The Past, Present, and Future of Honeynets Implemented in Virtual Networks

Michael Fernez

Abstract: Since the founding of the Honeynet Project in 1999, dozens of projects have been made public to help promote and implement honeypot servers in enterprise networks. These honeypots have been designed to serve a variety of niches from low-interaction port scan detectors to full-scale web servers with a high degree of customization. The concept of a honeynet is to combine these variously-purposed machines into an entire network of servers which divert, classify, and defend against web attacks. There have been many attempts at implementing such a system on a virtualized environment as a means to conserve system resources and contain the entire network on a single host machine. These attempts are documented and analyzed and judged based on strengths and weakness of their architecture.

I. Introduction

Lance Spitzner, founder of the honeynet project, defined a honeypot as "an information system resource whose value lies in unauthorized or illicit use of that resource. [1,2]" After founding the honeynet project in 1999, this notion bloomed into a complex network of different honeypots targeting different types of attackers. The practice of combing these honeypots became known as a honeynet [1] "a form of a high-interaction honeypot ... a network that contains one or more honeypots." However, there are siginificant costs and issues with such systems and there is significant debate on the return of investment of deploying a system of honeypots [2,3,4,5] The benefits of utilizing virtualization to implement a network of honeypot servers include lower cost, low investment of system resources, less network complexity, and potentially higher security and isolation from the rest of the network. Therefore, it is worth exploring the various ways researchers have tackled implementing such systems and analyzing the value of their approach.

This paper is organized in the following way: Section II provides detailed definitions of honeypots, honeynets, and virtual honeynets and outlines the evolution of honeynet architectures. Section III discusses the challenges of implementing traditional honeynet architectures and the potential benefits and drawbacks of virtualization. Section IV briefly describes recent examples of proposed and implemented virtual honeynets and analyzes these approaches based on the criteria established in Section III. Section V summarizes this analysis and poses questions for future work.

II. Background

A. Honeypots

Generally speaking, a honeypot is a network security device whose value lies in being probed and misused by an attacker so that it can collect, classify, and defend a network against such attacks [2]. Honeypot systems are generally classified by both the level of interaction they provide for an attacker and the way they are deployed and used on a network.

There is a spectrum of functionality among honeypots available on the web, but they are generally categorized as a high- or low-interaction system. In a low-interaction honeypot, a server is typically setup to emulate a few services one might expect from a webpage, computer, or other network device. Sandboxing provides an isolated environment for the attacker to try out his attacks and provide realistic responses so the attacker believes he is interacting with a real system. Examples include

Glastopf, honeyd, and nepenthes [2, 6, 11]. In a high-interaction honeypot, a server is set up to mimic an entire operating system for the attacker to interact with, while logging their every move in the background. Examples of high-interaction honeypots include Honeywall, HIHAT, and Sebek [2, 6 ,11]

Honeypots are further classified by the information they collect and use in the field. Production honeypots aim to provide an extra level of security for an organization, in the hopes that the attacker wastes time attacking a fake server long enough for security to notice and respond. They are essentially more sophisticated versions of intrusion detection systems. Research honeypots are designed by security researchers hoping to capture a lot of data on attacks from as many attackers as possible to learn the tactics of attackers and therefore better understand how to defend against them.

B. Honeynets

A honeynet is a highly controlled network of honeypots designed to increase the depth of interactions with attackers [1, 2, 7]. Traditional honeynet architectures have been defined by the following functionalities: data control, data capture, and data collection. When a honeypot receives a malicious request it will implement *control* to ensure the request does not compromise the system, *capture* the request by logging it, and *collecting* the attack in a database so it can be analyzed [1, 8].

Generation I honeynets introduced the entire concept. The approach was simply to set up a firewall with an IDS as the gateway, which led to several honeypots behind it. This required two interfaces on the Honeywall, one to the internal network and one to the external Internet. However, this technique was easily detected by attackers since the gateway acted as a network layer device [6].

Generation II and III Honeynets were devloped in 2001 and 2004 respectively to address the flaws of Generation I architectures [6]. The major improvement was to the Honeywall Gateway. Gen-II introduced a third interface to the mix. Two provided the bridge between the internal honeynet and external network and a third solely for management and configuration so the operator could configure the honeynet via the link layer. Gen-III simply added a Sebek server to the Honeywall. It is a collection of three data capture tools which increased the control Honeywall had to collect and analyze attacks.

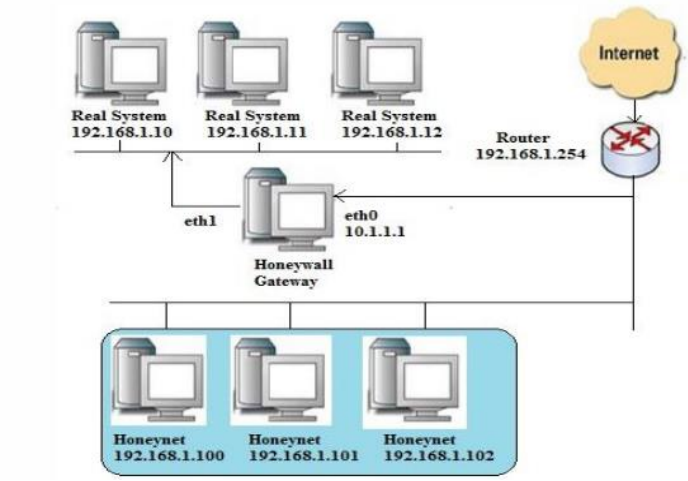

Fig. 1 A basic honeynet design [7]

C. Virtual Honeynet

In principle, any physical system can be run as a Virtual Machine using tools such as Virtual Box, VMWare, or KVM. A honeynet implemented on a series of virtual machines is therefore called a

virtual honeynet [1]. There are generally two flavors of virtual honeynets: self-contained and hybrid systems.

A self-contained honeynet architecture (Fig. 2) is a network of honeypots and Honeywall all contained within one physical system running many virtual machines. The benefit of this approach is portability and cheapness, since only one device is needed. One project on the Honeynet Project's page is Honeystick, which implements a Honeywall and network of honeypots all on a single USB device [11]. The disadvantage of this approach however, is that it creates a single point of failure-- the entire honeynet can be lost if compromised. The shared hardware of the honeynet also increases this risk and also may limit the software an operator can deploy. Further, the costs of creating a single machine powerful enough to run all of the honeynet components efficiently may offset the costs saved.

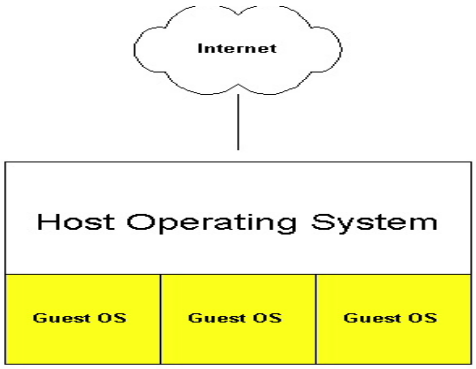

Fig 2. A Self-contained Virtual Honeynet [1]

A hybrid honeynet architecture utilizes a physical system as the gateway to the honeynet whose core components run as guest operating systems on a separate machine (Fig. 3). The benefit of this approach is security and flexibility. Isolating the honeynet from the firewall keeps an attacker from traversing beyond the honeypots on the virtual network or tampering with the Honeywall controller itself. It also enables an operator to diversify the hardware and software he can deploy and even add new machines to the network. The disadvantage to having multiple machines, however, is the cost of power, space, and limited portability.

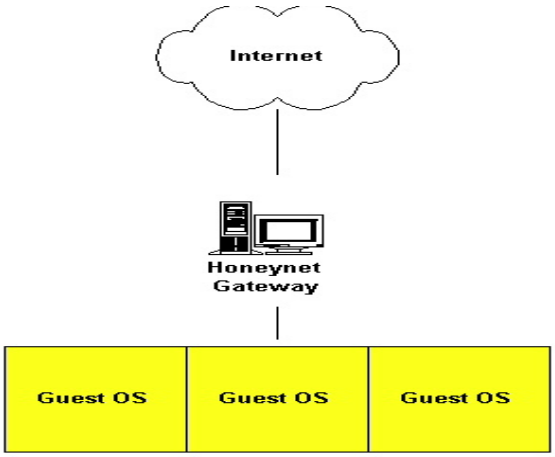

Fig 3. A Hybrid Virtual Honeynet [1]

III. Challenges for Honeynet Developers

A. Hardware and Cost limitations

Employing an entire network of honeypots on physical machines can be cumbersome and expensive as they require dedicated machines to act as honeypot servers. Not to mention they also add a layer of complexity to any network which can potentially open the door to more vulnerabilities.

Virtual machines can potentially provide a solution to both of these problems by containing the entire honeynet on a single machine using low-cost or free virtualization options. However, it is important to note the limitations of virtual hardware. First is the CPU usage by the VM, which tends to be high especially for more sophisticated honeypot systems. A honeynet developer should pay special attention to this while setting up the virtual machine, as an overloaded CPU can introduce vulnerabilities. The second hardware concern is virtual memory since honeypots tend to collect a large number of logs. Therefore, any deployment must carefully consider both the virtual memory used and the amount of logs stored on the virtual honeypot [4].

B. Security

Though honeypots can undoubtedly be an important security asset, they can also potentially introduce vulnerabilities due to the complexity they add to the network.

An analysis by F. Daryabar et al. [4] suggests two possible security flaws that might be overlooked when migrating from a physical to a virtual environment. First, is the lack of ability to monitor communications between VMs on the network. Second is the problem of containment, since an infection on one virtual machine will tend to spread to the rest and potentially beyond. The same group has proposed using a Virtual Private Network (VPN) from the honeynet to the firewall, which can serve as both a means to monitor communication between VMs while providing a layer of encryption.

Another issue is the passive fingerprinting of honeypots. Since honeypot software emulates vulnerabilities by providing believable responses to a malicious request, sometimes these responses, especially if they are hard-coded into the system, can alert an attacker to the presence of the honeypot. For example, a Kippo honeypot emulates the response to the “uname -a” command as “Wed Nov4 20:45:37 UTC 2009” by default, which is a dead giveaway for a knowledgeable attacker that Kippo is running [Table 1]. Similar attacks have been described to detect the presence of Honeywall and Sebek, the most common tools for monitoring a honeynet [10]. This can be addressed by designing honeypots with a flexible sandbox so that the honeynet operator can create custom responses more likely to fool an attacker.

Request		Response	
type	payload	type	payload
exact match	uname -a	exact match	Wed Nov 4 20:45:37 UTC 2009
pattern	.{7,}\n	exact match	bad packet length
exact match	vi	exact match	E558: Terminal entry not found in terminfo
exact match	ifconfig	exact match	HWaddr 00:4c:a8:ab:32:f4

Table 1. Commands and hard-coded responses that can identify the Kippo honeypot [9]

IV. Implementations and Analysis

A. VMWare-based Server with Honeywall Inside

A 2011 analysis of the core functions of a virtual honeynet by L. Tian-Hua, Y. Xiu-Shuang and M. Shi-Wei [8] implemented a hybrid virtual honeynet utilizing two physical machines: a VMWare server running the Honeywall and the internal network of honeypots plus an external management system.

The main benefit of this approach is security. Since the Honeywall is isolated from an additional machine which controls it, an attacker is inhibited from reaching any part of the network outside of the Honeywall, even if both the Honeywall and honeynet are identified or compromised. Another unique benefit to this team's approach is the diversity of operating systems including two Windows and one Ubtuntu with the potential to add more. However, this comes at a cost of complexity, space and resources. Not only does one have to power the two physical machines, but also pay for the proprietary software used, i.e. VMWare and Windows.

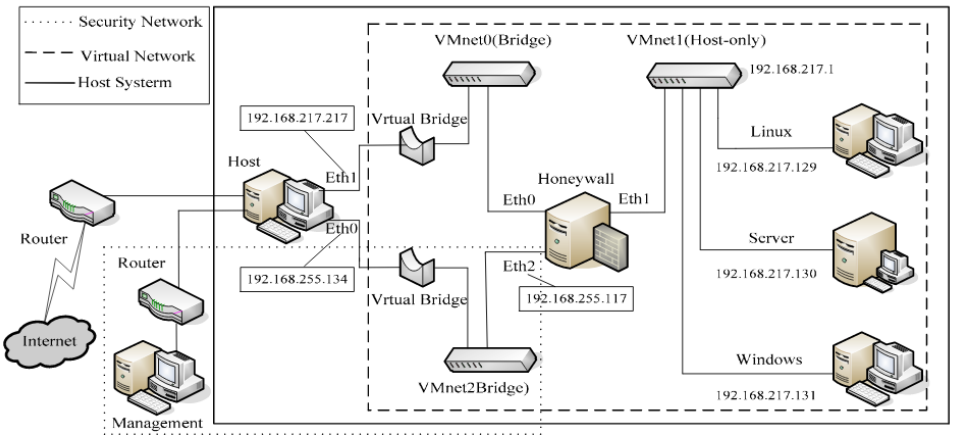

Fig 4. Architecture of the virtual honeynet implemented in [8]

B. Virtual Box-based Server using Host-Machine as Honeywall

In 2015, R. Gautam, S. Kumar and J. Bhattacharya proposed a self-contained virtual honeynet architecture with a focus on open-source software to reduce costs. Their implementation uses a single physical machine running the Honeywall Gateway in parallel with a series of virtual honeypots managed by Virtual Box.

The benefit of this setup is that it eliminates the need for an external controlling machine, allowing the operator to run the honeynet on a single device. In addition, the configuration of the host machine allows the operator to program backups and the shut down cycle of honeypots on the virtual network, mimicking real-world systems. Unlike other implementations, logs and data of incoming attacks do not need to be transferred between virtual machines but can all reside on the host.

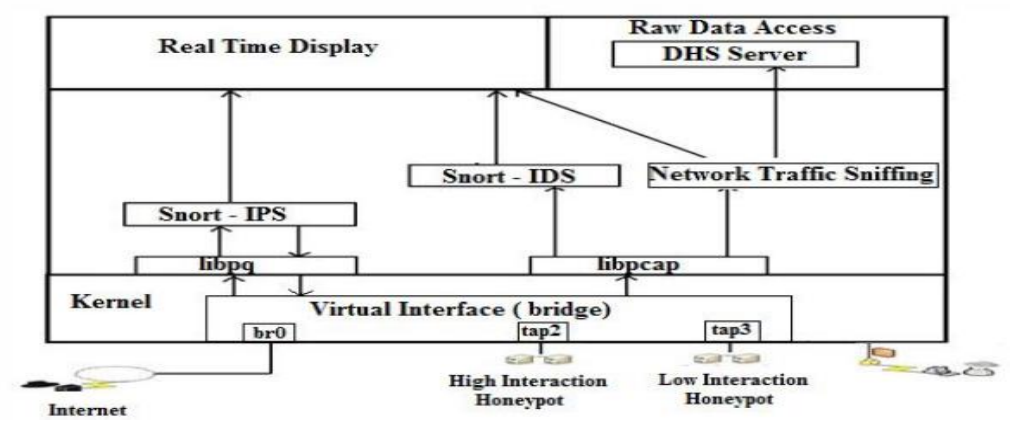

Fig 5. Architecture of the virtual honeynet implemented in [7]

C. A Hybrid System Utilizing an SDN Controller

A more recent approach [9] demonstrates the utility of SDN (Software Defined Networking) for controlling a honeynet. Software Defined Networking is an approach to network architecture that is built on a collection of software modules to control, manage, program, and monitor a network. An SDN controller uses such software to simplify and centralize control of a network. Kyung et al introduced the Honeyproxy controller with which they implemented a hybrid virtual honeynet. The network is composed of two physical machines, one which runs the controller while another hosts a virtual network of high- and low-interaction honeypots via KVM virtualization infrastructure.

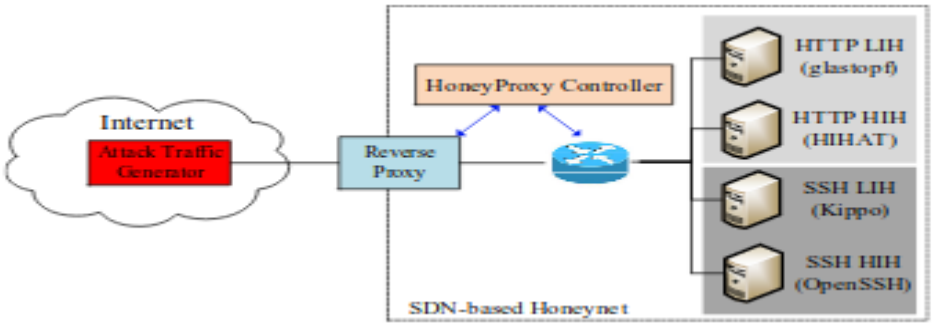

Fig 6. Architecture of the virtual honeynet implemented in [9]

The team behind Honeyproxy identified three major issues with current state of honeynet architectures, still largely relying on the 14-year old Gen III model: vulnerability to fingerprinting, internal propagation of malware, and lack of interopability of honeypots on the network. The SDN-based approach solves these problems by providing a control interface and rules which allow dynamic responses to fingerprinting attacks, easy monitoring of communication between honeypots, and efficient control of network services.

The traditional honeynet architecture treats honeypots on the virtual network as individual isolated systems. For example, if a Honeywall detects an SSH attack, it will redirect the attacker to an SSH honeypot and likewise an SQL attack to a web-server honeypot. It is not only easier to detect individual honeypots on the networks this way, but is also an inefficient use of the network since many honeypots on a honeynet run redundant services. For example, one might employ both a high- and low-interaction SSH honeypot, but which is the best honeypot for the job? In an SDN paradigm, the

separate honeypots can be organized into a cloud of different services and emulators that work together (Fig 7.). It does not completely eliminate the possibility of fingerprinting a particular honeypot on the network, nor the possibility of an advanced piece of malware to bypass the IDS, but the SDN architecture does obfuscate these connections and makes managing components on the network a much simpler process.

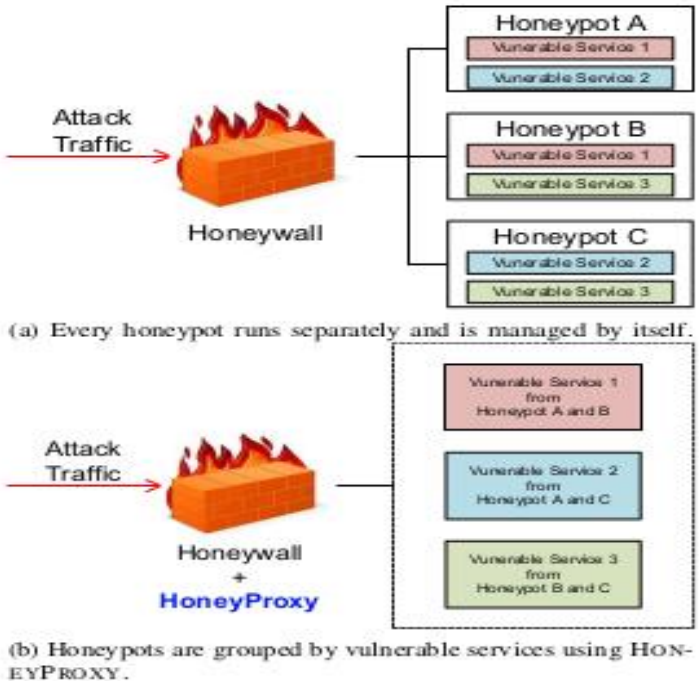

Fig 7. A comparison of traditional Honeywall to Honeyproxy

V. Conclusion

Beyond just virtual honeynets, honeynet architecture as a whole deserves much more consideration and attention then it currently gets. Though the honeynets presented here offer very promising improvements on honeynet architecture, we have not yet evolved beyond the Gen-III model of 2004.

Individual Honeypot projects seem to have stagnated also. Much of the traditional honeypot software such as Honeywall, Kippo, and Glastopf are outdated and often vulnerable to fingerprinting. The longer these tools go untested the more chance there is for an attacker to learn to identify and circumvent them. Stealth should be a core asset of any next-generation honeynet as it is practically synonymous with the security of the network.

The ongoing evolution of SDN-controllers is a particularly promising route at solving such challenges and reducing the overall complexity of deploying and maintaining a Virtual Honeynet. Understanding the traditional trade-off of a hybrid approach versus a self-contained one remains important, however, as it is simply a restatement of the old problem of security versus convenience.

VII. References

1. Spitzner, L. (n.d.). Know Your Enemy: Defining Virtual Honeynets. Retrieved November 24, 2018, from http://old.honeynet.org/papers/virtual/
2. W. Fan, Z. Du and D. Fernández, "Taxonomy of honeynet solutions," *2015 SAI Intelligent Systems Conference (IntelliSys)*, London, 2015, pp. 1002-1009. doi: 10.1109/IntelliSys.2015.7361266,URL: http://ieeexplore.ieee.org.ez.lib.jjay.cuny.edu/stamp/stamp.jsp?tp=&arnumber=7361266&isnumber=7361074
3. Z. Heng-ru and G. Jie, "Research and Design of Network Attack and Defense Platform Based on Virtual Honeynet," *2010 International Conference on Computational and Information Sciences*, Chengdu, 2010, pp. 507-510. doi: 10.1109/ICCIS.2010.130, URL: http://ieeexplore.ieee.org.ez.lib.jjay.cuny.edu/stamp/stamp.jsp?tp=&arnumber=5709135&isnumber=5708911
4. F. Daryabar, A. Dehghantanha, F. Norouzi and F. Mahmoodi, "Analysis of virtual honeynet and VLAN-based virtual networks," *2011 International Symposium on Humanities, Science and Engineering Research*, Kuala Lumpur, 2011, pp. 73-77. doi: 10.1109/SHUSER.2011.6008503, URL: http://ieeexplore.ieee.org.ez.lib.jjay.cuny.edu/stamp/stamp.jsp?tp=&arnumber=6008503&isnumber=6008470
5. Dornseif, M., & May, S. (2004). Modelling the costs and benefits of Honeynets. Third Annual Workshop on Economics and Information Security. Retrieved November 25, 2018, from https://arxiv.org/pdf/cs/0406057.pdf
6. Abbasi, F. H., & Harris, R. J. (2009, November). Experiences with a generation iii virtual honeynet. In *Telecommunication Networks and Applications Conference (ATNAC), 2009 Australasian* (pp. 1-6). IEEE. URL: https://www.researchgate.net/profile/Richard_Harris9/publication/224138278_Experiences_with_a_Generation_III_virtual_Honeynet/links/0f31753b488089fc36000000.pdf
7. R. Gautam, S. Kumar and J. Bhattacharya, "Optimized virtual honeynet with implementation of host machine as honeywall," *2015 Annual IEEE India Conference (INDICON)*, New Delhi, 2015, pp. 1-6. doi: 10.1109/INDICON.2015.7443696, URL: http://ieeexplore.ieee.org.ez.lib.jjay.cuny.edu/stamp/stamp.jsp?tp=&arnumber=7443696&isnumber=7443105
8. L. Tian-Hua, Y. Xiu-Shuang and M. Shi-Wei, "Core Functions Analysis and Example Deployment of Virtual Honeynet," *2011 First International Conference on Robot, Vision and Signal Processing*, Kaohsiung, 2011, pp. 212-215. doi: 10.1109/RVSP.2011.16, URL: http://ieeexplore.ieee.org.ez.lib.jjay.cuny.edu/stamp/stamp.jsp?tp=&arnumber=6114940&isnumber=6114587
9. Kyung, Sukwha, et al. "HoneyProxy: Design and implementation of next-generation honeynet via SDN." *Communications and Network Security (CNS), 2017 IEEE Conference on*. IEEE, 2017. URL: https://adamdoupe.com/publications/honeyproxy-cns2017.pdf
10. M. Dornseif, T. Holz and C. N. Klein, "NoSEBrEaK - attacking honeynets," *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, West Point, NY, 2004, pp. 123-129.doi: 10.1109/IAW.2004.1437807 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1437807&isnumber=30958
11. Honeynet Project - Projects. (n.d.). Retrieved November 25, 2018, from https://www.honeynet.org/project