WORM-HUNTER: A Worm Guard System using Software-defined Networking

Yixun Hu, Kangfeng Zheng, Xu Wang, Yixian Yang
Information Security Center, Beijing University of Posts and Telecommunications
Beijing, 100876 - China
[e-mail: hyx.bupt@gmail.com]
*Corresponding author: Yixun Hu

Received August 9, 2016; revised November 19, 2016; accepted December 9, 2016; published January 31, 2017

Abstract

Network security is rapidly developing, but so are attack methods. Network worms are one of the most widely used attack methods and have are able to propagate quickly. As an active defense approach to network worms, the honeynet technique has long been limited by the closed architecture of traditional network devices. In this paper, we propose a closed loop defense system of worms based on a Software-Defined Networking (SDN) technology, called Worm-Hunter. The flexibility of SDN in network building is introduced to structure the network infrastructures of Worm-Hunter. By using well-designed flow tables, Worm-Hunter is able to easily deploy different honeynet systems with different network structures and dynamically. When anomalous traffic is detected by the analyzer in Worm-Hunter, it can be redirected into the honeynet and then safely analyzed. Throughout the process, attackers will not be aware that they are caught, and all of the attack behavior is recorded in the system for further analysis. Finally, we verify the system via experiments. The experiments show that Worm-Hunter is able to build multiple honeynet systems on one physical platform. Meanwhile, all of the honeynet systems with the same topology operate without interference.

Keywords: network worm, virtual dynamic honeynet, software-defined networking

1. Introduction

With the large number of security issues reported in recent years, network security is once again receiving a great deal of attention. Network worms are one of the main threats on the Internet. They are a type of malicious program that is able to self-propagate across a network, exploiting security or policy flaws in widely used services [1]. The damage that code-red [2], slammer [3], and other worms caused is still remembered by many. Once a single network node is infected by a worm, more than its own safety has been breached; yes, its sensitive information can easily be stolen and its behavior can be monitored and controlled, but its neighbors are also threatened. This is because the infected node can be used to execute attacks, which may not be conducted directly.

Defense against network worms is continuously studied by industry and academia. Generally, defense methods against network worms can be classified into three categories:

1) Use propagation models of a network worm that were built by analyzing its behavior to detect anomalies according to a protected node's behavior [4].

2) Use a signature matching technique, e.g., Snort [5] and antivirus programs, to compare network traffic and local programs with the signatures of network worms, which are extracted from worm program segments.

3) Build a honeynet system [6] to actively defend against network worms. A honeynet system is able to attract network worms, record the worms' behavior, analyze unknown worms, and track the attackers.

As the most widely used technology, honeynet systems implement an active defense system. However, on account of the closed traditional network architecture, it is hard to dynamically adjust the honeynet's structure. Once a honeynet system is deployed, it is so difficult to change its structure that the existence of the honeynet could be easily recognized by attackers. Moreover, a primary function of a honeynet system is to observe worm behavior. However, a traditional honeynet system cannot build various network environments on one honeynet platform at the same time. Researchers must waste large amounts of time, effort and money deploying multiple honeynet systems. Therefore, a dynamic defense system against worms that is able to track network worms over their entire lifecycles, including attraction, detection, cultivation, observation, and detection feedback, is in demand.

The above requirements motivate us to study the worm defense system. Software-Defined Networking (SDN) is a new network architecture that decouples the control and data plane, makes the network programmable, achieves a flexible network structure and allows for network virtualization. Researchers can easily control network traffic according to their demands. Inspired by SDN, we start with the following question:

If the traditional network architecture in a honeynet system is replaced by the SDN architecture, can we combine network virtualization from SDN and virtual honeypots from server virtualization to achieve a dynamic virtual honeynet system?

In fact, the answer is positive: we propose a dynamic virtual honeynet system (Worm-Hunter) based on SDN to defend against worms, from worm propagation to the end of worms, that builds a closed-loop defense system of worms based on SDN. In particular, we introduce the SDN architecture to build the network infrastructure platform in Worm-Hunter. Our structure meets the directing requirements of network control in various applications, e.g., directing traffic, cultivating worms, building models, and providing feedback. Upon this network structure, the network platform of Worm-Hunter is formed, on which various security strategies that entail different network environments can be easily carried out. Finally, we deploy the proposed system and validate that the approach is able to track the entire worm lifecycle and run multiple honeynets on one physical platform.

The rest of this article is organized as follows. In Section 2, we give an overview of some related work. Section 3 describes the entire system. Section 4 describes experiments that were run on the system, presents the results, and performs analyses. Finally, Section 5 provides the conclusion of this paper.

2. Related Work

Defense measures against network worms have been studied in many ways. Refs. [5, 7] use signature matching technology to match worms' payloads, which can be strings at special offsets or regular expressions. Researchers have built systems [8-9] to monitor network traffic and use signature-matching technology to detect network worms. However, along with the development of this technology, polymorphic technology has appeared, which can evade signature-based detection. In polymorphic technology, a program may encode and re-encode itself into different successive byte strings, enabling the production of changing worm payloads [11]. To address this problem, Ref. [11] proposes a polymorphic signature-generating system that generates three types of signature: conjunction signatures, token-subsequence signatures, and Bayes signatures. The signatures have a loose structure rather than fixed strings, which is more applicable to describe polymorphic worms. Deguang Kong et al. [12] apply the hidden Markov model (HMM) to the refined data to generate state-transition-graph-based signatures against worms' polymorphism. Refs. [13, 14] also propose solutions to generate polymorphic signatures of worms.

Nevertheless, signature matching-based detection methods are only able to detect known worms, not worms utilizing unknown vulnerabilities. However, anomaly detection technologies have drawn researchers' attention, which can find new worms without any prior knowledge. Refs. [15, 16] both use anomaly detection techniques to protect target systems, and [16] uses packet sampling methods to generate anomaly detection metrics for detection. Towards massive network environments, Wagner et al. [17] propose an entropy-based approach to detect network anomaly states by calculating the entropy contents of traffic parameters and observing the changes caused by malicious events. Ref. [18] demonstrates a detection method through network traffic characteristics. It calculates the port entropy in the network and observes the *Kullback-Leibler (K-L) divergence* with maximum entropy distribution to detect worms.

The defender is often passive in an attacker-defender game relationship. The above defense techniques are used after an attack has occurred, so they are not able to make the defender take an active role in the attacker-defender game. On the contrary, a honeynet [6] is a type of active defense technology that is able to actively attract worms to attack the honeynet system, and then the attack can be caught, observed and analyzed. A honeynet can be regarded as a combination of honeypots [19]. Honeypots can be divided into two categories according to their interaction level with attackers: low-interaction honeypots and high-interaction honeypots. A low-interaction honeypot uses service emulating technology to interact with attackers, while a high-interaction honeypot provides an entire operation system to attackers. Honeyd [20] is a famous low-interaction honeypot tool that can dynamically create multiple honeypots. However, the honeypot generated by Honeyd can only interact with attackers according to predefined action scripts. In fact, most attackers' target is a user's sensitive information. Attackers usually attack a system via a certain leak of a certain service and then elevate the privilege to access core services of systems or sensitive information. A low-interaction honeypot does not have other system services or users' information in addition to interaction templates, and attackers can recognize it easily. Moreover, honeynet systems with low-interaction honeypots always have problems in traffic directing. For example, Honeyd [20] has to use arpd (an ARP spoofing tool) for cheating routers and switches to direct network traffic into corresponding honeypots. The method can only address layer-2 forwarding (according to the MAC address), and loses effectiveness in regard to layer-3 forwarding (according to the IP address). As a result, high-interaction honeypots and a virtual honeynet system are proposed. Virtualization technology can dynamically deploy virtual operating systems on a single physical machine, which can be easily managed and widely used in many areas [21, 22], and therefore it can be used to build high-interaction honeypots. Ref. [22] designs a virtual honeynet system using virtualization technology and proposes the notion of a honeypot farm using created virtual machines. A honeypot farm is a honeypot resource pool that can be used to build a honeynet system. Ref. [23] proposes a virtual honeynet system on a VMware Server running Honeywall CDROM Roo [24]. It uses the flexibility of virtualization technology to create a specialized network of hosts on a single physical machine and reduces hardware costs. Generally, virtual honeynet systems use a software honeywall [19] to realize traffic forwarding. As a layer-2 bridging device, a honeywall combines the functionality of an IDS and firewall into a single device that performs attack control and network activity logging [24]. Nevertheless, developing such a software honeywall requires lots of manual work and high-performance hardware. In addition, the software running on the honeywall has to be consistently updated to meet new requirements. These drawbacks heavily limit the usage of honeywalls. Furthermore, although these honeynet systems use virtualization techniques to improve the interaction of honeypots and the traffic forwarding ability, they still have many restrictions. All the honeynet systems described above are built with traditional network devices and existing network protocols. The closed-network devices and protocols restrict the honeynet to the use of IP or MAC addresses as the identifiers of network traffic or network nodes. As a result, it is difficult to build multiple honeynet systems with the same network structure on the same defense platform. Multiple malicious programs can only be caught in the same honeypot, and security researchers are not able to observe malicious programs separately. As a result, the behavior of malicious programs may interfere each other and impact analysis. Researchers cannot catch different worms in different honeypots in the same honeynet and observe worms' behaviors separately. However, building multiple honeynet systems is costly.

In recent years, industry and academia have gradually recognized the advantages of SDN, such as the ability to control networks and the ability to create virtual overlay networks. SDN has been industrially realized from an academic concept. Different from traditional network devices, the control and data planes are decoupled in the SDN-enabled devices, and the network becomes programmable. As the most popular implementation of SDN, Openflow Protocol [25] has been the current SDN standard-bearing reference implementation with considerable momentum in industry [26]. Most network vendors (JUNIPER, Big Switch, and other manufacturers) have launched their SDN support devices [27, 28, 29]. In [26, 30-32], researchers have made different efforts for security issues using SDN. These studies are divided into two aspects: solving security issues in SDN and using SDN to provide security services. Ref. [26] proposes two data plane extensions to harden the security system in SDN networks. Ref. [30] studies the potential attacks in cloud computing and SDN environment. They find that SDN is able to defend against DDoS (Distributed Denial of Service) attacks and then propose a DDoS attack mitigation architecture.

The programmable network caused by Openflow/SDN brings a new direction for our work. We propose that the disadvantages of traditional honeynet systems described above can be conquered by introducing Openflow/SDN into the traditional honeynet design. With an elaborate design, we finish the combination of them and propose an SDN-based worm guard system.

The differences between our work and previous work lie in two points. First, we introduce the SDN technique into honeynet systems and build a dynamic virtual honeynet system that is able to dynamically build different honeynet systems with different network structures on the same honeynet platform at the same time. On the contrary, it is difficult to achieve such flexibility in previous work. Second, our system provides the overlay honeynet feature, which is able to build multiple honeynet systems with the same structure and same honeypots with the same network addresses on one physical platform.

3. Design

In this section, we first introduce the background knowledge about the Openflow Switch and the usage in Worm-Hunter. Then, we describe the architecture of the Worm-Hunter and the detailed design of each module in the system. To remove the inaccessibility brought about by traditional network architectures, we introduce the SDN technique to build the network basis and afford the system the ability to realize network virtualization. As a result, the system can build honeynets with different network structures dynamically. In addition, multiple honeynets can be simultaneously run on the system.

3.1 Openflow and SDN

Openflow is a mainstream implementation of SDN. Through Openflow protocol and support hardware devices, we can build an SDN-based network structure. In this structure, data forwarding and forwarding strategy management are isolated from traditional network devices (such as routers), as shown in **Fig. 1**. The SDN-based network structure implemented by Openflow consists of an Openflow Switch and Controller. The Openflow Switch is responsible for data forwarding, and the Controller is responsible for forwarding strategy management. The Openflow Switch and Controller are connected to the security channel. Network developers can easily control the network traffic directing strategy and realize multiple network applications based on the SDN-based network structure.

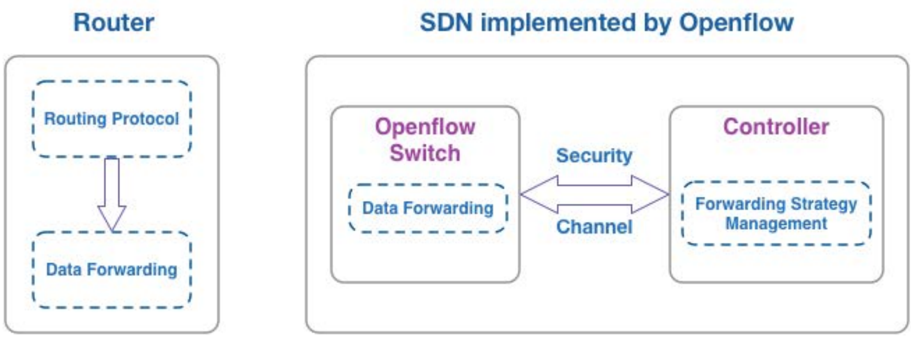


Fig. 1. Structures of the Router and SDN implemented by Openflow

Introducing Openflow/SDN into a traditional honeynet design is a core part of our work. In our work, the Openflow Switch is responsible for traffic directing, which is always taken charge of by software in traditional honeynet designs. By doing so, the complex and inflexible traffic directing method in traditional honeynet designs is changed. We design a Policy Controller to generate and manage traffic directing strategies. All the Openflow Switches in Worm-Hunter connect to the Policy Controller for receiving traffic directing strategies. The detailed procedure is described in the rest of this section.

3.2 Overall Architecture

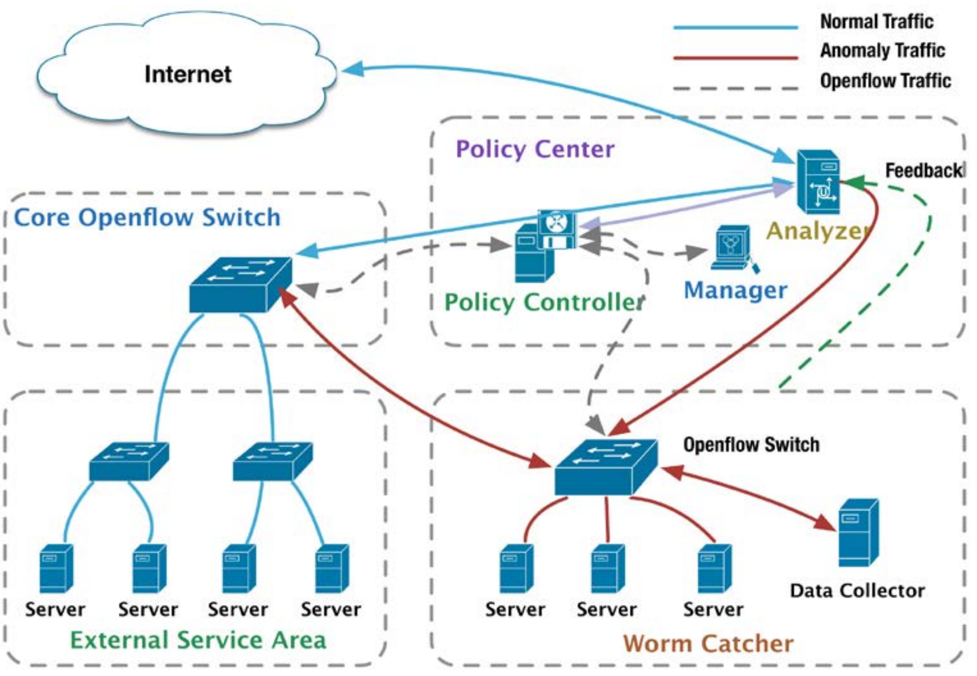


Fig. 2. Architecture of Worm-Hunter

Worm-Hunter can be deployed in enterprise networks. The aim of Worm-Hunter is to protect normal services of the Internet from network worms and other intrusion threats. The physical architecture of Worm-Hunter consists of four parts, as shown in Fig. 2.

- The **Core Openflow Switch**, located between the internal network and external network, is like the gateway in a traditional network architecture. It is responsible for directing the external traffic and carrying out security policies.
- The **Policy Center** contains an Analyzer, a Manager and a Policy Controller. The worm detection algorithms run on the Analyzer, including signature matching detection (using external traffic) and anomaly detection (using the statistic from the controller/SDN). The Policy controller is an Openflow/SDN controller with our security module on it.
- The **External Service Area** deploys servers with external services (HTTP, SQL, MAIL, etc.). External users can access these service via the Internet. These servers are cloned in the Worm Catcher to trick attackers using the virtual clone technique.

- The **Worm Catcher** is a dynamic virtual honeynet under the control of researchers through which suspicious traffic is directed and then observed. It consists of an Openflow Switch, used to control traffic, several servers, used to run virtual operating systems, and a Data Collector, used to collect all the traffic in the Worm Catcher. With the help of the Openflow Switch and server virtualization, the Worm Catcher is able to provide different network environments to cultivate worms and observe their behavior, running in a physical environment.

The Policy Center and Worm Catcher are two main components of Worm-Hunter and will be described in detail in the next part of this section.

3.3 Policy Center

The Policy Center consists of an Analyzer, a Manager and a Policy Controller. It is responsible for the intrusion detection of external traffic. At the same time, it controls the internal traffic directing to ensure the normal operation of Worm-Hunter.

(1) Analyzer

All the traffic in Worm-Hunter first passes through the Analyzer. The Analyzer is used to detect worms, including known and unknown worms, as shown in **Fig. 3**. The Analyzer's source data are from two sources: Internet traffic and the Core Openflow Controller. We adopt different detection methods for data from different places, that is, original network traffic for signature matching and statistics for anomaly detection. By doing so, the detection efficiency is increased. We set the time-consuming anomaly detection apart from detection using original network traffic so that the original network traffic can pass through the Analyzer with low delay. The none time-sensitive anomaly detection uses statistics that can be gathered from the Openflow Controller in parallel. Alternately, anomaly detection requires comprehensive data, including interaction data for external areas and inner interaction data, which the Openflow Controller is able to provide. All the analysis results are collected, and relevant directing rules are generated according to the results.

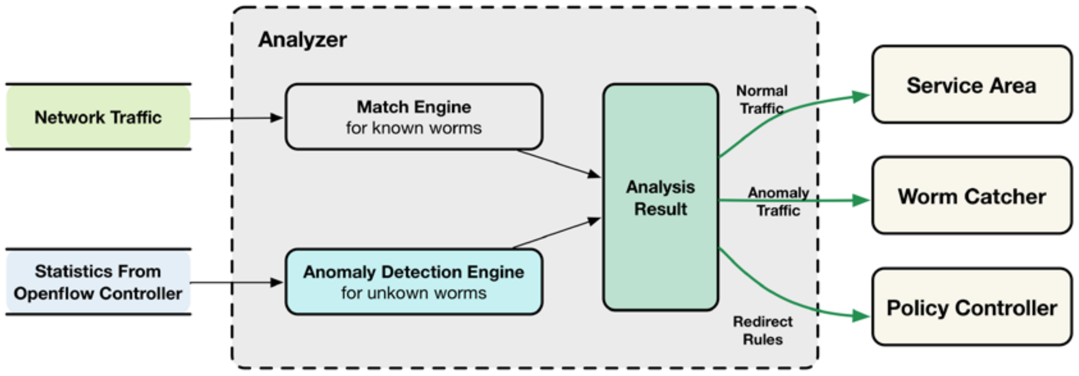


Fig. 3. Modules in the Analyzer

We use signature-matching technology to detect known worms in the Match Engine and use anomaly detection for unknown worms in the Anomaly Detection Engine. The Anomaly Detection Engine uses traffic statistics from the Openflow Controller to detect worms. Once a worm is detected, relevant redirection rules are generated, and the worm will be directed into the Worm Catcher to be cultivated and observed.

- **Match Engine**

In the Match Engine, we use Snort to achieve the signature matching function and apply a signature-generating algorithm to generate worms' signatures. Worm detection based on signature matching can be easily circumvented by polymorphism techniques, by which a program may encode and re-encode itself into different successive byte strings, enabling the production of changing worm payloads [8]. We use the algorithm in Ref. [8] to generate signatures of polymorphic worms, and it can be replaced by other algorithms to researchers' demand.

- **Unknown Worm Engine**

In our worm detection system, the detection of known worms uses worm signatures to perform signature matching. This method has a high detection rate but suffers from the drawback that it is only able to detect known worms whose signatures have been generated. Because attack techniques are constantly renewed, all types of new worms (called 0-day worms) make detection based on known signatures unworkable. Therefore, anomaly detection, which is able to detect unknown attacks, is of vital importance.

Worm-Hunter provides rich traffic statistic information through the Openflow Switch and Controller. The anomaly detection function is achieved by using this information. In our deployed system, we adopt the detection algorithm of Ref. [18], which uses information entropy to detect worms.

- **Analysis result and Flow Table**

We separate anomaly traffic from original network traffic according to the detection results and generate relevant Flow Table entries at the same time. The Flow Table resides in the Openflow Switch and guides network traffic forwarding. Each entry in the Flow Table contains three parts: 1) a packet header that defines the flow, (2) the action, which defines how the packets should be processed, and (3) statistics, which keep track of the number of packets and bytes of each flow and the time since the last packet matched the flow [25].

The Flow Table entries are generated to direct the detected anomaly traffic into the Worm Catcher. For example, from the matching engine, the Analyzer detects that the traffic with the source IP address 6.4.22.109 and destination IP address 10.3.2.11 contains a worm program. Then, the system will conduct two steps as follows: 1) block the detected traffic and send it into the Openflow Switch in the Worm Catcher; and 2) at the same time, generate relevant Flow Table entries. In this example, the relevant Flow Table entries to be generated are listed in **Table 1**. The Flow Table entry is sent to the Core Openflow Switch and used to direct all the traffic with source IP address 6.4.22.109 into port 3 in the switch. Port 3 of the Core Openflow Switch connects to port 1 of the Openflow Switch in the Worm Catcher. By doing so, the anomaly traffic is separated and directed into the Worm Catcher, and thus, all the traffic from the attacker will be handled like this.

Table 1. Relevant Flow Table entries

ID	Switch ID	Packet	Action
1	1	Source IP: 6.4.22.109	Output: 3

It is slightly different when the anomaly detection engine sends an alert. The anomaly detection detects a worm using traffic information of the computer node that has been infected. That means the worm has infected the computer node, so we need to conduct two other steps in addition to generating relevant flow table entries. The first step is to create a virtual copy honeypot of the infected node so that the honeypot catches the worm and we can observe its behavior. The second step is to clean the worm from the computer node. By doing so, the anomaly traffic can be conducted into the Worm Catcher and the worm is reserved.

(2) Manager

The Manager is the system management center for the system administrator. The system administrator can use it to manage the honeynet system and realize functions, including:

- Adding, removing, and editing servers in the Worm Catcher;
- Adding virtual honeypots into the Honeypot Factory;
- Worm cultivation.

We use vSphere (a virtualization management platform designed by VMware) to realize the virtual machines' management and make secondary developments to achieve a control center with easy operation for users. When users make changes to the Worm Catcher, the order will be sent to the Policy Controller, and relevant Flow Table entries will be automatically generated.

We realize the cultivation function through the organization of honeypots and the relevant Flow Table entries. Researchers are able to create a honeynet structure to their demand via the Manager. When a research makes a request to create a honeynet topology, the Manager sends the topology to the Policy Center. The policy program creates the desired honeypots and generates relevant Flow Table entries. The Flow Table entries will then be sent to the Openflow Switch to make the honeynet work. The detailed procedure is described in the following part.

(3) Policy Controller

The Policy Controller's main component is an Openflow Controller. An Openflow Controller is used to control the Flow Table in the Openflow Switch with a security channel. In our design, the Policy Controller connects to Openflow Switches and carries our policy program.

The policy program includes two main modules, the interaction module and topology module, as shown in **Fig. 4**. The interaction module is responsible for the data interaction with external modules, including the Openflow Controller and Analyzer. The topology module is responsible for the work regarding the topology, including generating the topology, maintaining the topology, and generating Flow Table entries.

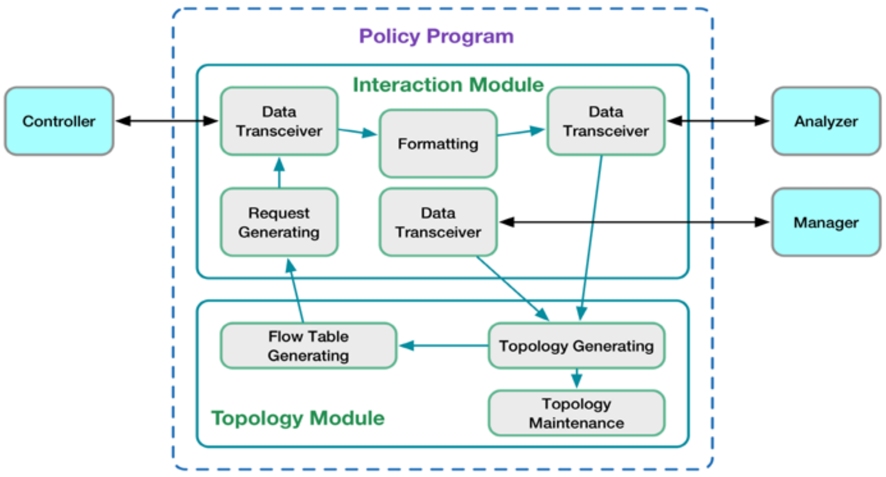


Fig. 4. Architecture of the Policy Program

The **Interaction Module** handles data interaction with the Controller, Analyzer and Manager. It uses the Restful API [25] to ask the Openflow Controller to add, remove and edit Flow Table entries and gather the statistical traffic information. The gathered statistical information is used in the Analyzer. The interaction module uses an SSL tunnel to connect to the Analyzer to send the formatted statistical traffic information and receive the analysis results from the Analyzer. The Manager connects to the Policy Program to realize the honeynet building function. When the system administrator uses the Manager to manage the honeynet system, the order will be sent to the Policy Controller, and then the Topology Module will fulfill the requirements.

After receiving the analysis result or Manager requests, the **Topology Module** generates an appropriate network topology to catch the detected worm according to the results and then generates relevant Flow Table entries according to the topology. At the same time, the topology information is saved into the topology database for topology maintenance.

3.4 Worm Catcher

After worms have been detected, the suspicious traffic is marked and directed into the Worm Catcher with generated directing rules. The Worm Catcher is a dynamic virtual honeynet system that is able to dynamically create multiple honeynets with different network structures. Thus, the Worm Catcher can be used to cultivate and observe worms with different environments in the same physical system at the same time. In addition, the system can gain new variants of worms, which can be used to generate worms' signatures for feedback to the signature database to strengthen the detection.

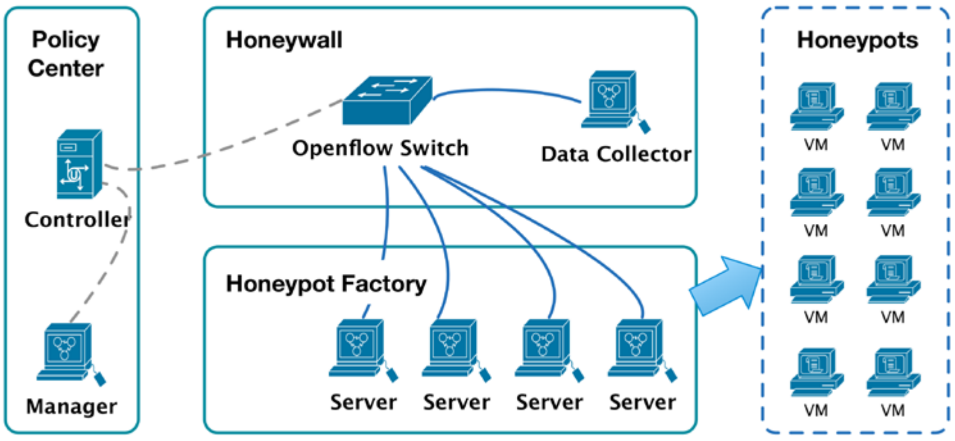


Fig. 5. Overview of the Worm Catcher

The Worm Catcher is based on the SDN/Openflow technique, and an overview is shown in **Fig. 5**. The system can be divided into two parts: the Honeywall and the Honeypot Factory. The Honeywall consists of an Openflow Switch and a Data Collector. All the traffic in the Worm Catcher will first be directed into the Data Collector. The Data Collector records all the traffic and sends it back to the Openflow Switch in the Honeywall. The switch directs the traffic into corresponding honeypots according to the security policies. The Honeypot Factory consists of multiple servers that carry virtual honeypots. These servers are connected by an Openflow Switch that supports the network virtualization feature to achieve a dynamic system.

The function of the Worm Catcher, including the network topology and virtual machines, is managed by the Policy Center, as described in Section 3.3.

(1) Honeywall based on Openflow

Honeywall is responsible for recording and dealing with all the network activities in a honeynet system and can be considered as a transparent gateway that the attacker cannot detect. A traditional honeywall is designed as a server that bridges network cards and achieves a transparent firewall function via iptables [32], which is a type of software firewall. Compared with hardware-based forwarding (switches and routers) in a normal network, software-based forwarding in a traditional honeywall suffers from higher latency. The high latency is fatal because attackers may recognize the existence of the honeynet via the latency. Alternately, a traditional honeywall is built with traditional network equipment whose network protocol is closed. Any change in the network causes complicated manual configuration work and consumes time. Therefore, we design a honeywall based on Openflow that is able to manage and control the traffic simply with easy configuration and low latency.

Recording worm behavior is another target of the Worm Catcher, so we deploy a Data Collector server in the honeywall. The Data Collector is a high-capacity storage server. All the traffic in the Worm Catcher, including external and internal traffic, passes through the Data Collector first and is stored in the Data Collector. The recorded data can be used to observe worms' propagation behavior, extract new variants of worms, track attackers and conduct other security research.

(2) Honeypot Factory

A Honeypot Factory can be recognized as "a factory to create honeypots." We set several servers in the Honeypot Factory area. These servers are physical entities of virtual honeypots on which we use server virtualization technology (virtual machine technology) to create multiple virtual honeypots. One physical resource can be separated into several virtual resources by using virtual machine technology, and thus one physical server can carry multiple virtual honeypots.

We combine network virtualization (provided by SDN) and server virtualization (provided by virtual machines) to design a virtual honeynet system. Firstly, we can easily build virtual machines via configure files or virtual clones of physical servers that contain the demand configuration of honeypots. Then, we connect every honeypot via a network virtualization technique to form different network topologies and build the demanded honeynet service. A flow is recognized in SDN via a 10-tuple (as shown in **Table 2**), while it is recognized in a traditional network via a 5-tuple (protocol, source IP address, source port, destination IP address, destination port) in layer 3 and a MAC address in layer 2. Therefore, in the SDN, not only is the number of items in packets matching more than the number of conditions in the traditional network, but the number of forwarding actions is also greater. Taking packets with the same destination address into account, different forwarding actions can be taken via distinguishing flow with different source addresses and different input ports in the switch, while it is unable to achieve this in the traditional network.

Table 2. 10-tuple in SDN

In Port	Vlan ID	Ethernet			IP			TCP	
		Source Address	Destination Address	Type	Source Address	Destination Address	Type	Src	Dst

The honeypot resources can be easily expanded just by setting more physical servers, and the Policy Center plays the management role in arranging these resources. Researchers do not need to know the physical structure of honeypots, and they can use the honeypot resources easily.

A virtual copy of a real server honeypot is a special type of honeypot that is cloned from a real server using virtualization technology. The virtual copy of real server honeypot, called VCRS-honeypot, has the same basic configuration as the real server, such as the same IP address, the same service, and the same operating system. However, due to the limitation of resources, a VCRS-honeypot does not have the entire content of the real server. For example, it has the same database structure but only partial data. Before attackers steal sensitive data, they only know the data structure rather than the content itself. As a result, we designed the VCRS-honeypot in this fashion so that the attackers are not able to recognize that the server is a honeypot rather than a real server from its content.

(3) Dynamic Virtual Honeynet

Our dynamic virtual honeynet system is based on the honeywall and honeypot factory. Once the relevant directing rules are set, the honeynet system is workable. After an attacker's anomaly traffic is detected and directed into it, the attacker is caught in the Worm Catcher. The procedure is shown in **Fig. 6**.

1) The Openflow Switch receives an external packet;
2) Direct the packet into the Data Collector;
3) After the Data Collector collects the data, the packet will be sent back to the Openflow Switch;
4) The switch searches the Flow Table to find the relevant flow-table entry;
5) The switch changes the packet's destination MAC address and directs it into the matching server according to the found flow-table entry;
6) A related virtual machine receives the packet and sends back a response packet to the switch;
7) The Data Collector collects the response data from the switch and sends it back to the switch;
8) After the Openflow Switch gets the internal response packet, it changes the source MAC address into the real server's MAC address and directs the packet to the external attacker;

The entire procedure is transparent to the attacker, and he believes he is communicating with the target node from the beginning.

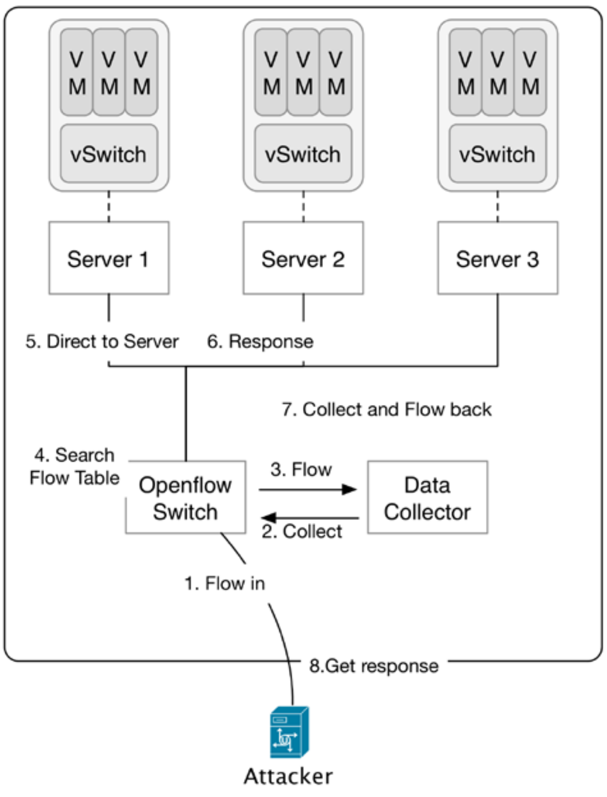


Fig. 6. Honeynet System

(4) Multiple honeynet systems without interference

There is an inevitable problem in honeynet systems built with traditional network devices, including virtual honeynet systems as in Refs. [22, 23]. That is, IP addresses are the unique identities for network traffic in traditional network structures, so it is difficult to differentiate traffic with the same IP address. Then, traffic cannot be directed into different honeypots with the same IP address. Therefore, it is not able to build multiple honeynet systems with the same network topology on one defense platform. However, the fusion of different malicious behaviors in one honeynet system can impact security researchers' analysis.

Worm-Hunter is able to build multiple honeynet systems with the same topology and the same honeypots with the same network addresses at the same time without interference. Compared with the 5-tuple (protocol, source IP address, source port, destination IP address, destination port) in traditional network devices, Openflow uses a 10-tuple to recognize more types of network traffic. In our work, the MAC address and connection port are used to identify traffic and honeypots. We can use the characters in the traffic to identify each honeypot with the same IP address, as shown in **Table 3**.

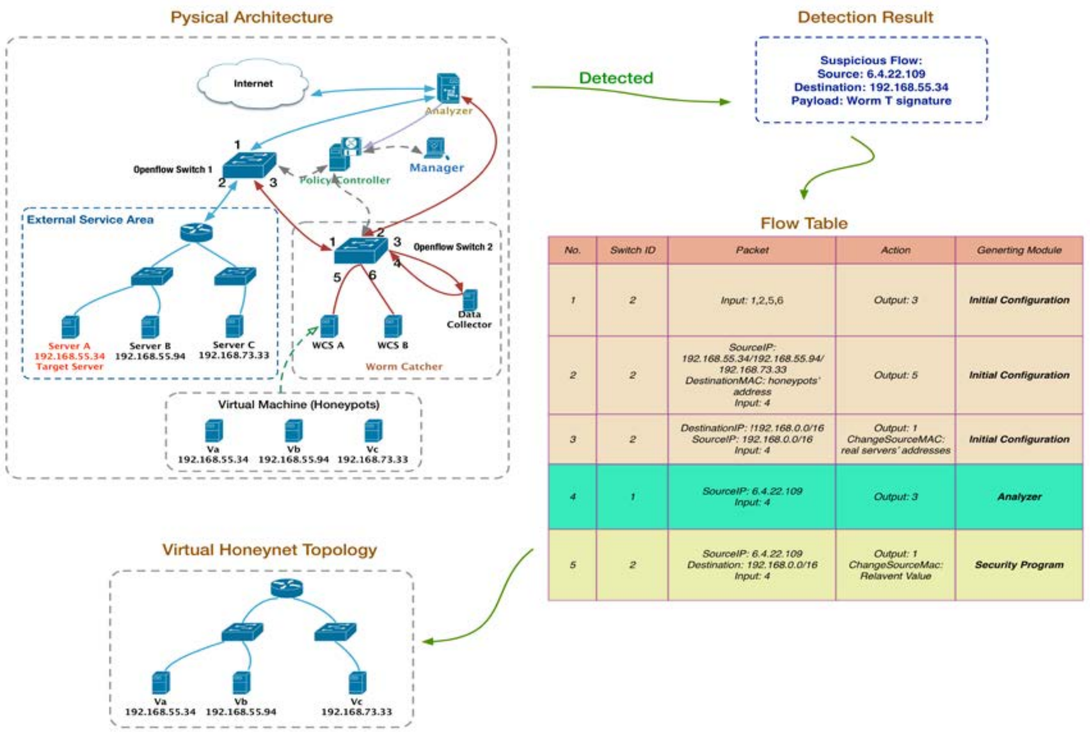
Table 3. Flow Table entry to identify honeypots

Match Item	**Content**
Source IP	Relevant Attacker's IP address
Connecting Switch Port	Connecting Switch Port of Honeypot
Destination MAC	Honeypot's MAC address

According to the connecting port in the Openflow Switch, we classify honeypots with the same IP address into two categories, including connecting the same switch port and connecting different switch ports. We use the MAC address to identify honeypots connecting to the same switch port and the connection port to identify honeypots connecting to different switch ports. By doing so, different honeypots with the same IP address can be identified. Then, we generate relevant Flow Table entries to build different honeynet systems with these honeypots. Therefore, it is able to build different honeynet systems with the same network structure and nodes on the same platform.

3.5 An Example

We use an example to describe the detailed procedure of generating a honeynet according to the analysis result. In the example, we deploy a Worm-Hunter system with two servers carrying external services and three Worm Catcher Servers (WCS) carrying virtual honeypots. WCS A carries three virtual honeypots (Va, Vb, and Vc), which are VCRS-honeypots of Server A, Server B and Server C. Detailed configurations are shown in **Fig. 7**.



Suspicious Flow:
Source: 6.4.22.109
Destination: 192.168.55.34
Payload: Worm T signature

No.	Switch ID	Packet	Action	Generting Module
1	2	Input: 1,2,5,6	Output: 3	**Initial Configuration**
2	2	SourceIP: 192.168.55.34/192.168.55.94/ 192.168.73.33 DestinationMAC: honeypots' address Input: 4	Output: 5	**Initial Configuration**
3	2	DestinationIP: !192.168.0.0/16 SourceIP: 192.168.0.0/16 Input: 4	Output: 1 ChangeSourceMAC: real servers' addresses	**Initial Configuration**
4	1	SourceIP: 6.4.22.109 Input: 4	Output: 3	**Analyzer**
5	2	SourceIP: 6.4.22.109 Destination: 192.168.0.0/16 Input: 4	Output: 1 ChangeSourceMac: Relavent Value	**Security Program**

Fig. 7. Honeynet Generation Procedure

The honeypots are created by the Manager. When the Policy Center receives the Manager's request to create these honeypots, relevant Flow Table entries are generated, which are listed in **Table 4**.

Table 4. Relevant Flow Table entries

ID	Packet	Action
1	Input: 1,2,5,6	Output: 3
2	Input: 4 Destination IP: 192.168.55.34/192.168.55.94/192.168.73.33 Destination MAC: honeypots' addresses Source IP: 192.168.0.0/16	Output: 5
3	Input: 4 Destination IP: !192.168.0.0/16 Source IP: 192.168.0.0/16	Output: 1 Change Source MAC: real servers' addresses

The first entry is used to direct all the traffic into the Data Collector. Our design aims to allow multiple honeynets to be deployed in the same honeynet physical entity, and multiple honeypots with the same IP address are not conflicting, so we use the MAC address to distinguish the honeypots with the same IP address. The second entry is used to make honeypots that are created at this time into a group so that their communications with each other can be led to the group rather than other honeypots with the same IP address. The third entry is used to change the source MAC addresses of all the traffic from VCRS-honeypots to the corresponding real servers' addresses. As a result, the external communication node gets the packet with the real server's address so that it does not know that it is communicating with the honeypot rather than its target.

After the honeynet is set, we assume that the attacker (with IP address 6.4.22.109) tries to implant a worm into Server A (with IP address 192.168.55.34). When the Analyzer detects the attack, it generates a Flow Table entry, as shown in **Table 5**.

Table 5. Relevant Flow Table entries

ID	Packet	Action
1	Source IP: 6.4.22.109 Input: 4	Output: 3

The usage of the entry is to direct anomaly traffic into the Worm Catcher. At the same time, the Analyzer sends the detection result to the security program. Then, the security program generates the relevant Flow Table entry, as shown in **Table 6**.

Table 6. Relevant Flow Table entries

ID	Packet	Action
1	Source IP: 6.4.22.109 Destination: 192.168.0.0/16 Input: 4	Output: 1 Change Source Mac: Relevant Value

The entry leads anomaly traffic (with source IP address 6.4.22.109) into WCS A, which carries Server A's virtual copy (Va). After the Flow Table takes effect, a virtual honeynet topology is created, which can be probed by the external attacker. The entire procedure is automated. The attacker keeps its connection when the traffic is redirected into the Worm Catcher so that it is not able to notice that it is caught.

4. Experimental Results and Analysis

In this section, we deploy the described defense system and simulate the worm injection procedure of the attacker to observe the system performance. We also make a comparison with previous work on honeynet features.

4.1 Procedure of Worm Defense

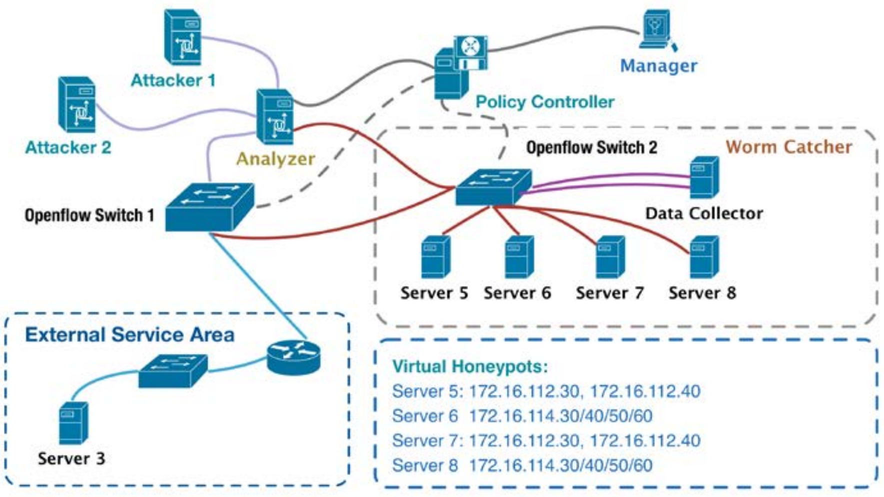


Fig. 8. System Deployment

We design three scenarios to observe the system performance, including how the Table Flow entries are generated and how the traffic is directed. Scenario one is designed for the situation when a worm attempts to attack a server node: the system acts on the defense, catches it in the Worm Catcher, and then cultivates it. The second scenario regards the system's reply to multiple attacks. The third scenario is designed for observing the system's performance in anomaly detection. Finally, we give a security analysis of Worm-Hunter.

(1) Experimental Settings

The system is deployed as in **Fig. 8**. Two attack servers (Attackers 1, 2) are deployed in the external area. One server (Server 3) is deployed in the external service area. Four servers (Servers 4-7) are deployed in the Worm Catcher to carry 12 honeypots.

The parameters of these servers are listed in **Table 7**. Server 5 and Server 7 each carry a VCRS-honeypot of Server 3 and a honeypot with an IP address in Server 3's LAN. Server 6 and Server 8 each carry four honeypots to be used in a cultivation experiment.

Table 7. Experimental parameters

Server ID	**Area**	**OS**	**Carrying**		**Switch Port**
1	External Area	Ubuntu 10.03 10.5.2.3	Attacker 1, Sasser Worm		External Switch 1: port 1
2	External Area	Ubuntu 10.03 10.5.2.10	Attacker 2, Slammer Worm		External Switch 1: port 1
3	Normal Service	Windows XP 172.16.112.30	HTTP Service, SQL Service		Internal Switch 1: port 2
4	Data Collector		Data Collector		Switch 2: port 3, 4
5	Worm Catcher	Windows 10 172.16.120.120	1	172.16.112.30, 00-0C-29-A7-22-38, HTTP, Win XP	Switch 2: port 5
			2	172.16.112.40, 00-0C-29-A7-22-48, HTTP, Win XP	
6	Worm Catcher	Windows 10 172.16.120.130	3	172.16.114.30, 00-0C-29-A7-22-58, HTTP, Ubuntu 10.03	Switch 2: port 6
			4	172.16.114.40, 00-0C-29-A7-22-68, HTTP, Win XP	
			5	172.16.114.50, 00-0C-29-A7-22-78, SQL, Ubuntu 10.03	
			6	172.16.114.60, 00-0C-29-A7-22-88, SQL, Win XP	
7	Worm Catcher	Windows 10 172.16.120.140	7	172.16.112.30, 00-0C-29-A7-25-38, HTTP, Win XP	Switch 2: port 7
			8	172.16.112.40, 00-0C-29-A7-25-48, HTTP, Win XP	
8	Worm Catcher	Windows 10 172.16.120.150	9	172.16.114.30, 00-0C-29-A7-25-58, HTTP, Ubuntu 10.03	Switch 2: port 8
			10	172.16.114.40, 00-0C-29-A7-25-68, HTTP, Win XP	
			11	172.16.114.50, 00-0C-29-A7-25-78, SQL, Ubuntu 10.03	
			12	172.16.114.60, 00-0C-29-A7-25-78, SQL, Win XP	

The Worm Catcher initializes several flow tables via the Policy Center to set up the system, and the Flow Table entries in Openflow Switch 2 are listed in **Table 8**. The usages of the flow tables are:

- "Honeypots in Server x's address" means the MAC addresses of honeypots in Server x (4, 5, 6, and 7 in this experiment);
- All the traffic in the Worm Catcher passes through the Data Collector first;
- Traffic towards the external area is directed to port 1. At the same time, we change the source MAC addresses of Server 3's VCRS-honeypots to Server 3's real address;
- Internal traffic is directed to relevant ports, and its destination MAC address is changed to the virtual copy's address.

Table 8. Relevant Flow Table entries in Openflow Switch 2

ID	Packet	Action
1	Input: 1,2,5,6	Output: 3
2	Input: 4 Destination IP: 172.16.112.0/24 Destination MAC: Honeypots in Server 4' address Source IP: 172.16.0.0/16	Output: 5
3	Input: 4 Destination IP: 172.16.114.0/24 Destination MAC: Honeypots in Server 5' address Source IP: 172.16.0.0/16	Output: 6
4	Input: 4 Destination IP: !172.16.0.0/16 Source IP: 172.16.112.30/16	Output: 1 Change Source MAC: Server 1's address
5	Input: 4 Destination IP: !172.16.0.0/16 Source IP: 172.16.0.0/16	Output: 1

In the initial settings, it is unnecessary for Server 7 and Server 8 to be set with relevant rules to make them work. We only need one set of workable honeypots to catch worms at once to conserve resources. Actually, we do not power on Server 7 and Server 8 in the first scenario. The constraints in the simulation are set as follows:

- Network cards of each honeypot have different MAC addresses.
- Each honeypot's ARP table has other nodes' MAC addresses so that it is unnecessary to send an ARP request package before connecting.

(2) Scenario One: the Sasser Worm attacks the system

In this scenario, an attacker attempts to attack an Internet server via his worm. In the experiment, the desired result is that Worm-Hunter detects the worm and directs the attack traffic into the Worm Catcher so that the researcher can cultivate the worm and observe its behavior.

We use the Sasser Worm to attack Server 3; when a server is infected by the Sasser Worm, it will scan for the network nodes with the Windows operating system in the same LAN to spread itself. Meanwhile, the signature of Sasser Worm is saved in the Analyzer's signature database. When the attack traffic (from 10.5.2.3) arrives at the Analyzer, the Analyzer sends the alert that the Sasser Worm is being detected to the Policy Center.

Then, the Policy Center generates a new flow table to redirect the attacker's traffic into the VCRS-honeypot in Server 4, which the attacker will not notice, as shown in **Table 9**. The Flow Table entry changes the destination MAC address of the traffic from the attacker towards node (172.16.112.30) to the honeypot's MAC address so that the honeypot can receive and handle the traffic.

Table 9. Relevant Flow Table Entry in Scenario 1

ID	Packet	Action
6	Source IP: 10.5.2.3 Destination IP: 172.16.112.30	Output: 1 ChangeDesMAC: 00:02:ef:c8

Then, the worm is caught and cultivated. The cultivation environment is set such that there is a neighbor node in the infected node's LAN area and two nodes with different network segments. After the worm successfully infects the honeypot, it starts to spread itself. It scans live nodes in the LAN and searches for nodes that can be infected. The worm spreading trace is collected by the Data Collector. The honeypots with IP 172.16.112.40 and IP segment 172.16.114.0/24 receive the scan packages, and then Windows servers are infected by the Sasser Worm. As a result, the worm is successfully caught in the Worm Catcher and the interaction data are recorded. Security researchers can observe the worm's behavior and conduct further analysis and research.

(3) Scenario Two: the Slammer Worm attacks the system after the Sasser Worm has been caught

In this scenario, we extend the experiment of scenario 1. We have caught the Sasser Worm in the honeypots and cultivated it as a presupposition. The system security manager finds that the Sasser Worm has been caught, and Server 5 and Server 6 are used for the Sasser Worm. Then, he powers on Server 7 and Server 8, which contain a VCRS-honeypot of Server 3 and honeypots for the cultivation environment. After a while, a new attacker on Server 2 wants to attack Server 3 using the Slammer Worm. The Slammer Worm is a Windows worm that only searches for nodes with SQL Server on the Windows operating system. Our experiment is based on the above description.

When the new honeypots are powered on, new flow tables, as shown in **Table 10**, are added to Openflow Switch 2 by the Policy Center. Changes of the Flow Table ensure that the honeypots on Sever 7 and Server 8 are in the same group. The previously captured Sasser Worm is still running for cultivation in the Worm Catcher, while Server 3's new VCRS-honeypot in Server 7 is working via the new Flow Table.

Table 10. New Relevant Flow Table entries in Scenario 2

ID	Packet	Action
1	Input: 1,2,5,6,**7,8**	Output: 3

7	Input: 4 Destination IP: 172.16.112.0/24 Honeypots in Server 7’ address Source IP: 172.16.0.0/16	Output: 7
8	Input: 4 Destination IP: 172.16.114.0/24 Honeypots in Server 8’ address Source IP: 172.16.0.0/16	Output: 8

As described in Section 4.1, we capture the Slammer Worm’s attack traffic (from 10.2.5.10). Then, the Analyzer sends an alert to the Policy Center, and the Policy Center performs corresponding measures to redirect the attacker’s traffic into the Worm Catcher. New Flow Table entries are shown in **Table 11**.

Table 11. New Relevant Flow Table entry in Scenario 2

ID	Packet	Action
9	Source IP: 10.5.2.10 Destination IP: 172.16.112.30	Output: 1 ChangeDesMac: 00:02:ff:a0

This entry connects the attacker (10.5.2.10) with the honeypot on Server 7. Beyond the Flow Table, attacker (10.5.2.10) is caught and can be cultivated as in the first scenario. Neighboring nodes with the Windows operating system are attacked by the Slammer Worm in the VCRS-honeypot of Server 6.

In this scenario, we catch the Sasser Worm and Slammer Worm in two VCRS-honeypots at the same time. The two honeynet systems are mutually independent, but honeypots in them have the same IP addresses. In the experiment, we find that the Sasser Worm is still working to scan other nodes when the Slammer Worm is being caught. The result demonstrates that Worm-Hunter is able to catch and cultivate multiple worms separately in the same physical entity. In this manner, researchers can realize multiple security research studies in separate environments to avoid the data perturbation between different worms in an easy way. The result also shows that Worm-Hunter can easily expand by adding new servers and honeypots. Meanwhile, the system can automatically generate relevant directing rules to make the system work.

(4) Scenario Three: Anomaly detection

The first and second scenarios are experiments based on matching detection, and in this scenario, we design an experiment based on anomaly detection to observe Worm-Hunter’s performance. We use the same experimental settings of the first scenario and Ref. [18]’s anomaly detection algorithm to detect the anomaly.

In the experiment, we infect Server 3 manually with the Sasser Worm. After the worm runs, the infected node (Server 3) starts to create lots of scans. Then, the Analyzer detects the anomaly and sends an alert that Server 3 is anomalous. After receiving the alert, the Policy Center copies Server 3 to create a new VCRS-honeypot of Server 3 using the virtualization API and then replaces the previous VCRS-honeypot of Server 3 in Server 5. At the same time, the security manager observes the alert and analysis of Server 3 to clean the worm. As a result, the worm is moved to the Worm Catcher, and all the traffic from the new VCRS-honeypot of Server 3 with its external area is directed into the Worm Catcher. The next procedure is the same as in the first and second scenario.

(5) Security Analysis

In all three scenarios, attackers are not able to notice that their target nodes obtain changed to honeypots. In the traffic directing process, all the traffic signatures from honeypots, such as IP addresses and MAC addresses, are the same as the real service nodes. Meanwhile, honeypots have the full data structure and partial content of real service nodes. Before sensitive information is stolen, attackers know only the data structure, not the full content, so they are unable to check whether they are caught via the content in the honeypots.

Nodes, connected with the Openflow Switch, run the same network protocol as a traditional network, so their traffic has the same content pattern and signature as those in a traditional network. For example, the traffic in both SDN and a traditional network has IP addresses and MAC addresses. Attackers are unable to recognize that the network environment is an SDN architecture. Even if attackers use a traditional network device attack technique to hack a network device in Worm-Hunter, the Openflow Switch is not broken since it has a different structure from that of a traditional device. Therefore, the security of Worm-Hunter is also not in danger.

When attackers hack a honeypot, it must take into account whether the attackers can use the honeypot to endanger the honeynet system. After attackers hack the honeypots, they can control the honeypots to do what they want. However, since all the traffic in the Worm Catcher is controlled by the Flow Table, the traffic from victim honeypots is isolated in the honeynet system that they belong to. Therefore, the attackers cannot cause further damage from the victim honeypots or endanger other honeynet systems in Worm-Hunter.

In conclusion, attackers can neither become aware that they are caught in Worm-Hunter nor damage Worm-Hunter from the honeypots.

4.2 Measurement

We compare Worm-Hunter with three common honeynet systems, i.e., Honeyd, Gen III [23] and the virtual honeynet in Ref. [22]. These three honeynet systems represent three typical types of honeynet systems.

- Honeyd is a low-interaction honeynet system.
- Gen III is a traditional virtual honeynet system. It builds a virtual honeynet system in a host machine, which includes a software honeywall, virtual switches, and multiple virtual honeypots.
- Ref. [22] promotes a dynamic virtual honeynet system using a virtualization technique. It uses the virtualization technique to create a honeypot farm collecting multiple virtual honeypots. By doing so, it increases the utilization rate of physical resources. However, it still uses traditional network devices to direct external network traffic and has many restrictions with regard to dynamically building a honeynet.

The criteria of measurement are based on Ref. [33], and we also add some new feature criteria. The criteria are Portability, Setup, Cost, Flexibility, Detectability, Adaptability and Overlay Honeynet.

Portability is the criterion of whether the honeynet can be moved. Honeyd and Gen III honeynet system are deployed in certain physical servers, so they are not portable. The honeynet system in Ref. [22] and Worm-Hunter are composed of virtual honeypots that are able to move to other physical servers easily, thus ensuring portability.

Setup measures the difficulty to set up a new honeynet system. Honeyd can be built easily by simply adding action scripts, and the scale of the honeynet can be expanded by adding more scripts. Gen III needs to deploy servers individually. Every component must be set up and configured to make the honeynet system workable. However, Ref. [22] and Worm-Hunter use a virtualization technique to easily build a virtual honeynet regardless of the differences between physical servers. The setup difference between Ref. [22] and Worm-Hunter lies in the deployment of network traffic devices. Ref. [22] must make extra effort to configure the router to lead the external traffic into the honeynet. In comparison, a policy program is contained in Worm-Hunter to automatically manage the complex traffic directing strategies.

Cost measures the hardware cost. Due to the simple features of Honeyd, Honeyd requires simple physical servers and low costs. This is a common advantage of low-interaction honeynet systems. In Gen III, each honeynet system is deployed in one physical machine that contains a software honeywall, multiple virtual switches, and multiple virtual honeypots. The software honeywall and virtual switches consume large amounts of computing resources. Ref. [22] and Worm-Hunter isolate virtual honeypots and generate a special area to support honeypot resources (the honeypot factory in Worm-Hunter). They fully utilize physical resources to build multiple honeypots in each physical server. Therefore, the hardware cost is reduced. **Fig. 9** shows the physical machine costs of the three virtual honeynet systems at different honeypot scales. Due to the existence of the honeywall and virtual switches in Gen III, a physical machine (with 4 Intel(R) Core(TM) i5 CPUs and 8 GB of RAM) is only able to carry 2 virtual honeypots. On the contrary, a physical machine with the same configuration can carry 4 virtual honeypots. Ref. [22] requires an extra physical machine to carry an agent to manage the traffic, and Worm-hunter requires an extra physical machine to be the Openflow Controller to manage the traffic directing strategies. As shown in **Fig. 9**, the 3 honeynet systems require the same number of physical machines at the 4-honeypot scale. In regard to larger honeypot scales, Gen III requires more hardware resources than Ref. [22] and Worm-Hunter. The results show the low cost of Ref. [22] and Worm-Hunter.

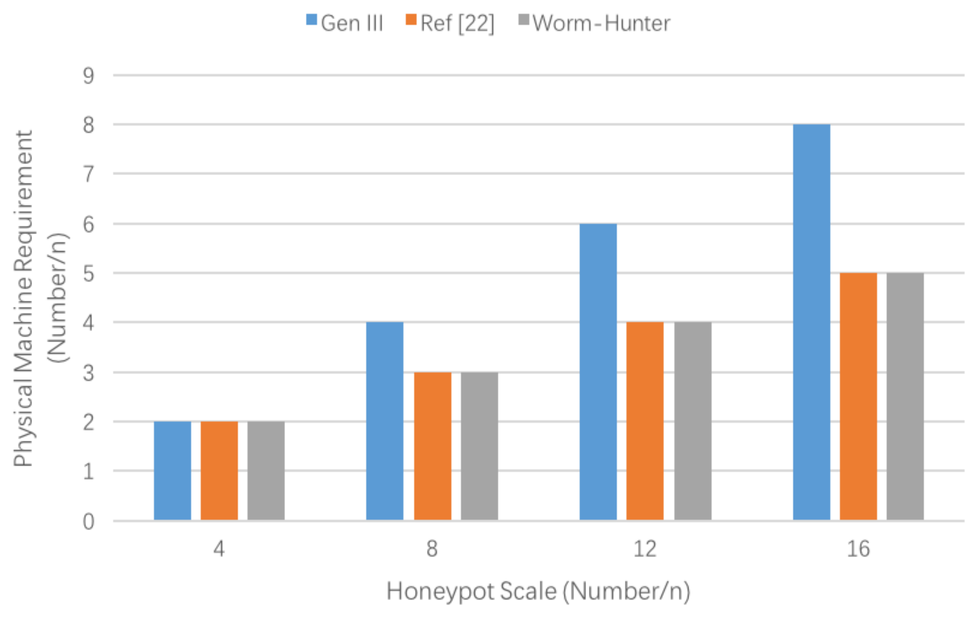


Fig. 9. The Number of Physical Machines Required

Flexibility measures the ability to form different types of honeypots. Honeyd can only form honeypots according to the action scripts. Interactions beyond the action scripts cannot be achieved in Honeyd. The three other honeynet systems are able to form honeypots within any operation systems and configurations and have powerful attractions.

Detectability measures the difficulty faced by attackers in recognizing the honeynet system. Honeyd is a low-interaction honeynet system. An attacker may recognize the honeynet when he finds that he cannot make further interactions with the target host. The other three honeynet systems afford full operation and service for attackers such that attackers cannot identify the honeynet by interacting with it. Nevertheless, Gen III uses a kernel monitor technique to observe the attacker's behavior. The attacker is able to make a reasonable effort to find the kernel monitor task after he hacks the honeypot. Ref. [22] and Worm-Hunter monitor the anomaly traffic from the network hardware and do not have a software task in the honeypot, so it is difficult for attackers to recognize the honeynet system. However, advanced attackers may recognize Ref. [22] due to the network delay. Ref. [22] uses a software honeywall to direct network traffic, which will cause a higher network delay than a hardware honeywall in traffic directing. **Fig. 10** shows the average network delay of different directing devices, including an Openflow-based honeywall and a software honeywall. We calculate the average delay in a 5-minute ping experiment. The results show that Worm-Hunter has a lower delay in traffic directing.

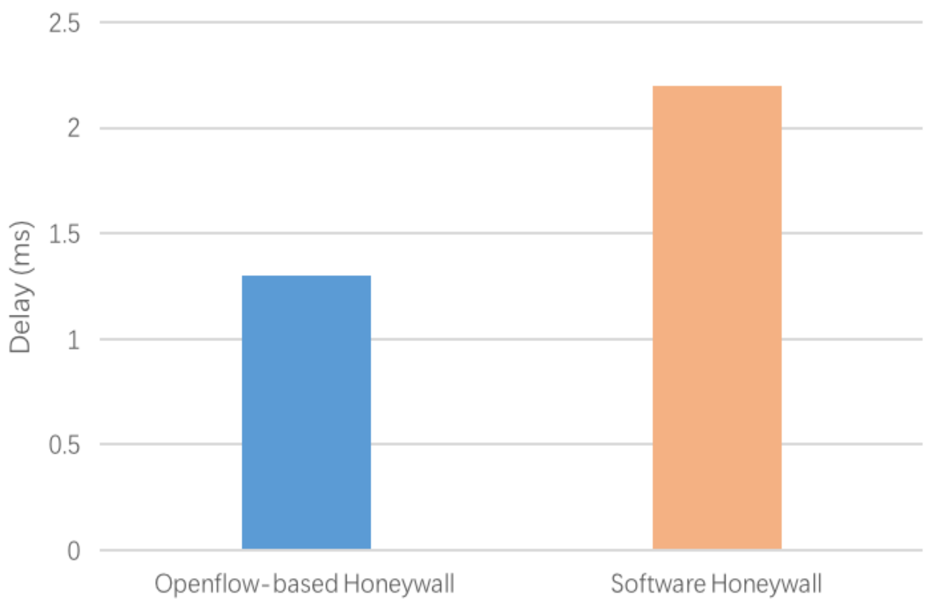


Fig. 10. Comparison of the Time Cost

Adaptability is a measure of what types of honeynets can be configured dynamically [33]. Ref. [22] and Worm-Hunter are able to dynamically build different honeynet systems by using a virtualization technique, while the other two systems are not able to do so. However, Worm-Hunter has stronger adaptability because it is able to build multiple honeynet systems with the same network structure at the same time.

Overlay Honeynet is the feature to build honeynet systems with the same topology and the same honeypots with the same network addresses on one physical platform. Our work is able to achieve this feature, while the previous work does not provide this capability. Relevant experiments regarding overlay honeynet are described in Section 4.1.

According to the comparisons above, we summarize the results in **Table 12**.

Table 12. Comparison of honeynet systems

Item	**Honeyd**	**Gen III**	**Ref. [22]**	**Worm-Hunter**
Portable	No	No	Yes	Yes
Setup	Easy	Involved	Easy	Easy
Cost	Low	High	Low	Low
Flexible	Limited	Very	Very	Very
Detectability	Easy	Reasonable Effort	Hard	Hard
Adaptability	No	No	Yes	Yes
Overlay Honeynet	No	No	No	Yes

5. Conclusion

This paper presents a worm defense system that can be used to dynamically deploy a honeynet within all types of network structures in an easy manner. It can be used by researchers to design different network scenarios to meet their demand for security experiments that can be performed on one physical platform, while all of the experiments can be performed without interference. Compared to existing worm defense solutions, our main contributions are as follows. Worm-Hunter combines SDN technology with a server virtualization technique to realize a dynamic virtual honeynet system. This system can quickly deploy honeynet systems with different network structures, and all of the honeynet systems are non-interfering. It is noteworthy that honeypots in different honeynet systems can have the same network address and not lead to conflicts. Worm-Hunter provides a framework for worm detection, and all of the parts of the system can be replaced with researchers' own methods. For example, researchers are able to use their own detection algorithms in the Analyzer. We will perform further work on signature matching to promote detection efficiency and traffic throughput. An applicable anomaly detection method for SDN is also a component of our future research plan.

References

[1] Weaver, Nicholas, Vern Paxson, Stuart Staniford, and Robert Cunningham, "A taxonomy of computer worms," in *Proc. of the 2003 ACM workshop on Rapid malcode*, pp. 11-18, 2003. Article (CrossRef Link)

[2] Moore, David, and Colleen Shannon, "Code-Red: a case study on the spread and victims of an Internet worm," in *Proc. of the 2nd ACM SIGCOMM Workshop on Internet measurement*, 2002. Article (CrossRef Link)

[3] Moore, David, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver, "Inside the slammer worm," *IEEE security and privacy*, vol. 1, no. 4, pp. 33-39, 2003. Article (CrossRef Link)

[4] Zou, Cliff Changchun, Weibo Gong, and Don Towsley, "Code red worm propagation modeling and analysis," in *Proc. of the 9th ACM conference on Computer and communications security*, pp. 138-147, 2002. Article (CrossRef Link)

[5] Roesch, Martin, "Snort: Lightweight Intrusion Detection for Networks," *LISA*, vol. 99, no. 1, pp. 229-238. 1999. Article (CrossRef Link)

[6] Spitzner, Lance, "The honeynet project: Trapping the hackers," *IEEE Security & Privacy*, vol. 1, no. 2, pp. 15-23, 2003. Article (CrossRef Link)

[7] Paxson, Vern, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23, pp. 2435-2463, 1999. Article (CrossRef Link)

[8] Kreibich, Christian, and Jon Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," *ACM SIGCOMM computer communication review*, vol. 34, no. 1, pp. 51-56, 2004. Article (CrossRef Link)

[9] Kim, Hyang-Ah, and Brad Karp, "Autograph: Toward Automated, Distributed Worm Signature Detection," *USENIX security symposium*, vol. 286, 2004. Article (CrossRef Link)

[10] Singh, Sumeet, Cristian Estan, George Varghese, and Stefan Savage, "Automated Worm Fingerprinting," *OSDI*, vol. 4, pp. 4-4, 2004. Article (CrossRef Link)

[11] Newsome, James, Brad Karp, and Dawn Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proc. of 2005 IEEE Symposium on Security and Privacy (S&P'05)*, pp. 226-241, 2005. Article (CrossRef Link)

[12] Kong, Deguang, Yoon-Chan Jhi, Tao Gong, Sencun Zhu, Peng Liu, and Hongsheng Xi, "SAS: semantics aware signature generation for polymorphic worm detection," *International Journal of Information Security*, vol. 10, no. 5, pp. 269-283, 2011. Article (CrossRef Link)

[13] Kruegel, Christopher, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna, "Polymorphic worm detection using structural information of executables," in *Proc. of International Workshop on Recent Advances in Intrusion Detection*, pp. 207-226, 2005. Article (CrossRef Link)

[14] Bayoglu, Burak, and Ibrahim Sogukpinar, "Polymorphic worm detection using token-pair signatures," in *Proc. of the 4th international workshop on Security, privacy and trust in pervasive and ubiquitous computing*, pp. 7-12, 2008. Article (CrossRef Link)

[15] Nychis, George, Vyas Sekar, David G. Andersen, Hyong Kim, and Hui Zhang, "An empirical evaluation of entropy-based traffic anomaly detection," in *Proc. of the 8th ACM SIGCOMM conference on Internet measurement*, pp. 151-156, 2008. Article (CrossRef Link)

[16] Brauckhoff, Daniela, Bernhard Tellenbach, Arno Wagner, Martin May, and Anukool Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proc. of the 6th ACM SIGCOMM conference on Internet measurement*, pp. 159-164, 2006. Article (CrossRef Link)

[17] Wagner, Arno, and Bernhard Plattner, "Entropy based worm and anomaly detection in fast IP networks," in *Proc. of 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, pp. 172-177, 2005. Article (CrossRef Link)

[18] Gu, Yu, Andrew McCallum, and Don Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proc. of the 5th ACM SIGCOMM conference on Internet Measurement*, pp. 32-32, 2005. Article (CrossRef Link)

[19] Honeynet Project, "Know Your Enemy: Honeynets," *http://www.symantec.com/connect/articles/know-your-enemy-honeynets*, 2001. Article (CrossRef Link)

[20] Provos, Niels, "Honeyd-a virtual honeypot daemon," in *Proc. of 10th DFN-CERT Workshop, Hamburg, Germany*, vol. 2, p. 4, 2003. Article (CrossRef Link)

[21] Nance, Kara, Brian Hay, and Matt Bishop, "Virtual machine introspection," *IEEE Computer Society*, vol. 6, no. 05, pp. 32-37, 2008. Article (CrossRef Link)

[22] Yang, Hwan-Seok, "A study on attack information collection using virtualization technology," *Multimedia Tools and Applications*, vol. 74, no. 20, pp. 8791-8799, 2015. Article (CrossRef Link)

[23] Abbasi, Fahim H., and R. J. Harris, "Experiences with a Generation III virtual Honeynet," in *Proc. of Telecommunication Networks and Applications Conference (ATNAC)*, 2009 Australasian, pp. 1-6, 2009. Article (CrossRef Link)
[24] Spitzner, Lanz, "Know Your Enemy: Honeywall CDROM Roo 3rd Generation Technology," 2005. Article (CrossRef Link)
[25] McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008. Article (CrossRef Link)
[26] Shin, Seungwon, Vinod Yegneswaran, Phillip Porras, and Guofei Gu, "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," in *Proc. of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 413-424, 2013. Article (CrossRef Link)
[27] Big Switch Network. Article (CrossRef Link)
[28] Juniper. Article (CrossRef Link)
[29] Huawei. Article (CrossRef Link)
[30] Wang, Bing, Yao Zheng, Wenjing Lou, and Y. Thomas Hou, "DDoS attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, vol. 81, pp. 308-319, 2015. Article (CrossRef Link)
[31] Shin, Seungwon, and Guofei Gu, "Attacking software-defined networks: A first feasibility study," in *Proc. of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 165-166, 2013. Article (CrossRef Link)
[32] Kreutz, Diego, Fernando Ramos, and Paulo Verissimo, "Towards secure and dependable software-defined networks," in *Proc. of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 55-60, 2013. Article (CrossRef Link)
[33] Fan, Wenjun, David Fernández, and Zhihui Du, " Adaptive and flexible virtual honeynet," in *Proc. of International Conference on Mobile, Secure and Programmable Networking*, pp. 1-17, 2015. Article (CrossRef Link)

Yixun Hu, is currently a Ph.D. candidate at the School of Computer, Beijing University of Posts and Telecommunications, Beijing, China. He received his B.E. degree in information security from Beijing University of Posts and Telecommunications in 2010. His research interests include network security, intrusion detection, honeynet system, and software-defined networking. The corresponding author. Email: hyx.bupt@gmail.com

Kangfeng Zheng, is a vice-professor, Ph.D supervisor at the School of Computer, Beijing University of Posts and Telecommunications, China. He received his Ph.D. in 2006 from School of Information Engineering in Beijing University of Posts and Telecommunications. His research interests are network security, intrusion detection, malicious code analysis, honeynet system, cyber-attack modeling and Advanced Persistent Threat.

Xu Wang received the B.S. degree from Beijing Information Science and Technology University, Beijing, China, in 2010. He is currently pursuing the Ph.D. degree with the Information Security Center, Beijing University of Posts and Telecommunications, Beijing, China. He visited CSIRO, Australia, in 2014. His main research interests lie in graph theory, Markov theory, intrusion detection, and cyber security.

Yixian Yang received the M.S. degree in applied mathematics and the Ph.D. degree in electronics and communication systems from the Beijing University of Posts and Telecommunications, Beijing, China, in 1986 and 1988, respectively. He is the Managing Director of Information Security Center, Beijing University of Posts and Telecommunications. He has co-authored 300 scientific papers. He holds 50 patents. His current research interests include network security, information security, and coding theory.