey@home client to a botnet will cause deniability issues at the core. While the core does not suffer from false positive problems (specially crafted traffic that can trigger false alerts), its capacity in processing power is limited. To prevent attackers from massive installations, Honey@home clients are verified to the core by providing a registration key. The registration can be done at the official site of Honey@home. We employ Enhanced CAPTCHAs[9] techniques for the registration process. Additionally, two or more users with the same registration key cannot be connected with the core at the same time.

4. DETECTABILITY OF HONEY@HOME CLIENTS

Honey@home clients are part of a distributed honeypot infrastructure and although they do not present any special functionality themselves, they are considered as detectors from an attacker point of view. Detectability of honeypots is an open issue and has been studied partially in [17, 2, 18] but those techniques focus more on detecting the underlying software system, like honeyd or Sebek. In this work we do not study how to secure the core honeypots from exposure but we try to identify how an attacker can understand if a host is running the Honey@home client.

A honey@home client relies on the Tor network to forward its packets to the core honeypots. The attack scenario is as follows. An attacker wants to identify whether some hosts in a given subnet are running Honey@home. To do so, she performs a TCP scan for specific ports at this subnet. Some hosts will respond in time $t_{respond}$. Honey@home clients will need additional time t_{tunnel} to forward the packet through Tor, wait for the response and inject it back to the attacker. If t_{tunnel} is much larger than $t_{respond}$ then it is an indication that Honey@home is running. Time t_{tunnel} is the response time between Honey@home client and the core honeypots multiplied by a delay factor due to the Tor network. We measured the delay introduced by Tor in terms

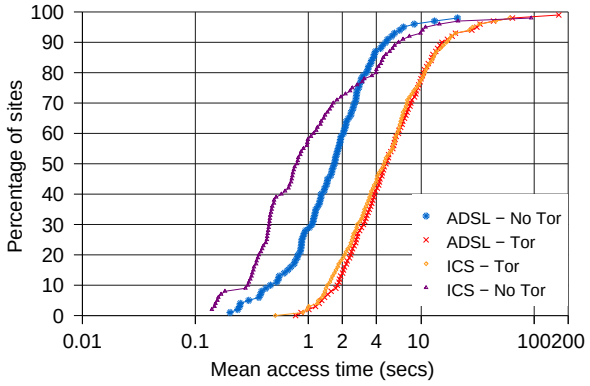

Figure 4: Mean access time for retrieving the index file of top500 sites with and without Tor

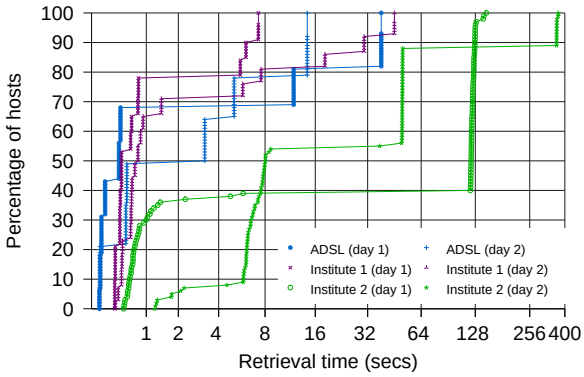

Figure 5: Cumulative distribution function of retrieval time

of time needed to browse several popular web pages (only their main index file). In our experiment, we downloaded the index file of 500 web pages with and without TOR from two different environments: our institute's 1GBps line and an ADSL connection of 1MBps. The results are summarized in Figure 4. For 90% of the cases, we needed up to 4 seconds to retrieve the index file without Tor for both environments and up to 15 seconds in the presence of Tor. This gives us a slow factor of 3 to 4 when using Tor thus making the time t_{tunnel} noticeable.

Honey@home clients are designed to run by normal users. As we cannot reduce the delay of forwarding packets (even the absence of Tor would still yield a significant overhead), we rely on users' behavior to hide Honey@home clients. We performed a scan to port 80 in ten different subnets. Two of them belong to institutes, one in U.S. and one in Singapore, and the rest belong to ADSL users of two major Greek ISPs. For those hosts that had port 80 open, we measured the time to retrieve the index file. We performed the same experiment for the next 2 days after the initial measurement and took again the times. The results are summarized in Figure 5. To avoid making the figure crowded, we plotted the results from the first two days. The results for ADSL subnets are merged and plotted in a single line. We made three interesting observations. First, the hosts that were found to have port 80 open differ slightly from day to day. Some hosts never reappear with port open and some new hosts respond to the port. In fact, only 7% of the hosts were persistently responding to the port all the 3 days. Second, hosts that persistently respond to port 80 present a variation in their response time. Third, a significant percentage of hosts respond in time greater than 5 seconds, with some hosts needing even more than 30 seconds. This delay response allows us to hide honey@home clients as it is unclear for attacker if it is a host responding very slow or a Honey@home node that forwards traffic to core honeypots.

5. SIMULATIONS

As Honey@home is not yet largely deployed, we performed simulation to measure its effectiveness in terms of detection rate and speed. Our simulated scenario involved the spread of a worm, where each infected hosts scans 10 other hosts per second and the vulnerable population is 2% of total IP address space (around 80 million hosts). We assumed that infected hosts do not scan at the loopback (127.0.0.0/8) and NAT subnets (192.168/16, 172.16/12 and 10/8 as defined in [1]) and the initial number of infected hosts was arbitrarily set to 10. Infected hosts were doing random scanning without any topological information included in the scanning process. We consider that the population of Honey@home clients remains unchanged throughout the one simulated minute.

Our initial measurement counted the number of worm instances observed as a function of Honey@home population within the first minute of the worm spread. Results are summarized in Figure 6. As the number of honey@home clients increases, the number of observed attacks is also increased in a linear fashion. For example, with 10,000 clients we can observe 100 instances of the attack, while we need more than 100,000 clients to have 1000 samples. Taking into consideration that attacks may have polymorphic or metamorphic properties, we need multiple observations to characterize the attack or even generate signatures for it.

Our second measurement focused on the detection delay of the first worm instance, that is the time needed to see at least one attack instance in one of the deployed Honey@home clients. The results are displayed at Figure 7. Linearity does not apply for the detection delay. Doubling the number of clients does not necessarily mean that we will see the first instance of the attack in half time. With 100 clients, we need around 52 secs to observe the first instance, while to see it in half time (nearly 25secs) we need around 50k clients. In order to have a detection time of less than 10 seconds we need more than a half million clients. We would like to note here that the term client does not directly refer to a person that will install Honey@home. We use the term client here to denote an unused IP address. As Honey@home is designed to monitor arbitrarily large unused IP address space, the number of actual installations may be significantly less than the number of monitored addresses.

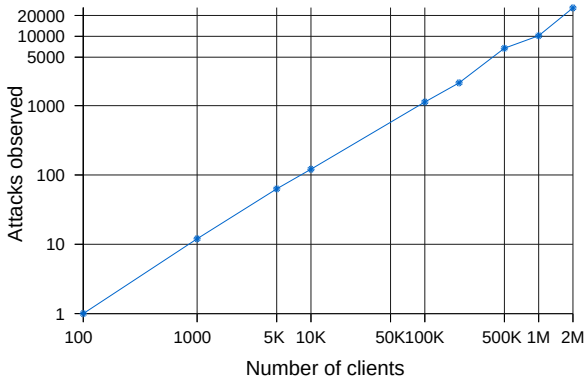

Figure 6: Instances of an attack observed as a function of Honey@home population

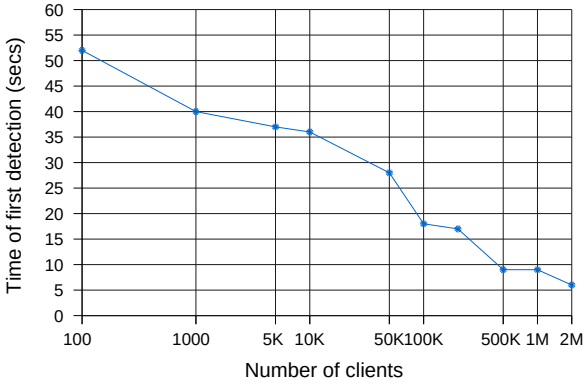

Figure 7: Time needed to detect first instance of an attack as a function of Honey@home population

6. RELATED WORK

In [27], the authors describe the risk to the Internet due to the ability of attackers to quickly gain control of vast numbers of hosts. They argue that controlling a million hosts can have catastrophic results because of the potential to launch distributed denial of service (DDoS) attacks and access any sensitive information that is present on those hosts. Their analysis shows how quickly attackers can compromise hosts using "dumb" worms and how "better" worms can spread even faster. In subsequent work [26], the same authors show how a worm using pre-compiled lists of IP addresses known to be vulnerable can infect one million hosts in half a second.

The Honeynet project [4] is a non-profit organization that is devoted to the research concerning honeypots and their underlying architecture. Central aim of the project is the in-depth analysis of attacks and the capture of malware (e.g. IRCBots). The Honeynet project deploys an architecture that consists of a central gateway, "Honeywall", and the honeypot network. Honeywall separates the network in which the honeypots are deployed from the rest of the network. Additionally, Honeywall performs access control of outbound connections from the honeypots and captures network data. The network behind Honeywall consists of high-interaction honeypots without emulation. Honeypots are also instrumented to trace their system calls through the Sebek tool [6].

Collapsar [30] is a project developed by the Purdue University aiming at the deployment and management of a large number of coordinating high-interaction honeypots across different network domains. The Collapsar architecture is comprised of a *Collapsar center*, a centralized operation center which hosts a network of high-interaction honeypots, and *traffic redirectors*. The redirector allows to virtually deploy honeypots in arbitrary networks and its basic function is to forward all traffic received to the honeypots of the Collapsar center. The redirector is implemented as a virtual machine based on the User-Mode Linux (UML [8]). High-interaction honeypots of the Collapsar center are based on either VMware or UML. Collapsar is somehow similar to our approach but has two major differences. First, redirectors are implemented as virtual machines. This approach requires more resources than our client and thus needs a dedicated machine to run. Second, redirectors are considered as trusted entities. Honey@home is designed to run in any workstation, thus it makes no assumption about trusting its clients.

Vigilante[14] is an infrastructure that aims at worm containment. The architecture of Vigilante is based on the collaboration of end hosts and makes no assumptions that collaborating hosts trust each other. The proposed approach has preferred to move from network-level to host-level in order to eliminate problems like encrypted traffic or lack of information about software vulnerabilities. End hosts act as honeypots; they run instrumented version of software that normally wouldn't run on the host. For example, a host can run an instrumented version of a database, not a common application for a normal host. One of the major contributions of Vigilante is the concept of self-certifying alerts (SCAs). SCAs are distributed among the collaborating of hosts and their novelty is that they can be verified by recipients. This property eliminates the need for trust between the hosts. Three types of SCAs can be identified: arbitrary execution control, arbitrary code execution and arbitrary function argument. Honey@home does not require clients to run instrumented versions of applications, which needs a handful of resources. We focus more on the scalability, supporting thousands of clients requiring minimum resources from them, and on the attack detection. Signatures generated by our architecture are similar to the ones described in [21], that is common subsequences between the memory pages around the overflown buffer and the network trace of the attack. Such signatures are used at the network level to catch attacks. On the other hand, Vigilante detectors generate assembly-level filters that are used at the host level to block the attacks. Finally, our architecture does not include any signature distribution mechanism at its current phase.

Bailey et al. in [11] propose a hybrid honeypot architecture for scalable network monitoring. In their architecture they filter prevalent content by using low-interaction honeypots and use a handoff mechanism to enable interaction between low and high-interaction honeypots. Low-interaction honeypots filter out uninteresting traffic such as unestablished TCP connections or payloads that have been observed many times in the past. Apart from honeypots, their proposed architecture introduces a control component. This

component aggregates traffic statistics from low-interaction honeypots and analyzes all received data for abnormal behavior.

7. CONCLUSIONS

We have explored the design of Honey@home, a lightweight tool that enables users without expertise in honeypot technologies to contribute the fight against cyber-attacks. Honey@home claims unused IP addresses and ports, either dynamically or statically, and forwards all traffic directed to them to a centralized core of honeypots. The core consists of honeyd instances as low-interaction honeypots and Argos systems as high-interaction ones. As Honey@home can be used by anyone, including attackers, three major challenges need to be addressed: hide the identity of users, hide the identity of honeypots and prevent automatic installations. By using Tor, a well-known and deployed anonymization network used by thousands of users, we can hide the identity of both clients and honeypots. To prevent automatic installations, each client needs to be registered through the official website, where CAPTCHA techniques are used similar to many popular large-scale services. Overall, Honey@home enables the creation of a distributed infrastructure with little effort and aims toward a scalable solution that overcomes the problem of classic honeypots, that is monitoring a small portion of unused IP address space.

Acknowledgments

This work was supported in part by the project CyberScope, funded by the Greek General Secretariat for Research and Technology under the contract number PENED 03ED440, and by the FP6 project NoAH, funded by the European Union under the contract number 011923. Spiros Antonatos and Evangelos Markatos are also with University of Crete.

8. REFERENCES

[1] Address allocation for private internets. RFC 1918 *http://www.faqs.org/rfcs/rfc1918.html*.

[2] Advanced Honey Pot Identification and Exploitation. `http://phrack.ru/63/p63-0x09.txt`.

[3] Dshield.org, distributed intrusion detection system. *http://www.dshield.org*.

[4] Honeynet project. *http://www.honeynet.org*.

[5] Qemu data handling multiple command execution and denial of service vulnerabilities. *http://www.frsirt.com/english/advisories/2007/1597*.

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

[7] Upnp forum. *http://www.upnp.org*.

[8] User-mode linux. *http://user-mode-linux.sourceforge.net/*.

[9] E. Athanasopoulos and S. Antonatos. Enhanced captchas: Using animation to tell humans and computers apart. In *Proceedings of the 10th IFIP Open Conference on Communications and Multimedia Security*, October 2006.

[10] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A Distributed Blackhole Monitoring System. In *Proceedings of the* 12^{th} *ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 167–179, February 2005.

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

[12] J. Bethencourt, J. Franklin, and M. Vernon. Mapping Internet Sensors With Probe Response Attacks. In *Proceedings of the* 14^{th} *USENIX Security Symposium*, pages 193–208, August 2005.

[13] M. Bhattacharyya, M. G. Schultz, E. Eskin, S. Hershkop, and S. J. Stolfo. MET: An Experimental System for Malicious Email Tracking. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, pages 1–12, September 2002.

[14] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of internet worms. In *Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, October 2005.

[15] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local Worm Detection Using Honepots. In *Proceedings of the* 7^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 39–58, October 2004.

[16] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the* 13^{th} *Usenix Security Symposium*, Aug. 2004.

[17] M. Dornseif, T. Holz, and C. Klein. Nosebreak - attacking honeynets. In *Proceedings of the 5th IEEE Information Assurance Workshop*, June 2004.

[18] M. Dornseif, T. Holz, and C. Klein. NoSEBrEaK - Attacking Honeynets. In *Proceedings of the 2004 Workshop on Information Assurance and Security*, June 2004.

[19] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the* 10^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 251–261, October 2003.

[20] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In *21st Annual Computer Security Applications Conference*, December 2005.

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

[22] N. Provos. A virtual honeypot framework. In *Proceedings of the 12th USENIX Security Symposium*, pages 1–14, August 2003.

[23] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of USENIX LISA '99*, November 1999. (software available from *http://www.snort.org/*).

[24] Y. Shinoda, K. Ikai, and M. Itoh. Vulnerabilities of Passive Internet Threat Monitors. In *Proceedings of the* 14^{th} *USENIX Security Symposium*, pages 209–224, August 2005.

[25] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, 2003.

[26] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The Top Speed of Flash Worms. In *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*, pages 33–42, October 2004.

[27] S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the USENIX Security Symposium*, pages 149–167, August 2002.

[28] T. Toth and C. Kruegel. Accurate Buffer Overflow Detection via Abstract Payload Execution. In *Proceedings of the* 5^{th} *Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2002.

[29] T. Toth and C. Kruegel. Connection-history Based Anomaly Detection. In *Proceedings of the IEEE Workshop on Information Assurance and Security*, June 2002.

[30] D. X. Xuxian Jiang. Collapsar: A vm-based architecture for network attack detention center. In *Proceedings of the 13th USENIX Security Symposium*, pages 15–28, August 2004.

[31] V. Yegneswaran, P. Barford, and D. Plonka. On the Design and Use of Internet Sinks for Network Abuse Monitoring. In *Proceedings of the* 7^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 146–165, October 2004.