HoneyLab: Large-scale Honeypot Deployment and Resource Sharing

W. Y. Chin*, Evangelos P. Markatos†, Spiros Antonatos†, Sotiris Ioannidis†
*Cryptography and Security Department
Institute for Infocomm Research, Singapore
†Institute of Computer Science,
Foundation for Research and Technology Hellas

***Abstract*—Honeypots are valuable tools for detecting and analyzing malicious activity on the Internet. Successful and time-critical detection of such activity often depends on large-scale deployment. However, commercial organizations usually do not share honeypot data, and large, open honeypot initiatives only provide read-only alert feeds. As a result, while large and resourceful organizations can afford the high cost of this technology, smaller security firms and security researchers are fundamentally constrained.**

We propose and build a shared infrastructure for deploying and monitoring honeypots, called HoneyLab, that is similar in spirit to PlanetLab. With an overlay and distributed structure of address space and computing resources, HoneyLab increases coverage and accelerates innovation among security researchers as well as security industry experts relying on honeypot-based attack detection technology. Unlike current honeypot infrastructures, HoneyLab allows security firms and security researchers to deploy their own honeypot services, instrumentation code, and detection algorithms, dispensing the need for setting up a separate honeypot infrastructure whenever a new attack detection method needs to be deployed or tested.

***Keywords*-honeypot; honeypot infrastructures; security;**

I. Introduction

As the internet threat landscape evolves, modern worms and attacks are spreading on a large scale at alarming rates. By the time security researchers and experts are notified by internet users of such outbreaks, significant damage has already occured. Many recently proposed approaches for detecting large-scale attacks rely on the use of honeypots. Honeypots are systems that run sacrificial services with the intention of luring malicious users or worms into attakcing them and subsequently revealing the attack vector to the honeypot owner. To identify the vector of an incoming attack, honeypots are typically heavily instrumented with an array of detection and reporting capabilities. A key constraint in honeypot-based attack detection is the need for large-scale deployment, as timely detection is tighly dependent on a large number of honeypot sensors spread over different location on the Internet.

Although significant progress has been made in the honeypot research space in the last few years, there are three pressing issues that hinder further development. The first problem is infrastructure fragmentation: security companies like Symantec operate their own proprietary infrastructure, and while there are numerous different volunteer-based honeynet initiatives, each with a different approach, many organizations are reluctant to devote resources to join more than one network. The second problem is flexibility: in current honeypot initiatives, participants have very limited "read-only" access to raw data and alerts. Capabilities such as providing more lively interactions between the honeypots and the researchers are seriously restricted. The last issue is the limited IP address space: once the IP addresses of honeypots are known to an attacker, the address space can be black-listed and this information shared among fellow attackers. Thus, detecting future attacks can prove to be futile.

In this paper we present the architecture and implementation of an overlay and distributed infrastructure called HoneyLab. Our system is designed increasing coverage and accelerating innovation among security researchers and security industry experts relying on honeypot-based attack detection technology. HoneyLab provides the architecture to share address space and computing resources. This way, security researchers can forgo the setting up of distributed honeypot sensors in different geographical locations. Unlike current honeypot infrastructures, HoneyLab allows security firms and security researchers to deploy their own honeypot services, instrumentation code and detection algorithms, despensing the need for setting up yet another network of honeypots whenever a new technology needs to be deployed and tested.

II. Design

HoneyLab provides an infrastructure for deploying and managing security services — sensors and honeypots. In this section we outline the design requirements facing

such a system, we describe our architecture, and we focus on specific design challenges and considerations.

A. Requirements

Scalability. : The platform must support large numbers of deployments, as detection technology is critically dependent on a large number of honeypot sensors being placed at different locations over the Internet.

Flexibility. : To support arbitrary, possibly unforeseen, honeypot configurations, the users must have full control over the honeypot execution environment including operating system and network configurations, or even kernel modifications.

Attack containment. : A compromised honeypot should not adversely affect other honeypots, the HoneyLab infrastructure, or other external parties. For example, if a worm infected a honeypot, HoneyLab computers should not start attacking others computers on the Internet.

Stealth. : It should not be possible for an attacker to map the sensor locations simply by participating as a client of HoneyLab. Furthermore, the infrastructure should not allow the implicit detection of individual honeypots, for instance by imposing a very high latency to their replies when probed by an attacker.

Resource management. : Providing honeypot deployment as a generic service requires mechanisms for controlling resource allocation, accounting for deployment operations and resources consumed, and translating this information into payment requests.

Ease of deployment. : Last but not least, the honeypots should be easy to deploy and set up. Users are not burdened by complicated configurations and procedures in order to set up their own honeypots. Friendly user interface which masks the underlying complex structure is essential.

B. Architecture

The HoneyLab architecture consists of a central node called HoneyLab Central (HLC), and a number of sensor nodes distributed worldwide. The sensor nodes are deployed on servers that run a virtualisation platform (Xen [1] in our implementation), which allows dynamically deploying code to be executed on them inside Virtual Machines.

Our framework supports the following two deployment setups, each of which is suitable for different user requirements.

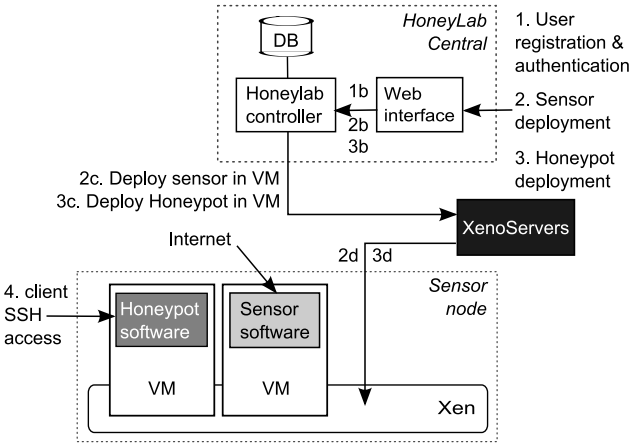

Figure 1. HoneyLab architecture

C. Sensor hosting

In this first setup the sensor software runs on the sensor nodes, and the honeypot software that detects potential attacks runs on nodes provided by the user — termed "client nodes" in Figure 1.

A web application running in HLC manages user registration, and purchase of resources — in the form of IP addresses as well as other parameters including sensor usage time and total allowed sensor traffic — needed for hosting sensors. Users can request a number of IP addresses without knowing the exact IP addresses allocated, though some requirements can be specified — such as physical locations, and OS type. The user receives continuous feedback about the sensor used and associated resource consumption and cost. The process of IP address allocation will be discussed further in the next section.

To collect the traffic and redirect it to the honeypot running at the client side, the user is provided with a VPN server address. Traffic can be delivered to the client host by connecting to the VPN server with standard VPN software (we use OpenVPN in our prototype implementation). It is then redirected to the honeypot software using a virtual interface, created after connecting to the VPN server.

Using the above deployment setting incurs a trade-off. This approach is simple, hides the location of the sensor nodes, and provides full control of the honeypot software to the user. However, it introduces scalability and performance challenges. Firstly, all honeypot traffic has to be forwarded to the client nodes through the central HLC router. Additionally, the observed latency in node to honeypot communication can be high, especially during periods of high load.

For the above reasons it is desireable, in some cases, to avoid sending traffic to the honeypots through HLC — or even avoid sending it outside the sensor node altogether — while preserving the feature of hiding the location of the sensors.

D. Sensor and honeypot hosting

To address the challenge described in the previous section we extended our HoneyLab architecture to cover a second deployment scenario. In this setup both the sensor software and the honeypot software run on the sensor nodes, which avoids forwarding traffic to a remote client node over the network.

This setting requires mechanisms that allow users to deploy honeypot code on the sensor nodes. In our implementation, these are provided by the XenoServer platform [2], which allows deploying and managing Virtual Machines (VMs) across a wide-area server infrastructure from a central location. The VMs are isolated from each other using the Xen Virtual Machine Monitor, ensuring that no adverse cross-VM interference can take place.

XenoServers provide features for creating and deploying VM appliances — frozen and ready-to-go images of VM stacks including an OS and all applications, which we utilise to allow users to rapidly deploy honeypot images on the infrastructure provided. Users are allowed to use one of the globally available appliances, or upload their own. XenoServers also provide facilities for controlling, monitoring, and accounting for resource usage, which is a feature needed to support charging and billing HoneyLab users.

Clients register and deploy their honeypots through the web application running on HoneyLab central. This communicates with the XenoServers facility through the corresponding APIs to get the selected honeypot appliance deployed on one or more hosts. From that point on users can access the VMs and deployed honeypots through VNC, and remote shell. Sensor traffic is available on a special interface as in the previous cases.

Aside from the benefit of avoiding forwarding traffic through HLC, this setup has another advantage: honeypots often need to run for long periods of time and this hosting facility can be handy for researchers with limited/unreliable resources.

In the future we will provide an API for spawning new VMs on demand, enabling the client to implement various types of honeyfarms. In addition to preconfigured VMs, it will be possible to dispatch traffic to a pool of VMs whose size can be adjusted on dynamically. Spawned, identical VMs, as well as VMs of users who limit themselves to availble system images, could use sharing techniques for honeyfarms as in [3] to provide a highly scalable service, which in return would be charged less.

E. Buying Address Space

A HoneyLab client that wishes to deploy sensors start by requesting the allocation of IP addresses. Instead of asking for specific addresses, the client only requests the *kind* and *number* of IP addresses that will be required. The client can also specify a number of criteria including the required port ranges, whether the IP addresses to be provided are statically allocated to HoneyLab or they come from a pool of contributed address space, their location, and whether exclusive or shared/read-only access is needed. The client also selects the images to be used for the sensor and honeypot VMs — which may be images provided by HoneyLab or uploaded by the client.

HoneyLab Central then finds a suitable set of addresses for which sensor nodes with adequate capacity are available in their corresponding networks. Subsequently, it deploys the selected VM images on the chosen sensor nodes, and arranges that each one of them receives traffic from its corresponding IP address space.

Furthermore, the client can instruct HoneyLab to allow access to its sensor traffic to third parties: if another client buys access to the traffic, the initial client will receive credit for future purchases. This feature can benefit researchers with limited resources, and can support easier sharing of information within the research community.

F. Processing of sensor traffic

Port sharing, as promoted by HoneyLab, may lead to incoherent OS fingerprints where ports on the same host appear to remote OS detection tools, such such as `nmap`, to be served by different Operating Systems. Therefore, users of the same IP address must agree on the Operating System profile of the computer that is supposed to exist behind that address. HoneyLab postprocesses outgoing packets, similar to honeyd [4], to enforce consistent OS fingerprints.

Additionally, HoneyLab uses many-to-many network address translation to hide the actual IP address of the sensor. This is both to avoid mapping out the sensors and to facilitate transparent reassignment of the IPs used.

G. Restrictions on honeypot traffic

HoneyLab imposes some restrictions to the outgoing traffic of client's honeypots. Some of these restrictions could possible be disabled under some additional

agreement, but our default policy denies all out-going connections except towards hosts that probed them and a predefined set of hosts such as DNS servers. These kind of restrictions are typical for honeypots, to prevent them from being used for unlawful activities such as spreading a worm. For HoneyLab, such restrictions are further desired for preventing attackers from finding out the sensor IP addresses by renting a honeypot and calling home.

In addition to restrictions imposed by HoneyLab on outgoing traffic, some ISPs will have default firewall polices affecting incomming traffic. Although HoneyLab opts to use IP addresses without additional restrictions, these may be difficult to remove, or worse, may be implicit, seriously affecting sensor measurements. To maintain accurate, up-to-date profiles for each IP address, HoneyLab periodically connects them to a special honeypot that responds on all ports and probes them to determine their network restrictions. These profiles are used when buying address space.

H. Aggregation

HoneyLab provides benefits to its clients through aggregation. First, there is sharing of resources, such as port space. For example, a client interested in port 80 can share an IP address with a client interested in port 135. Second, there is sharing in the form of passive monitoring: a client may chose to allow access to its traces to third parties. In this case, HoneyLab also makes past traffic traces available for download. Third, there is sharing of computing resources through the use of Virtual Machines to host honeypots of different clients on the same physical hardware.

In addition to sharing, HoneyLab offloads honeypot researchers from burdens such as renewing IP addresses in order to keep them secret. Researchers can focus on their research while honeylab searches for sources of fresh Internet address space. IPv4 address space is a scarce resource and many honeypots require vast amounts of address space that can often be multiplexed or shared. With honeylab, a researcher can quickly get temporary access to a vast address space, which would not only be difficult to set up, especially if it is to be distributed, but entirely out of question if needed only for a few experiments.

I. Hiding Honeypot Address Space

We are investigating several mechanisms for keeping HoneyLab's address space secret. First, we rewrite addresses in IP packets so the a honeypot cannot know its real address. Second we allow outbound connections only back to the connecting host. This way a honeypot cannot contact a known server and register its IP address.

However, none of these mechanisms is perfect. It is known to be feasible to pinpoint a honeypot by scanning the entire Internet address space [5]. This limitation of outgoing connections is stronger than desirable and rewriting IP packet headers may cause problems too.

For these reasons, we plan to investigate hiding the honeypots in large pools of dynamic IPs, as described in the next section.

J. Dynamic IP address allocation

To cover the project's huge demands for IP address space, in addition to IPs allocated specifically for the HoneyLab project, we intend to collect Internet address space by leasing unused addresses from DHCP pools and LANs similar to the way that software such as arpd [6] claim unused IP addresses in a LAN by replying to any ananswered ARP requests.

Another possible source of address space is unused port space from end-users running a Honey@Home application. Using dynamic IP addresses like this can give large number of geographically distributed IPs, continuously renewed IPs which are a moving target for attackers to blacklist.

Interestingly, while blacklisting is not desireable from the sensor's perspective, one could speculate that if blacklisted addresses actually deter attackers, many users would be willing to offer their address space in the hope of becoming blacklisted. However, it may be very possible that known honeypot locations become targets of attacks to prevent users from joining.

To deal with dynamically allocated IPs, whose availability varies and they may be reclaimed at any time, we maintain statistics such as the current and mean number of dynamic IPs available to the HoneyLab system. The clients will be charged for the actual number of IPs used in the case that the system could not fullfill the client demand.

The price of resources could be determined by supply and demand: charge more for address space during a worm epidemic, for useful resources such as web traffic, and for exclusive, non-shareable capabilities such as the right to respond to attacker probes.

III. Implementation Details

A. HoneyLab daemon

Each node runs a component called HoneyLab daemon, responsible for controlling other components on the node and communicating with HoneyLab Central. HoneyLab Central also runs a HoneyLab daemon which

coordinates the daemons running on the sensors and receives XML-RPC commands and queries from the Web interface. HoneyLab daemon is written in Python for flexibility.

For each user, HoneyLab daemon maintains the count and constraints of requested IP addresses and configuration of hosted honeypots. Periodically it will try to adjust allocations to match current requests.

B. HoneyRoute

Each sensor node has a routing component called HoneyRoute that maintains a table mapping each IP address to a client's MAC address. HoneyRoute receives commands over a UNIX socket from the HoneyLab daemon to update its internal table. All honeypots on a HoneyLab node are connected to a virtual switch and are identified by their MAC address. HoneyRoute copies packets from the incoming interface to the virtual switch and rewrites the destination MAC address according to the destination IP address and its internal table. The switch can then distribute it to the appropriate honeypot. HoneyRoute is written in C for speed.

HoneyRoute also maintains a list of active IP addresses by observing IP packets sent to its incomming interface's MAC address Entries are expired after a time of innactivity, or when it observes traffic to/from another MAC address. HoneyLab daemon will periodicaly request this list and may adjust allocations accordingly.

C. ARP responder

HoneyLab nodes in our prototype run an ARP responder that claims IP addresses by replying to ARP requests for multiple addresses. Traffic to such claimed IP addresses will be picked up by HoneyRoute and they will be entered in its available IP addresses list. HoneyLab daemon periodically queries that list and will use them to satisfy user requests.

D. VPN server

Each HoneyLab node runs a VPN server (OpenVPN in our implementation). The VPN server has been configured to notify HoneyLab daemon over XML-RPC whenever users connect. HoneyLab daemon will authenticate the users and update HoneyRoute's tables to direct the appropriate traffic to the VPN client.

E. VPN client

After connecting to the VPN server, the /dev/tap0 virtual interface is created, which can be used as a regular network interface by honeypot software running on the client's host — we used honeyd [4], which can be configured to listen for incoming traffic on the virtual interface.

Note that honeyd will use the system's routing table for selecting the interface to send replies, which may not be the same interface it received the probes in the typical case where the machine also uses a real network interface for network connectivity. For honeyd, and other software with the same issue, a source address based routing entry in the OS routing table may be required to direct traffic from the honeypot IP to the virtual interface.

F. Limitations of Prototype

Currently HoneyLab serves IP level traffic. In the future it can be extended to handle layer 2 traffic.

The current implementation does not support sharing of port space among clients.

IV. Related Work

Wu *et al.* [7] detect worms by monitoring probes to unassigned IP addresses ("dark space") or inactive ports and computing statistics on scan traffic, such as the number of source/destination addresses and the volume of the captured traffic. By measuring the increase on the number of source addresses seen in a unit of time, it is possible to infer the existence of a new worm when as little as 4% of the vulnerable machines have been infected.

HoneyStat [8] runs sacrificial services inside a virtual machine, and monitors memory, disk, and network events to detect abnormal behavior. For some classes of attacks (*e.g.,* buffer overflows), this can produce highly accurate alerts with relatively few false positives, and can detect zero-day worms. Although the system only protects against scanning worms, "active honeypot" techniques [9] may be used to make it more difficult for an automated attacker to differentiate between HoneyStats and real servers. FLIPS (Feedback Learning IPS) [10] is a similar hybrid approach that incorporates a supervision framework in the presence of suspicious traffic.

Sidiroglou and Keromytis [11] propose the use of honeypots with instrumented versions of software services to be protected, coupled with an automated patch-generation facility. This allows for quick (*i.e.,* less than 1 minute) fixing of buffer overflow vulnerabilities, even against zero-day worms.

The Internet Motion Sensor [12] is a distributed blackhole monitoring system aimed at measuring, characterizing, and tracking Internet-based threats, including worms. Cooke *et al.* [13] explore the placement of

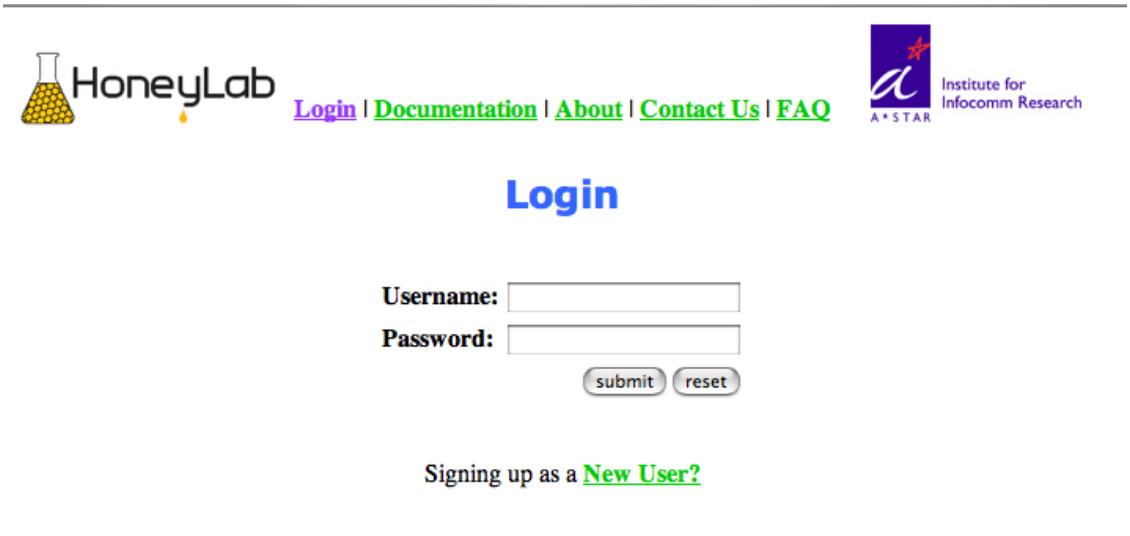

Figure 2. HoneyLab Login Screen

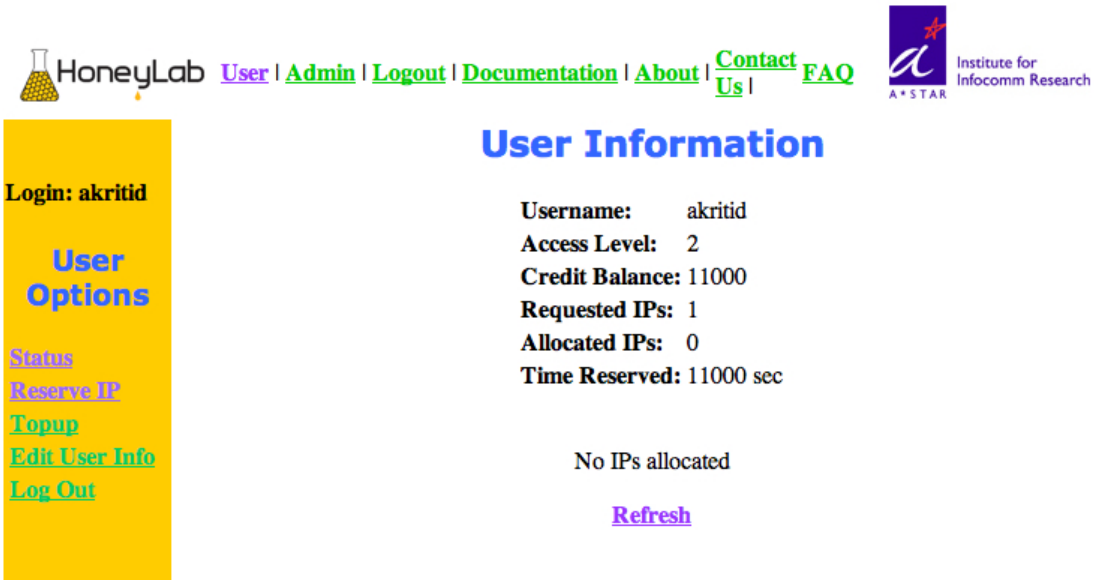

Figure 3. HoneyLab Resource Reservation

honeypots, the correlation of their findings, and the impact on worms and other attacks.

Rajab *et al.* [14] show that a distributed worm monitor can detect non-uniform scanning worms two to four times as fast as a centralized telescope, and that knowledge of the vulnerability density of the population can further improve detection time.

However, recent work has shown that it is relatively straightforward for attackers to detect the placement of certain types of sensors [15], [16], [17]. Furthermore, Richardson *et al.* show that the fidelity of global scanning worm detectors deteriorates rapidly as the level of background "noise" (unsolicited but benign, or at least irrelevant, traffic) increases [18].

Current honeypot initiatives, such as the HoneyNet project [19], the mwcollect alliance and nepenthes [20] malware collection communities, and the DShield distributed intrusion detection system [21], provide very limited "read-only" access to raw data and alerts. Antivirus companies like Symantec operate their own private infrastructure such as Symantec's DeepSight

Threat Management System [22] and are reluctant to join multiple efforts.

HoneyLab allows participants to share resources such Internet address space and computing resources. Honeypot infrastructure software such as honeyd [4], potemkin [3], or frameworks such as Vigilante [23] and Collapsar [24] that require access to Internet address space, can all run under HoneyLab and possibly share resources.

At a high-level, we envision the service model of HoneyLab to be very similar to that of the highly successful PlanetLab initiative [25]. They differ in that HoneyLab is focused on sourcing Internet address space and may have to resort to Collapsar-style cenralization to access address space far beyond where it can host running code.

V. Conclusion

In this paper, We have designed and proposed honeyLab: an overlay and distributed architecture for deploying and monitoring honeypots. By sharing address space and computing resources, HoneyLab allows security researchers and experts to implement honeypot services around the world without the need to set up their own physical networks. Thus, deployment time and cost are significantly reduced. HoneyLab promotes efficient sharing of address space by permitting users to share IP address as well as passive monitoring among fellow researchers. Unlike other honeypot infrastructures, HoneyLab allows security firms and security researchers to deploy their own honeypot services, instrumentation code and detection algorithms, dispensing the need for setting up a separate honeypot infrastructure whenever a new attack detection method needs to be deployed or tested.

We have successfully developed and tested an implementation of HoneyLab. The initial evaluation is encouraging. We will further improve on the design to make it more robust and transparent to the users. We hope it is useful to the security community in detecting modern worms and attacks.

Acknowledgments

This paper is part of the 03ED440 research project, implemented within the framework of the "Reinforcement Programme of Human Research Manpower" (PENED) and co-financed by National and Community Funds (25% from the Greek Ministry of Development-General Secretariat of Research and Technology and 75% from E.U.-European Social Fund). This work was also supported in part by the Marie Curie Actions Reintegration Grants project PASS. Spiros Antonatos and Sotiris Ioannidis are also with University of Crete.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, 2003, pp. 164–177.

[2] "XenoServers," http://www.xenoservers.net.

[3] 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)*, 2005, pp. 148–162.

[4] N. Provos, "A Virtual Honeypot Framework," in *Proceedings of the* 13^{th} *USENIX Security Symposium*, August 2004, pp. 1–14.

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

[6] "Honeyd tools," http://www.honeyd.org/tools.php.

[7] J. Wu, S. Vangala, L. Gao, and K. Kwiat, "An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques," in *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, February 2004, pp. 143–156.

[8] 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)*, October 2004, pp. 39–58.

[9] 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)*, October 2004, pp. 146–165.

[10] M. Locasto, K. Wang, A. Keromytis, and S. Stolfo, "FLIPS: Hybrid Adaptive Intrusion Prevention," in *Proceedings of the* 8^{th} *Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2005.

[11] S. Sidiroglou and A. D. Keromytis, "A Network Worm Vaccine Architecture," in *Proceedings of the IEEE Workshop on Enterprise Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Enterprise Security*, June 2003, pp. 220–225.

[12] 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)*, February 2005, pp. 167–179.

[13] E. Cook, M. Bailey, Z. M. Mao, and D. McPherson, "Toward Understanding Distributed Blackhole Placement," in *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*, October 2004, pp. 54–64.

[14] M. A. Rajab, F. Monrose, and A. Terzis, "On the Effectiveness of Distributed Worm Monitoring," in *Proceedings of the* 14^{th} *USENIX Security Symposium*, August 2005, pp. 225–237.

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

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

[17] M. A. Rajab, F. Monrose, and A. Terzis, "Fast and Evasive Attacks: Highlighting the Challenges Ahead," in *Proceedings of the* 9^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2006, pp. 206–225.

[18] D. W. Richardson, S. D. Gribble, and E. D. Lazowaska, "The Limits of Global Scanning Worm Detectors in the Presence of Background Noise," in *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*, November 2005, pp. 60–70.

[19] "The Honeynet Project," 2003, http://www.honeynet.org/.

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

[21] "DShield.org, Distributed Intrusion Detection System," http://www.dshield.org.

[22] Symantec, "DeepSight threat management system home page," 2004.

[23] 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 Systems Principles (SOSP)*, 2005, pp. 133–147.

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

[25] L. Peterson, D. Culler, T. Anderson, and T. Roscoe, "A blueprint for introducing disruptive technology into the internet," in *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, USA, Oct. 2002.