Adaptive and Flexible Virtual Honeynet

Wenjun Fan[1(✉)], David Fernández[1], and Zhihui Du[2]

[1] Departamento de Ingeniería de Sistemas Telemáticos, ETSI Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain
efan@dit.upm.es

[2] Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract. Honeypots have been largely employed to help securing computer systems and capture malicious activities. At present, virtual honeynets -network scenarios made of virtual honeypots- are frequently used to investigate the adversary's behaviour. The static deploying scheme used traditionally, in which the configuration of the honeynet is determined by security experts beforehand, lacks the capability of dynamically adapting its configuration after deployment. In this paper, a new adaptive and flexible virtual honeynet management system is proposed that dynamically creates, configures and deploys both low-interaction and high-interaction honeypots, emulating multiple operating systems. The results and measurements of the experiments carried out illustrate that new virtual honeynet system is more capable than previous virtual honeynet architectures.

Keywords: Dynamic honeynet · Virtual honeynet · Honeynet configuration

1 Introduction

A honeypot is an information system resource whose value lies in its unauthorized or illicit use [1]. Nowadays, honeypots are widely used in a variety of different situations to address system and network threats, as preventing the production systems from being attacked and other security goals.

First of all, honeypots can be used to detect unknown attacks. For instance, they can help solving the false negative problem of network intrusion detection systems (NIDS), when they cannot identify the signatures of the new and unknown attacks. All of the traffic sent to a honeypot is almost certainly unauthorized traffic, meaning no false positives. Honeypots can identify the suspicious activities by monitoring those network packets that could have been previously marked as non-suspicious by a normal NIDS. Thus, honeypots are able to assist intrusion detection system to reduce the number of false negatives and help in detecting potential attacks.

Secondly, honeypots can also be used to study the adversary's behaviour, discovering the attacking tactics and skills. The case in point is when the intrusion response system (IRS) always diverts the intrusion traffic to honeypots that are located on a virtualization platform. The IRS honeypots can be used to collect and analyse the activities

S. Boumerdassi et al. (eds.):MSPN 2015, LNCS 9395, pp. 1–17, 2015.
DOI: 10.1007/978-3-319-25744-0_1

of the community of intruders, capture their keystrokes and attack conversations. Honeypots are an important part of the infrastructure of an IRS because they enable the security expert to react to the attack.

Therefore, honeypot systems are useful and important to secure production systems. However, the usefulness of one single honeypot is limited. In order to provide a better protection, honeynets, defined as a network of honeypots [13], were created. A honeynet can include various honeypots just like a "zoo" has many animals. For example, a honeynet can consist of several low-interaction and high-interaction honeypots as a hybrid system to get a good balance among scalability, fidelity and performance. A honeynet can also be used to clone a target network for security research purposes. Additionally, a honeynet can receive the redirected intrusion traffic to investigate the adversary's behaviour.

Although honeynets have many advantages, deploying a honeynet is not an easy task. On the one hand, it is difficult achieve scalability in honeynet deployment when using physical honeypots, due to physical machine resources cost. On the contrary, the use of virtualized honeypots is a safety and effective way to deploy honeynets. Thus, virtualization technologies are very efficient alternative to deploy honeynets, as one physical machine can run multiple virtual machines simultaneously. Therefore, using virtual machines to run honeypots is cost-efficient and can simplify their management. On the other hand, honeynet configuration is another challenging task. Traditionally, security experts determine the configuration of the honeynet beforehand and reconfigure it manually when the honeynet needs to be redeployed. This traditional static deploying scheme has several shortcomings. First of all, manually configuring a honeynet is generally a complex and costly task. Secondly, a static honeynet deploying scheme is not able to react to an intrusion event. Thirdly, it is not flexible enough to adapt the change of the target network in real time. Thus, it is meaningful to conquer the challenge of dynamic configuration and deploy the honeypots by using virtualization technology.

Although the well-known Honeyd tool [2] can dynamically reconfigure the emulated virtual honeypots and services, it only provides low-interaction honeypots. The availability of dynamic configuration for high-interaction honeypots depends on the ability of the virtualization platform where honeynets are deployed. However, there is not any ready-made virtualized tool that can provide dynamic configuration for high-interaction honeypots deployment. Therefore, in this paper, a novel adaptable virtual honeynet architecture is proposed. This easy to manage adaptable virtual honeynet management tool allows the dynamic reconfiguration of the honeynets deployed and it is also very flexible because it allows deploying multiple honeypot types and even hybrid systems. The tool is designed to allow the easy management of virtual honeynet, hiding the user all the underlying virtualization complex details.

The organization of this paper is as follows: in Sect. 2 related work is described; in Sect. 3 a honeynet are defined and described in detail; in Sect. 4, the new virtual honeynet architecture is proposed; in Sect. 5 some experimental results and measurements are demonstrated; finally, in Sect. 6 some conclusions are presented and some future work is suggested.

2 Related Work

Honeyd [2] provides a virtual honeypots framework, which can simulate multiple honeypots simultaneously following certain network topology. However, it has several shortcomings. First of all, it is a software solution that only focuses on low-interaction honeypots simulation. Secondly, though Honeyd can dynamically reconfigure every template, it cannot change the simulated route configuration, which must be set beforehand. Third, every template only can set one Ethernet with MAC address, but it can bind multiple IPv4 addresses. Thus, Honeyd has its own original limitation. While, because of its lightweight and distributed appearance as well as its dynamic feature, Honeyd still has many applications. For instance, it can be used to clone the target network for research purposes [3]. It also can be employed to study the algorithms for hiding honeypots [4, 5].

With the generalization of the use of virtualization technologies, more and more virtual machine based solutions were proposed to create honeypots. For instance, User-Mode Linux (UML) as the virtualization engine was used in the virtual honeynet architecture [6] to mimic Gen II honeynets. In this case, the hosting machine acts as Honeywall to monitor the virtual honeypots running in the hosting machine. The built-in tty logging mechanism enables the keystroke sent to hosting machine to be silently captured. Hence, using UML virtualization technology, the proposed honeynet architecture is much more portable, easier to setup and more cost effective. The author proposed an alternative approach to deploy honeynet based virtualization technology, which is very inspired for other system administrators. However, this virtual honeynet architecture did not provide the dynamic configuration capability, which is a limitation for meeting the requirement of current honeynet research.

Another similar case in point is the use of VNUML (Virtual Network User Mode Linux) in virtual honeynet deployment [7]. The authors of this work found UML (User-Mode Linux) as a powerful but complex tool: it is generally difficult to manage medium-to-big size scenarios by hand. Thus, they devised a high-level description of the virtual honeynet and build it without dealing with virtualization low-level complex details. VNUML had two components proposed: a simple descriptive XML-based language that can specify a honeynet scenario to be simulated; and an application that interprets the honeynet description and generates and manages the honeynet scenario inside the hosting machine. The authors provided a case using VNUML to mimic a GenII honeynet. However, this work still did not provided the capability of dynamic reconfiguration of the honeynets deployed. The user has to describe the honeynet scenario beforehand manually by using the XML-based language. Very similar work to VNUML at first sight is NoSE (Network Simulation Environment) [14]. VNUML still focus on using UML to build honeynet and there seems to be no effort going beyond UML, while NoSE integrates different virtual machine emulators such as Xen, UML, and QEMU, hence it can support to emulate various operating systems. But the honeynets created by NoSE are still static.

Other virtualization technologies like VMware also can provide the ability to create a specialized network of hosts on a single physical machine. For example, the authors of the work [8] shared their experiences with a Generation III Virtual Honeynet deployment by

using VMware server. But the author also did not devise the capability of dynamic configuration for the VMware server. Compared with previous virtual honeynet architectures, this work did not place the Honeywall on the host, but installed it on a single virtual machine. The data capture tool namely Sebek [9] was used to capture the keystroke in virtual honeypots.

Most recently, some new virtualization technologies were proposed to facilitate more capable honeynet architectures. For instance, the decoys based on KVM (Kernel-based Virtual Machine) hypervisor [21] can provide low-level surveillance from outside the guest OS, which can keep the system activity monitor stealthy and the intruder has no way to bypass that surveillance. Besides, LXC (Linux Containers) virtualization provides a lightweight alternative to hypervisor-based virtualization [22]. LXC can create multiple isolated Linux user-space instances by partitioning the resource of the host. Thus, the startup of a LXC based virtual machine is much quicker than that based on KVM, but LXC can only be used to emulate Linux over Linux, but not other operating systems.

3 Honeynet Description

3.1 Basic Concepts

(1) Honeypot. In general, any computer system with no authorized value can act as a honeypot. Additionally, instead of a computer system, the definition implies that a honeypot can also be a digital entity. For example, in Cliff Stoll's book "The Cuckoo's Egg" [10], we learn how he deploys digital files to track and monitor a German hacker. Honeytoken [11] is the terminology for this kind of digital entity acting as honeypot. In summary, a honeypot can be either a computer system or more generally a digital entity.

A physical machine running as a honeypot is named a *physical honeypot*, while a virtual machine running as a honeypot is named a *virtual honeypot*. The main concept we must keep in mind is that a virtual honeypot is simulated or emulated inside another machine that responds to network traffic sent to the virtual honeypot [12]. In addition, the honeypots represented by software solutions are considered virtual honeypots too. Hence, for example, a honeypot emulated by a software solution running on a physical machine is a virtual honeypot instead of physical honeypot.

Honeypots can be classified into low-interaction honeypots and high-interaction honeypots. High-interaction honeypots can provide unlimited functions and has no difference with the conventional information system resource. Thus, high-interaction honeypots can be computer system or honeytoken. On the contrary, low-interaction honeypots are always emulated by honeypot software solutions that only provide a limited or even minimum set of functions.

(2) Honeynet. In a sense, a honeynet is an extension of the honeypot concept. The narrow honeynet term refers to Gen II honeynet that is an interaction type of honeypot solution [13], while the generalized honeynet term means a network of honeypots. In this paper, the honeynet is not limited to its narrow definition. Our proposed definition is that a honeynet is a network of honeypots following certain network topology.

A physical honeynet is made of honeypots running in separate physical machines. A virtual honeynet is a honeynet made of virtual honeypots running over one or more physical machines. By using virtualized tools, all the honeypots are virtually housed in one or more machines, but they still appear to the attacker like being different separate machines.

Some honeypot software solutions have the capacity of generating virtual honeynet, i.e. Honeyd. However, the honeynets created by Honeyd only consists of low-interaction honeypots. High-interaction honeypots running on virtual machines also can form virtual honeynet.

3.2 Honeynet Functionalities

Data Control – Data Control functionality is aimed to mitigate the risk that the adversary uses the compromised honeypot to attack other non-Honeynet systems, such as any system on the Internet. The outbound attack must be controlled in order to protect the non-Honeynet systems, but, at the same time, we have to minimize the attacker's or malicious code chance of detecting it. Thus, the challenge of data control is how to set the threshold of outbound activity. The more you allow the attacker to do, the more you can learn. However, the more you allow the attacker to do, the more harm they can potentially cause. It is a complex task to hunt the balance between how much intrusive data you want to get and how much baleful activity you want to restrict.

Data Capture – The purpose of Data Capture is to log all of the attacker's activity for later investigation. Three critical layers of Data Capture were identified: firewall logs (inbound and outbound connections), network traffic (every packet and its payload as it enters or leaves the honeynet), system activity (attacker's keystroke, system call, modified files, etc.). The more data and the higher quality of the data the honeynet can capture, the better the honeynet is.

Data Collection – Data Collection requirement proposes the secure means of centrally collecting all of the captured data from distributed honeynets. Due to the fact that honeypots are themselves insecure systems, the captured data must be centralized in an external secure system. On the other hand, if the data were distributed stored, it is not an easy way to manage. Thus, Data Collection also provides an easier management of the captured data.

Another requirement is that honeynets should be stealthy. In other words, the Data Control, Data Capture and Data Collection should be hidden or camouflaged to avoid the adversary to be aware of these honeynet activities. The adversary has many ways to detect if he is in a honeynet, for example, by detecting whether he is in an environment set up to record his activity, in which case, the adversary will erase all his tracks and break the connection with the honeynet and then no data would be recorded.

3.3 Honeynet Architecture

(1) Physical Honeynet Architecture. The physical honeynet architecture proposed by the Honeynet Project [13] has evolved across 3 generations.

Generation I – Gen I Honeynet was developed in 1999 by the Honeynet Project. Figure 1 shows a graphical representation of Generation I Honeynet architecture. As can be seen, a firewall separates the network into three different parts: Internet, Honeynet and Administrative Network. This architecture is made up of several components, which are used to facilitate the main honeynet functionality.

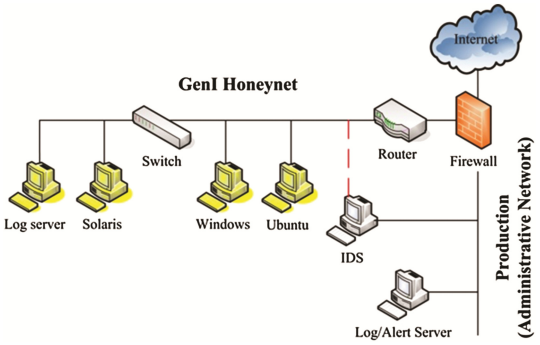

Fig. 1. Gen I honeynet architecture

The firewall keeps the track of any connection that has been made between the honeynet and the Internet. The firewall can block the outbound connections once a defined limit has been reached for the goal of Data Control. On the other hand, the firewall also logs all connections to and from the honeynet for the goal of Data Capture.

The router is located between the firewall and the honeynet. It hides the firewall from the attacker and provides the attacker a more realistic network with a production router in order to keep the attacker from becoming suspicious. Besides, the router also works for Data Control, for example, it only allows packets that have the source IP address of the honeynet to leave in order to protect against attacks such as spoofing, DoS, and ICMP based attacks.

The IDS connects to the honeynet via a physical switch. Thus, the IDS can use a "port monitoring" port enabling it to record all network activity for the goal of Data Capture. In Gen I Honeynet architecture, the IDS is signature based. When a packet matches a signature, an alert with detailed information about the connection will be provided by the IDS though any traffic on a honeynet is considered suspicious.

Finally the attacker reaches the honeynet. The honeypots capture all system activity and the remote log server can provide the centralized Data Collection. If an advanced attacker detects the syslog and even compromises the remote log server, we still have the IDS that act as a backup remote log system.

There are several limitations of Gen I Honeynet. First of all, if the limit of outbound connections is reached and all of the attacker's outbound activity is blocked, the attacker will suspect he is in a honeynet environment. Secondly, the firewall is easy to be identified since all traffic that passes through the firewall has TTL (time to live) decrement. Thirdly, the data such as keystroke and user activity are captured at the network level. IDS such as Snort can capture protocols such as FTP, Telnet or HTTP, which are plaintext. However, the attacker can use encryption technology such as protocol SSH to transfer the data. Thus, it will be failed if we monitor the attacker's connection to capture keystroke and user activity. Last but not least, the deployment is also limited, the Gen I Honeynet must be deployed on an isolated network otherwise the non-honeynet systems will be dangerous.

Generation II – In 2001, Honeynet Project released a honeywall, called eeyore, which allowed for Gen II Honeynet architecture and improved both Data Capture and Data Control capabilities over Gen I Honeynet architecture. Figure 2 illustrates the Generation II Honeynet architecture.

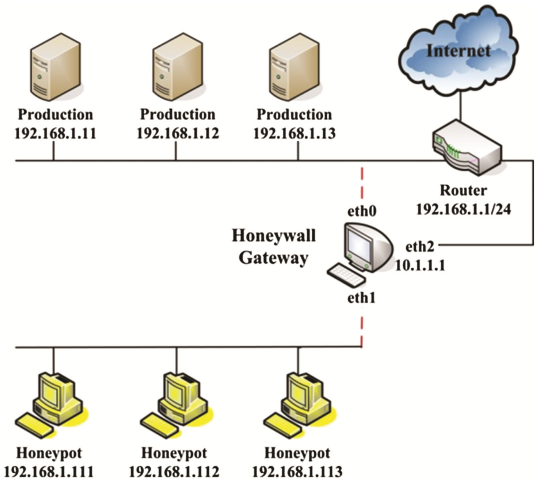

Fig. 2. Gen II honeynet architecture

Honeywall is traditionally a layer 2 bridging device with three interfaces. Two of them (eth0 and eth1) are used to segregate the honeypots traffic from the production network. They are bridged interfaces with no IP stack. The third interface (eth2, which is optional) has an IP stack and is used for remote administration. The honeywall combines the functionality of both the IDS and the firewall in a single system, which can perform attack control and network activity logging. The deployment of Gen II Honeynet is much easier than Gen I Honeynet. There are several advantages of using the honeywall.

Firstly, the honeywall is implemented as a transparent bridge to the attacker, meaning the device should be invisible to anyone interacting with the honeypots. Any traffic going to or from the honeypots must go through the honeywall but there is no routing of packets, no TTL decrement of system hops, thus the honeywall is hard to detect.

Secondly, instead of relying on a layer-three firewall that applies Data Control based on IP headers. Gen II honeynet applies a technology called IDS gateway, which not only block connections based on service, but it also has the intelligence to distinguish between an attack and legitimate activity.

Thirdly, depending on the advantage of the layer 2 interfaces, the honeynet deployment can be part of a production network instead of being on an isolated network. Although in reality all the systems including honeypots and production systems are part of the same network, the honeywall divides the honeynet from the production network at layer two, as opposed to layer three.

In addition, for the purpose of Data Capture of system activity, Sebek [9] kernel modules were developed to modify the system kernel in order to record system activity, especially keystrokes, in a harder way to be detected.

The honeywall is also used for Data Collection. If there are several Gen II honeynets deployed in a distributed environment, the data can be encrypted by some technology such as IPsec tunnels to a central point by the third interface of the honeywall and then all distributed honeynets are managed.

However, Gen II Honeynet architecture still has several limitations: Sebek and eeyore are no longer maintained and Sebek can currently be detected easily.

Generation III – In the summer of 2005, Honeynet Project released a new honeywall namely roo, which enables Gen III Honeynet architecture. Indeed, Gen III has no architectural difference from Gen II, but the roo improved the data model over eeyore.

Firstly, Roo improved Data Capture capability by introducing a new hflow database schema and pcap-api for manipulating packet captures. Secondly, it improved data analysis capability by introducing a new web based analysis tool called walleye. And thirdly, it improved installation, operation, customization and the user interface.

However, the Gen III Honeynet architecture still has some limitations. The problem of system activity capture is still unresolved. The honeynet deploying on physical machine is difficult for dynamic configuration.

(2) Virtual Honeynet Architecture. As stated, a virtual honeynet is a solution that allows running multiple honeypots simultaneously over a single physical machine by using virtualization software. Virtual honeynets can be broken into two categories, Self-contained and Hybrid virtual honeynet.

Self-contained – A self-contained virtual honeynet is an entire honeynet network deployed over a single computer. The left part of Fig. 3 presents an overview of self-contained virtual honeynet architecture.

The physical machine running the host operating system includes the whole honeynet architecture that consists of data control and data capture tools and the virtual honeypots running separately different guest operating systems. Thus, this virtual honeynet architecture is very portable and cost effective. Another advantage

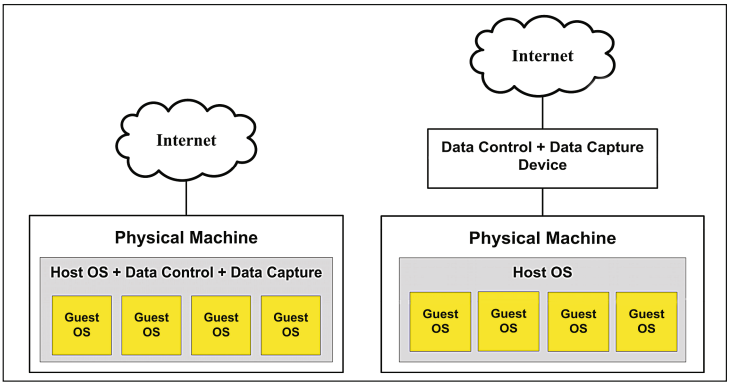

Fig. 3. Self-contained virtual honeynet and hybrid virtual honeynet

is the convenience of Data Collection because it is not necessary to use encrypted tunnel to collect data but we can use some system technology to log the data on the virtual honeypots.

However, the main drawback of this virtual architecture is that since all the services run on one physical machine, if that machine fails or it is compromised then the whole honeynet will break down. Furthermore, service performance is another problem. If the hardware and the virtual software provide a limited service performance, the attacker may easily detect the virtual honeynet environment.

Hybrid – A hybrid virtual honeynet is a combination of the data control and data capture implemented in physically separate machine and the virtual honeypots running on another computer. The right part of Fig. 4 exhibits an overview of hybrid virtual honeynet architecture.

This isolation of the data control and data capture can reduce the risk of honeynet compromise. Thus, this virtual honeynet architecture is much more robust than the self-contained virtual honeynet architecture. Moreover, this virtual honeynet architecture can get a better service performance profit from the physically separate deployment.

Nevertheless, the hybrid virtual honeynet architecture still has several shortcomings. Firstly, this virtual honeynet architecture is not as portable as the self-contained virtual honeynet architecture because there are at least two machines. Secondly, due to the physically separate deployment the cost is not as efficient as the self-contained virtual honeynet architecture.

4 New Virtual Honeynet Architecture

In this section, a new virtual honeynet architecture is proposed. The creation of complex honeynets results in the need of specialized tools to manage them. They should facilitate their definition – topology, addresses, and types of systems, among others–, their deployment, their monitoring, and their security, hiding all the complexity of the underlying virtualization platforms to the user. These specialized tools also must achieve the goal of data control and data capture. In our case, the following tools were selected:

VNX [16] as high-interaction honeypots and virtual network generation tool; Honeyd [2] as low-interaction honeypots creation tool; Nitro [17] as system data capture tool; Snort as intrusion detection tool; and Honeybrid gateway [18] as the network data capture and control tool. Figure 4 presents an overview of our adaptive and flexible virtual honeynet architecture.

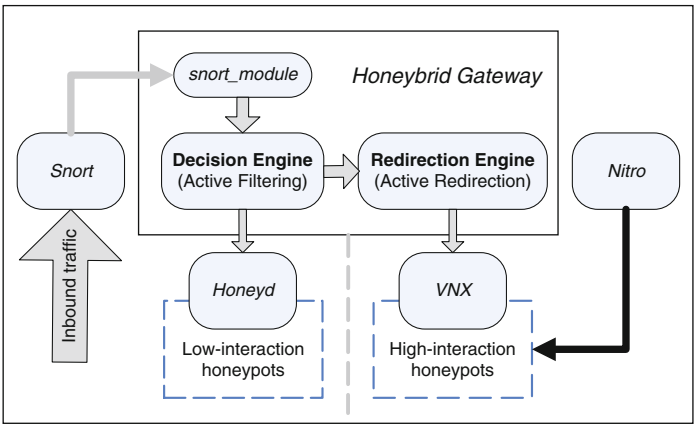

Fig. 4. An overview of the new virtual honeynet architecture

We use these specialized tools as the components for hosting the new adaptive and flexible virtual honeynet, but our ideas are not limited to these tools. For the future research, other advanced specialized tools can replace the current tools based on the architecture.

Currently virtual honeynet does not mimic Gen III honeynet, but similar to other virtualization software, VNX also can host a Gen III honeynet. Indeed, our virtual honeynet improves the Gen III honeynet. Firstly, our virtual honeynet can bridge into a production network or deploy in an isolated environment, thus the deployment strategy is flexible. Secondly, the combination of low-interaction honeypots and high-interaction honeypots provides a good balance among scalability, performance and fidelity. Thirdly, we use the out-of-the-box approach to capture the system activity which is much stealthier. Fourthly, the capability of dynamic configuration allows the virtual honeynet to adapt itself to the real-time network environment. The following discussion depicts each tool in detail.

VNX – There are several tools available to create and manage virtual network scenarios, such as VNX (Virtual Networks over linuX), Netkit, MLN (Manage Large Networks), and vBET (VM-Based Emulation Testbed). Among them, VNX emerges as the most powerful solution due to (i) its scalability in creating very complex high interaction honeynets, due to the ability to deploy virtual network scenarios over a cluster of servers; (ii) its ability to automatize the execution of commends inside virtual machines; and (iii), its support for multiple virtualization platforms like KVM, which allows emulating various operating systems for high-interaction honeypots on demand.

Last but not least, the new version of VNX has included the capability of dynamic configuration. In other words, the user can reconfigure the scenario on the fly. The user only needs to write a reconfiguration file based on XML syntax, and later he can use VNX to automatically reconfigure the scenario by processing the reconfiguration file. One part of the running scenario can be redeployed on the fly without impacting on the rest part of the on-going honeynet. Figure 5 shows the state diagram of a high-interaction virtual honeypot emulated by VNX.

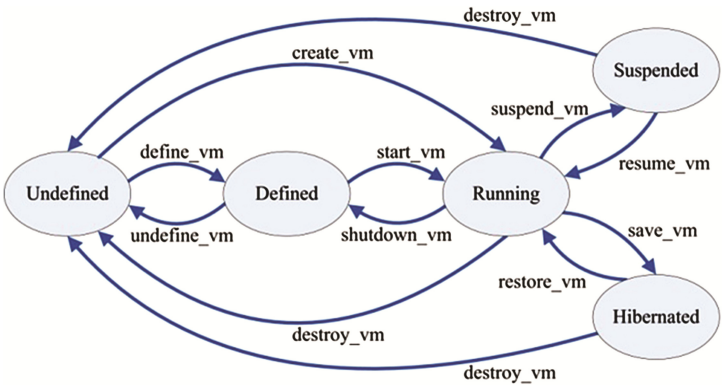

Fig. 5. State diagram of a virtual machine emulated by VNX

A virtual machine emulated by VNX has five states. They are undefined, defined, running, suspended and hibernated. This state diagram follows the states model defined in libvirt and practically uses the same terminologies. VNX has developed functions that can switch a virtual honeypot from one state to any other state.

The undefined state is the unborn state of virtual honeypot. It means the virtual honeypot node is not defined in the honeynet scenario. The defined state is the initial state of virtual honeypot, which indicates that the virtual honeypot exists in the honeynet scenario but the system is power-off. The running state is the normal working state of virtual honeypot. The virtual honeypot on this state can provide services. The suspended state is the power saving state. This method cuts power to most parts of the machine aside from the RAM, which is required to restore the machine's state. The hibernated state is resource saving state. This approach saves the machine's state into swap space and completely powers off the machine. When the machine is powered on, the state is restored. Until then, there is zero power consumption. The virtual honeypot can spin up much quicker from suspended state than from hibernated state, but the hibernated state is much more energy efficiency than suspended state.

Therefore, VNX can host the adaptive and flexible high-interaction honeynet.

Honeyd – Honeyd is a well-known low-interaction virtual honeypot framework. It can quickly deploy a virtual honeynet integrating into a target network or standing alongside a target network. It can emulate fingerprints of various operating systems.

Honeyd has a doorway called Honeydctl to communicate the inner workings of Honeyd. Honeydctl can be used to reconfigure the templates on the fly. All commands

used in the regular configuration file of Honeyd template can be interactively used after "honeydctl>" prompt, however, this is also the shortcoming of Honeydctl. The user has to input the commands interactively, one command by one. But, instead of interactively interacting with Honeydctl, our system requires an automatic capability of reconfiguration. Fortunately, the author of Honeyd leaves us a UNIX socket located in/var/run/honeyd.sock. Using this socket, we can create a client socket to communicate with the inner workings of Honeyd, in other words, we can reconfigure the Honeyd templates by the client socket with a script consists of Honeyd commands. Every template has two states, created (online) and deleted (offline). We can use Honeyd commands two spin up and down any template and even any service of a template freely.

In summary, Honeyd can provide the adaptive and flexible low-interaction honeynet.

Nitro – In the Gen II and Gen III honeynet, Sebek/Qebek [15] is the monitor tool used to track the system activity. The main drawback of this kind of kernel module is that it must be installed inside the virtual honeypot to do use an in-the-box monitor approach. However, this monitoring approach can be easily be detected by the adversary. Another monitor method using the idea called "out-of-the-box" [19] is proposed in order to prevent the monitor tool from being detected. In this paper, we employ the tool Nitro to monitor the high-interaction honeypots from out-of-the-box, since it focus on track the system call of the KVM based virtual machine.

Thus, Nitro is applied to capture the system activity but the potential monitor tool is not limited to it.

Honeybrid Gateway – Honeybrid consists of three parts: a gateway, a set of low-interaction honeypots (front-end) and a set of high-interaction honeypots (back-end). The Honeybrid gateway is the central part including the Decision Engine and the Redirection Engine in charge of orchestrating the filtering and the redirection between front-end and back-end. The Decision Engine is used to select interesting traffics and the Redirection Engine is used to transparently redirect the traffic from low-interaction honeypots to the farm of high-interaction honeypots. A contribution of Honeybrid is that a simple classification of interesting attacks is proposed: attacks matching a specific fingerprint (well-known attacks); attacks presenting an original content that was never seen before (0day attacks); attacks sending commands that are not implemented in the low-interaction honeypots (i.e., not implements in Honeyd). Using this classification, Honeybrid can effectively redirect the attacks worth of further investigation to high-interaction honeypots.

The main drawback of Honeybrid is that it is not able to distinguish the malware automated attacks and human manual attacks. More accurately speaking, it neglects the human manual attacks. In the current design, a pair of low-interaction honeypots and high-interaction honeypots has different IP addresses in a honeynet deployment, and the Honeybrid gateway can transparently redirect the traffic from the low-interaction honeypot to the corresponding high-interaction honeypot. It is useful to catch the malware automated attacks but if the attack is from an intelligent human adversary, he can easily detect the traffic redirection by simply checking the IP address of the final compromised system. If the final IP address of the compromised system is different from

his original attacking destination IP address, the adversary can convince that he is immersed in a honeypot. As shown in Fig. 5, the Honeybrid gateway is not only a data controller, but also a function controller. When it decides to redirect the interesting traffic from the low-interaction honeypot to the high-interaction honeypot, before it replays the connection, it must switch off the low-interaction honeypot and spin up the corresponding high-interaction honeypot.

Therefore, Honeybrid can play the role of network data capture and control.

Snort and Other Honeybrid modules – Snort is used as the NIDS (network intrusion detection system) to monitor the virtual honeynet. In our virtual honeynet, it is an auxiliary tool for Honeybrid. Because Honeybrid is a programmable tool that provides API to develop new modules for its data control. The security expert can develop a Snort module for Honeybrid blocking the well-known attacks. As such, we allow the Snort NIDS mode to send alerts and output to a UNIX socket that the new Honeybrid module can listen on. The Decision Engine can decide to accept the packet or discard it according to the output from Snort. Besides, we also can develop other modules to achieve other data control goals.

5 Experiments and Measurement

In this section, the proposed virtual honeynet is validated by some experiments. We also compare our virtual honeynet architecture with the existing honeynet architectures in term of the standard measurement criteria provided by Honeynet Project.

5.1 Experiments

Our virtual honeynet can be applied to self-contained virtual honeynet or hybrid virtual honeynet. In this paper, we use the self-contained approach to present our virtual honeynet architecture. The testbed architecture is shown in Fig. 6.

The incoming traffic can firstly connect to the low-interaction honeypot through br0. If Snort detects a well-known attack, the Honeybrid gateway can discard the packet. After the traffic penetrates the NIDS, the Honeybrid gateway should decide to filter the packet or redirect the traffic into a high-interaction honeypot. If the decision is redirection, so before the connection replay, the low-interaction should be deleted and the corresponding high-interaction honeypot has to be waked up. As soon as the high-interaction honeypot get into running state, the interesting traffic is redirected into it.

The possible problem is the delay to start a high-interaction honeypot. So, we test the duration to start or wake up a VNX based high-interaction honeypot. The system parameters of the host machine are: CPU, 4 Intel(R) Core(TM) i5-3470 CPU @ 3.20 GHz; RAM, 16 GB; OS, Ubuntu 14.04; Kernel, Linux 3.13.0-24-generic.

VNX can deploy KVM based virtual machine and LXC based virtual machine. The delay is very short to start up a LXC based virtual machine. It is less than 1 s. But the virtual machine based on LXC only can emulate the Linux operation system that uses the same Linux kernel with the host. Thus, this method lacks fidelity. While virtual

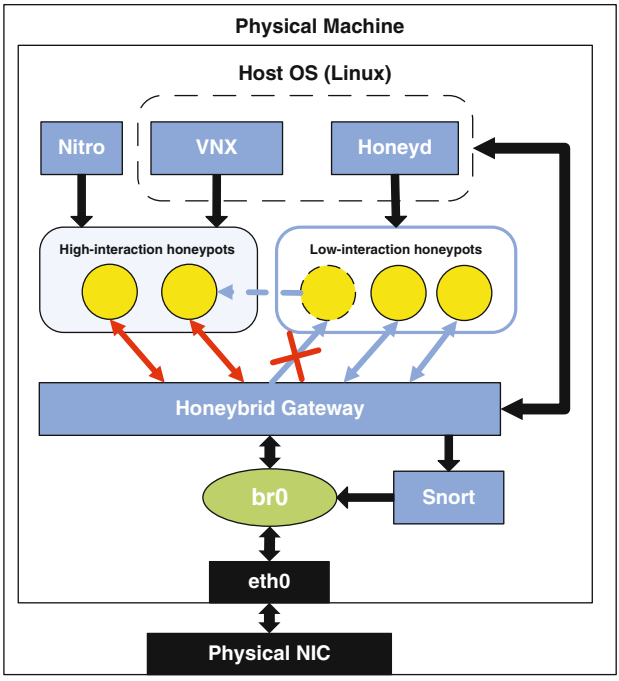

Fig. 6. Testbed Architecture

machine based on KVM can emulate various operating systems. The startup delay of a KVM based virtual machine from different states is demonstrated in Fig. 7.

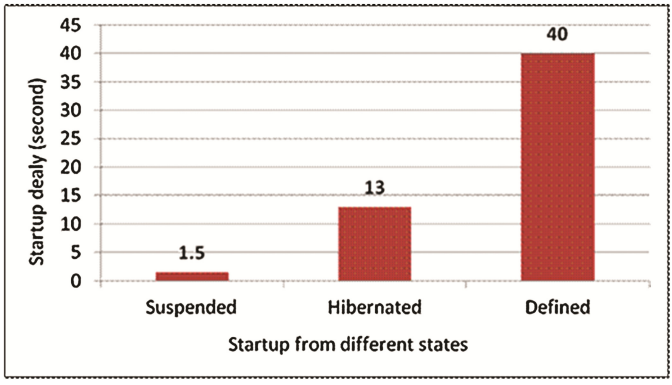

Fig. 7. Startup delay of different states

Thus, to start up or reboot a KVM based virtual machine, the time cost is 40 s. This delay is too long and could result in time out of connection request. To wake up a hibernated virtual machine is not so long, but still need 13 s, thus it is not a good choice either. Lastly, a suspended virtual machine spins up only within 1.5 s. Hence, the

suspended is the best state that has the shortest delay to wakeup. Therefore, the approach for KVM based virtual honeypots is to start up a group of virtual honeypots and then keep them into suspended state. When the interesting traffic is decided to be redirected into the high-interaction honeypot, then the corresponding suspended virtual honeypot is wake up to provide service.

5.2 Measurement

Table 1 compares some criteria of measurement provided by the Honeynet Project [20] plus new features namely adaptability provided by us.

Table 1. Comparison of honeynet features

Item	Gen III [8]	UML based [6]	Our work
Portable	No	Yes	Yes
Setup	Involved	Plug and catch	Plug and catch
Cost	High	Low	Low
Flexibility	Very	Limited	Moderated
Security	Secure	Software dependent	Software dependent
Detectability	Reasonable effort	Reasonable effort	Hard to detect
Adaptability	No	No	Yes

The Gen III [8] is the traditional physical virtual honeynet, and the UML based virtual honeynet [6] is the typical self-contained virtual honeynet that is a representation of previous virtual honeynets. Each measure is elaborated below.

Portable is the feature indicates if the honeynet can be moved around. Gen III honeynets are not potable since each components reside on its own physical machine. The self-contained virtual honeynet can be configured and deployed in a single physical machine, thus the UML based virtual honeynet and this virtual honeynet is portable.

Setup is measure of the ease of setting up a new honeynet at a new location. As the physical honeynet, Gen III honeynets have to set up each physical machine individually. However, self-contained virtual honeynets are "plug and catch", which can quickly deploy a whole virtual honeynet without driver conflicts or other hardware related configuration problems

Cost is a measure of hardware cost. Since Gen III honeynets require physical machine to run every component, the hardware cost is high. On the contrary, self-contained virtual honeynets can be setup on a single physical machine therefore hardware cost is low.

Flexibility is a measure of what types of honeynets can be configured. UML is only able to support Linux based operating systems, while KVM can support operating systems that run on the x86 processor architecture. The physical machine can support any kind of operating system with the right hardware.

Security is a measure of how susceptible the honeynet is to be compromised and becomes a threat to systems on the outside. In Gen III honeynet, the security depends on the honeywall. Since the honeywall is tightly configured and access is limited to network based methods, the security of Gen III honeynets is high. However, the security of virtual honeynet depends on the virtualization software. Protection against host-based attacks is provided by software running on the host.

Detectability is a measure that indicates if the honeynet can be detected by the attacker. Kernel module based monitor is easy to be detected by the blackhats currently, thus Gen III honeynets and UML based virtual honeynets are detectable. We use the "out-of-the-box" approach to monitor the system call, thus it is hard to be detected.

Adaptability is a measure of what kinds of honeynets can be configured dynamically. Our virtual honeynet system provides the capability of dynamic configuration and deployment, but the previous work didn't provide this kind of capability.

6 Conclusion and Future Work

In this paper, a new adaptive and flexible virtual honeynet architecture has been proposed. Compared to the previous honeynets, our new honeynet has many advantages, like the dynamic configuration capability, which isn't supported by previous honeynets. Furthermore, as a virtual honeynet, our work overcomes the limitation of operating system supporting. Our virtual honeynet is quite flexible and it can support most of the operating systems that runs over x86 processor architecture. In addition, our virtual honeynet architecture is also flexible, since the security experts can replace the recommended tools with future advanced tools.

Since our virtual honeynet is programmable, for the future work, we will develop much more modules for Honeybrid to provide much better data control. Additionally, we also will design and develop a tool that can manage the whole virtual honeynet system. We hope other security experts can benefit from our work.

Acknowledgement. This research is supported in part by National Natural Science Foundation of China (No. 61440057, 61272087, 61363019 and 61073008), Beijing Natural Science Foundation (No. 4082016 and 4122039), the Sci-Tech Interdisciplinary Innovation and Cooperation Team Program of the Chinese Academy of Sciences, the Specialized Research Fund for State Key Laboratories. It also has been partially funded with support from the Spanish MICINN (project RECLAMO, Virtual and Collaborative Honeynets based on Trust Management and Autonomous Systems applied to Intrusion Management, with codes TIN2011-28287- C02-01 and TIN2011-28287-C02-02) and the European Commission (FEDER/ERDF).

References

1. Spitzner, L.: The Value of Honeypots, Part One: Definitions and Values of Honeypots, 10 Oct 2001. http://www.symantec.com/connect/articles/value-honeypots-part-one-definitions-and-values-honeypots
2. Provos, N.: A virtual honeypot framework. In: SSYM 2004 Proceedings of the 13th Conference on USENIX Security Symposium, vol. 13 (2004)

3. Hecker, C., Hay, B.: Automated honeynet deployment for dynamic network environment. In: 2013 46th Hawaii International Conference on System Sciences (HICSS), pp. 4880–4889, 7–10 Jan 2013
4. Fu, X., Bryan, G., Cheng, D., Bettati, R., Zhao, W.: Camouflaging virtual honeypots. In: Texas A&M University (2005)
5. Wang, H., Chen, Q.: Dynamic deploying distributed low-interaction honeynet. J. Comput. N. Am. **7**, 692–698 (2012)
6. Yan, L.K.: Virtual honeynets revisited. In: Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop, IAW 2005. pp. 232–239, 15–17 June 2005
7. Galán, F., Fernández, D.: Use of VNUML in Virtual Honeynets Deployment. IX Reunión Española sobre Criptología y Seguridad de la Información (RECSI), Barcelona (Spain), September 2006. ISBN: 84-9788-502-3
8. Abbasi, F.H., Harris, R.J.: Experiences with a generation III virtual honeynet. In: 2009 Australasian, Telecommunication Networks and Applications Conference (ATNAC), pp. 1–6, 10–12 Nov 2009
9. Honeynet Project. Know Your Enemy: Sebek, A kernel based data capture tool, 17 November 2003. http://old.honeynet.org/papers/sebek.pdf
10. Stoll, C.: The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage. Pocket, New York (1990)
11. Spitzner, L.: Honeytokens: The Other Honeypot, 17 July 2003. http://www.symantec.com/connect/articles/honeytokens-other-honeypot
12. Provos, N., Holz, T.: Virtual Honeypots: From Botnet Tracking to Intrusion Detection, 1st edn. Addison-Wesley Professional, Boston (2007)
13. Honeynet Project. Know Your Enemy: Honeynets, 26 April 2001. http://www.symantec.com/connect/articles/know-your-enemy-honeynets
14. Stumpf, F., Görlach, A., Homann, F., Bruuckner, L.: NoSE - building virtual honeynets made easy. In: Proceedings of the 12th International Linux System Technology Conference, Hamburg, Germany (2005)
15. Honeynet Project. Know Your Tools: Qebek – Conceal the Monitoring, 03 Nov 2010. http://www.honeynet.org/papers/KYT_qebek
16. Fernandez, D., Cordero, A., Somavilla, J., Rodriguez, J., Corchero, A., Tarrafeta, L., Galan, F.: Distributed virtual scenarios over multi-host Linux environments. In: 5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM), pp. 1–8, 24 Oct 2011
17. Pfoh, J., Schneider, C., Eckert, C.: Nitro: hardware-based system call tracing for virtual machines. In: Iwata, T., Nishigaki, M. (eds.) IWSEC 2011. LNCS, vol. 7038, pp. 96–112. Springer, Heidelberg (2011)
18. Berthier, R., Cukier, M.: Honeybrid: a hybrid honeypot architecture. In: USENIX Security Symposium 2008 (2008)
19. Jiang, X., Wang, X.: "Out-of-the-box" monitoring of VM-based high-interaction honeypots. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 198–218. Springer, Heidelberg (2007)
20. Honeynet Project. Know Your Enemy: Defining Virtual Honeynets, 27 January 2001. http://old.honeynet.org/papers/virtual/
21. Capalik, A.: Next-generation honeynet technology with real-time forensics for U.S. defense. In: Military Communications Conference, MILCOM 2007, pp. 1–7. IEEE, 29–31 Oct 2007
22. Memari, N., Hashim, S.J.B., Samsudin, K.B.: Towards virtual honeynet based on LXC virtualization. In: 2014 IEEE Region 10 Symposium, pp. 496– 501, 14–16 April 2014