Enabling an Anatomic View to Investigate Honeypot Systems: A Survey

Wenjun Fan, Zhihui Du, *Senior Member, IEEE*, David Fernández, and Víctor A. Villagrá

***Abstract*—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 accesses. It is also useful for investigating the behavior of attackers, and in particular, unknown attacks. For the past 17 years plenty of effort has been invested in the research and development of honeypot techniques, and they have evolved to be an increasingly powerful means of defending against the creations of the blackhat community. In this paper, by studying a wide set of honeypots, the two essential elements of honeypots—the decoy and the captor—are captured and presented, together with two abstract organizational forms—independent and cooperative—where these two elements can be integrated. A novel decoy and captor (D-C) based taxonomy is proposed for the purpose of studying and classifying the various honeypot techniques. An extensive set of independent and cooperative honeypot projects and research that cover these techniques is surveyed under the taxonomy framework. Furthermore, two subsets of features from the taxonomy are identified, which can greatly influence the honeypot performances. These two subsets of features are applied to a number of typical independent and cooperative honeypots separately in order to validate the taxonomy and predict the honeypot development trends.**

***Index Terms*—Computer security, honeypots, intrusion detection, network security, virtualization.**

I. Introduction

THE new domain of cyberspace is so pervasive that the US Department of Defense has put cyberspace on a par with land, sea, and air as a war-fighting domain [1]. Systems in cyberspace are constantly faced with cyber threats every day. In 2015, Symantec discovered 54 zero-day vulnerabilities, a 125% increase from the year before [2]. Since cyber threats cannot be eliminated completely, the strategy to securing cyberspace is to remove as many vulnerabilities as possible before they can be exploited [3]. A honeypot is a vital security facility aimed at sacrificing its resource to investigate unauthorized accesses in order to discover potential vulnerabilities in operational systems, and reduce the risks. Due to its unique design and application features, it can help to address the deficiencies of other existing security methods.

Firewalls are often deployed around the perimeter of an organization in order to block unauthorized access by filtering certain ports [4] and content, but they do little to evaluate the traffic. They can block all accesses to a certain service in order to prevent malevolent traffic, but this also blocks any benevolent traffic that wants to access the service. Conversely, honeypots are aimed at opening ports in order to capture as many attacks as possible for subsequent data analysis. An intrusion detection system (IDS) is used to evaluate the traffic and detect any inappropriate, incorrect, and anomalous activity. However, IDSs often have the “false alert problem,” i.e., signature (rule-based) IDSs often generate false negative alerts, whilst anomaly-based IDSs generate false positive alerts. Compared to an IDS, a honeypot has the big advantage that it never generates false alerts, because any observed traffic to it is suspicious since there is no production service running on the honeypot. Hence, an integration of a honeypot with an IDS can largely reduce the number of false alerts [5].

An intrusion prevention system (IPS), comprising a firewall plus an IDS, can evaluate the traffic and block malicious data. It acts as a shield against attacks, but it is not able to distinguish whether an application-layer request is normal or not. This drawback could potentially result in attacks permeating the shield without being detected, e.g., a social engineering attacker may gain sensitive information by using a compromised legitimate username and password [6]. If an IPS integrates with a honeypot, the whole system can then capture all attacking activities regardless of whether they are performed by inside or outside adversaries. Also, the data captured by honeypots can be used to create countermeasures, e.g., the automated intrusion response systems often uses honeypots as the data capture infrastructure [7].

Honeypots are often used to investigate currently unknown attacks [5], [8]. The Blackhat community is intelligent enough to create new-unknown threats. A good way to investigate new threats is to capture the malicious activity step-by-step as it compromises a system. Honeypots therefore can add value to research by providing a sacrificial system to be attacked. Furthermore, it is worth observing what the adversaries do in the compromised system, such as communicating with other attackers and uploading new rootkits. Also, honeypots can effectively

Manuscript received November 4, 2016; revised April 23, 2017, June 20, 2017, and September 3, 2017; accepted October 4, 2017. This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB1000602 and Grant 2017YFB0701501, in part by the MOE Research Center for Online Education Foundation under Grant 2016ZD302, and in part by the National Natural Science Foundation of China under Grant 61440057 and Grant 61363019. *(Corresponding author: Wenjun Fan.)*

W. Fan is with the School of Computing, University of Kent, Canterbury CT2 7NZ, U.K. (e-mail: W.Fan@kent.ac.uk).

Z. Du is with the Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: duzh@tsinghua.edu.cn).

D. Fernández and V. A. Villagrá are with the Department of Telematics Engineering, Universidad Politécnica de Madrid, Madrid 28040, Spain (e-mail: david@dit.upm.es; villagra@dit.upm.es).

Digital Object Identifier 10.1109/JSYST.2017.2762161

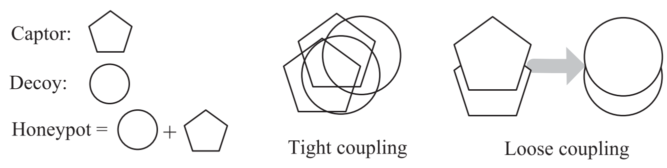

Fig. 1. Honeypot Anatomy: core elements and their organizations.

capture automated attacks [9], [10]. Due to the fact that automated attacks often target the entire network, honeypots can quickly capture them for investigation.

Hence, according to different security requirements, a variety of honeypots have been proposed, i.e., there is not only dedicated honeypot software [11], but also complex cooperative honeypot systems, such as honeynets [12] and hybrid systems [9], [10], [13], etc. However, there is a lack of a distinct method that can quickly catch the key points of the various honeypots and that can discover new insights, and advance research and development in this area. Our paper proposes to address these problems. The main contributions of this paper can be summarized as follows.

1) Two essential elements (decoy and captor) of a honeypot are captured, and how these elements are organized is described, which provides a general view for analyzing honeypots.
2) A decoy and captor (D-C) based taxonomy is proposed to study different aspects of honeypot technology.
3) Several development trends are identified by comparing honeypots according to the taxonomy.

The remainder of this paper is organized as follows: Section II defines the core concepts and terminology; Section III proposes a way to investigate different honeypot technologies by providing a novel D-C-based taxonomy; Section IV surveys a number of honeypots based on the taxonomy in order to analyze their development; Section V makes a conclusion.

II. Honeypot Anatomy

The first idea for honeypots comes from the book titled "The Cuckoo's Egg" [14] that described a series of events about tracking a hacker. The second material about honeypots was reported in a whitepaper [15]. The definition of honeypot was proposed by Spitzner in 2003 [16]: "A honeypot is an information system resource whose value lies in the unauthorized or illicit use of that resource." However, this definition describes a honeypot based on its application value, rather than what it is. We therefore provide a clearer definition of what a honeypot is (see Fig. 1).

A *honeypot* is an information system that includes two essential elements, decoys and captors. It aims at using its information resources to attract unauthorized and illicit access with the purpose of security investigation. The *decoy* can be any kind of information system resource, and the *captor* (security program) facilitates the security-related functions, i.e., attack monitoring, prevention, detection, response, and profiling. Besides, the captors should be running in stealth mode to avoid detection.

Among the existing honeypot projects and honeypot research work, the terminology is not consistent. Some refer to decoys as honeypots. For example, a decoy can be a fake digital entity. The terminology for a digital entity acting as a decoy is *honeytoken* [16]. In the book "The Cuckoo's Egg," Stoll deployed honeytokens, i.e., digital files, with security programs (here we call them captors) to track a German hacker. Thus, the honeytoken is a decoy, but Stoll's system is a honeypot system. Our definition clarifies that a vulnerable system without any captor is only a decoy rather than a honeypot. Unless it is equipped with a captor then we do not call it a honeypot.

The organization of the two essential elements can be roughly categorized according to their degree of coupling: loose and tight (see Fig. 1). Coupling refers to the amount 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 other separate ones. It enables components to remain completely autonomous and unaware of each other while still interfacing with each other. In contrast, tight coupling is when a group of components are highly dependent on one another, or are built into the same unit to perform the task. An *independent honeypot* refers to one using tight coupling, and a *cooperative honeypot* indicates one using loose coupling. Nawrocki *et al.* [11] surveyed a number of honeypots that are independent honeypots, while complex systems like the honeynets [12] and hybrid systems [9], [10], [13] are cooperative honeypots. This paper uses the term "honeypot" and "honeypot system" interchangeably.

III. Review With D-C-Based Taxonomy

This section proposes a novel D-C-based taxonomy, as Fig. 2 shows. The classification scheme is divided into two categories. The first category includes the features of a decoy, and the second one consists of the functions of a captor. The D-C-based taxonomy is used as a basic conceptual model in order to investigate honeypot technology. Under this taxonomy framework, we review typical honeypots and specific honeypot-related techniques. The terminology in this paper is described in a technical way, which can make their definitions distinct and easy to understand.

A. Features of Decoy

The decoy aims to capture data by being attacked. There are several primitive characteristics that comprise the design of a decoy.

1) Fidelity: It denotes the degree of exactness of an information system resource that the decoy provides to the attacker. It classifies the interaction into three levels: low, medium, high (see Fig. 3).

Low-interaction honeypot (LIH) only provides a little interaction to adversaries. The LIH decoy is also known by another name: facade. A traditional LIH, e.g., Honeyd [17], is a program that emulates the protocols of an operating system (OS), but with a limited subset of the full functionality. Consequently, an adversary is not able to compromise a LIH because there are only the fingerprints of OS functions instead of the real functionality. Also, a LIH can provide a captor to monitor the facade in order to capture the network activity.

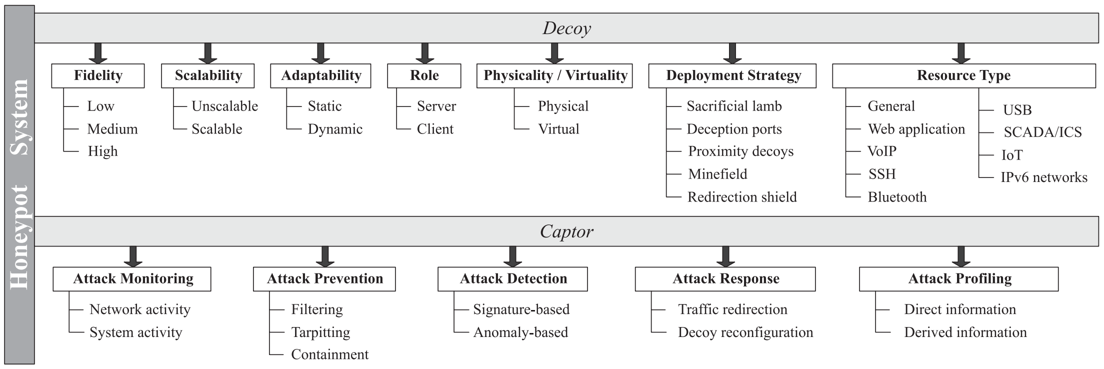

Fig. 2. D-C-based Taxonomy of honeypot systems.

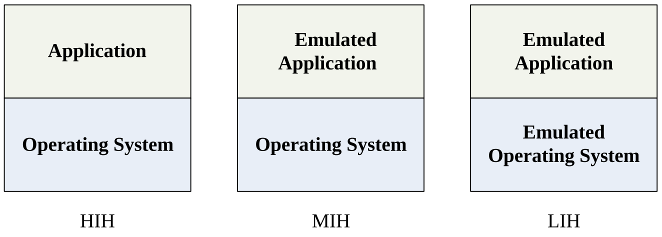

Fig. 3. Three types of fidelity.

Medium-interaction honeypot (MIH) provides much more interaction to the adversaries. However, unlike an LIH, a MIH does not implement the TCP/IP stack by itself. Instead, MIHs, e.g., Dionaea [18] and Cowrie [19], bind to sockets and leave the OS to do the connection management. In contrast to 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 fake responses will be created by the captor of the MIH.

High-interaction honeypot (HIH) is a fully functional system that can be completely compromised by adversaries. Its decoy is often a genuine system, such as Argos [8] and Cuckoo Sandbox [20]. Because the fully functional honeypot can be compromised, the HIH must equip the 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 [9], and Bailey's [10] and Lengyel's [13] hybrid honeypot architectures. In a hybrid system, the LIHs or MIHs are often used as frontends for large-scale deployments, and the HIHs are used as backends for deep investigation. These distributed frontends are named *sinkholes*, which could be the devices (i.e., sensors, redirectors, etc.), such as network telescopes [21], darknet [22], blackholes [23], IMS [24], and iSinks [25], or software artifices assigned with a portion of the routed IP address space. Instead of deploying a large number of HIHs across multiple networks, they can be centrally deployed in a consolidated location, which is called *honeyfarm*, such as the one used in Potemkin [26].

2) Scalability: It represents the capability to provide a growing number of decoys, or its potential to be enlarged to accommodate that growth. It is 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 [8] can only monitor one virtual decoy. On the contrary, a scalable honeypot system can deploy multiple decoys and its captor is able to monitor those decoys simultaneously, e.g., Honeyd [17] is able to emulate multiple OS fingerprinting artifices at the same time. A honeynet is a type of scalable honeypot system. In [12] and [27], the term of *honeynet* was proposed, which define a honeynet as a network consisting of HIHs that provides real systems, applications, and services for adversaries to interact with. The data captured by scalable honeypots deployed in multiple domains often need to be collected by secure channels and stored in an isolated data center for further analysis.

3) Adaptability: It 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 [28] and Dionaea [18], need the security researcher to determine the configuration beforehand and manually configure/reconfigure it. 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 an intrusion event.
3) It is not able to adapt to changes in the objective of the cloned network.

In contrast, a *dynamic honeypot* is able to adapt to specific events in a timely manner. It is able to change its configuration periodically, or even adapt to environmental changes in real-time, and respond to intrusion events, e.g., Honeyd [17] and Glastopf [29].

4) Role: It describes in which side the decoy plays within a multitier architecture. A honeypot can play two roles: server and client. This refers to whether a honeypot actively detects malicious program or passively captures unauthorized traffic. Most honeypots are server side ones, e.g., Honeyd [17] and Dionaea [18], which passively wait being attacked. Adversaries find these honeypots on their own initiative and probe and attack them. Most server-side honeypots never advertise themselves, but some can "advertise" themselves, e.g., Glastopf [29] that works like a normal web server with a number of vulnerable

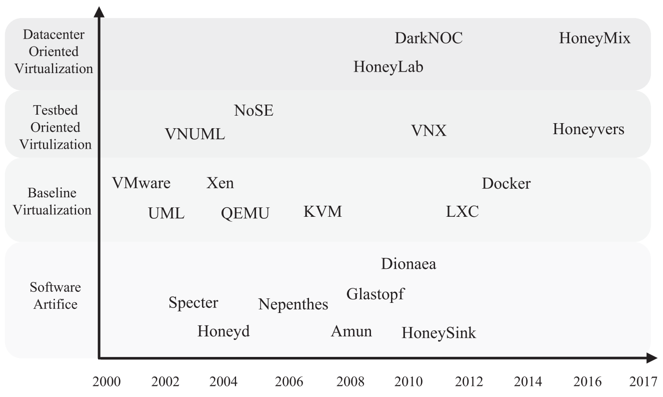

Fig. 4. Virtualization technologies for decoy deployment.

paths and scripts (referred to as dorks) so that the attackers can index them by using a search engine and/or web crawler. A *client honeypot* is used to investigate client-side intrusion. This type of honeypot can actively initiate requests to servers and investigate malicious program on the server side, such as Ghost [30].

5) Physicality/Virtuality (P/V): it denotes the 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 a virtual HIH. 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 network traffic sent to the virtual decoys [31]. We can have multiple virtual honeypots hosted concurrently by one physical machine.

Although according to the definition of a honeypot, any type of information system resource can be deployed as a decoy, the use of virtualization technologies has important advantages in terms of ease of management and maintenance. On the one hand, all the LIHs and MIHs are virtual honeypots according to the nature of their design. The decoys of LIHs are software artifices, which emulate the fingerprints of OSs and services. On the other hand, HIHs can be virtualized by using virtualization technologies. Galan *et al.* [32] summarized the virtualization technology evolution through three categories: baseline virtualization, testbed oriented virtualization, and datacenter oriented virtualization. Fig. 4 shows the virtualization technologies used in deploying HIHs over the last 17 years (dates are approximate).

For the HIHs using baseline virtualization technologies, the first example is the user-mode Linux (UML). This was used as the virtualization engine to mimic the Gen II Honeynet in [33], 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 silently capture the keystrokes of the honeypots. However, UML only enables Linux Kernel-based virtual machines (VM) to run as an application within a normal Linux host. Instead, Abbasi and Harris [34] used a VMware server to deploy a virtual Gen III Honeynet, which can support various OSs based on the x86 architecture. Different from the previous work, it applied a multisystem virtual honeynet architecture that installs the Honeywall on a separate singe virtual machine instead of the host. It also used Sebek to perform system activity capture in the virtual honeypots. Similarly, the KVM (Kernel-based virtual machine) hypervisor can also provide emulation of different OSs. Capalik's system [35] 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 have provided alternatives to the hypervisor-based virtualization for honeypot deployment. For example, Memari *et al.* [36] created a virtual honeynet, based on LXC (Linux Containers), which can simultaneously create multiple Linux user-space instances through partitioning the resource of the host. An LXC-based virtual machine can startup very quickly, but it can only emulate Linux over Linux. Another case in point is the honeypot [37] created using Docker, which can implement a high-level API to provide lightweight containers that run processes in isolation. The Docker container is even more lightweight than the LXC container since it implements application virtualization rather than full system virtualization.

Because 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) [38] proposed a high-level description for virtual honeynets and developed a tool to process the description automatically, avoiding the user having to deal with the complex low-level details. However, it only focuses on using UML as its underlying technology so it still can only emulate Linux Kernel-based virtual machines VM. Some generic tools that integrate multiple virtual machine hypervisors were also proposed. Network simulation environment (NoSE) [39] addressed the multihypervisor issue through integrating a variety of virtual machine hypervisors, such as UML, Xen, and QEMU, into one generic platform. The drawback of NoSE and the previous proposals is that they lack the capability of dynamic configuration for the honeynet deployment. VNX [40] is a more powerful generic virtualized tool, which integrates more hypervisors, such as UML, QEMU, KVM, LXC, etc., and can even undertake dynamic configuration. Following the idea of VNX, Fan *et al.* proposed the Honeyvers [41] framework aimed at creating and managing heterogeneous honeypots.

Apart from the tools described above, some other multitenant datacenter oriented virtualization technologies for HIHs deployment have also been proposed. Honeylab [42] provides a platform to share IP address space and computing resources. It is a distributed infrastructure overlay that allows security researchers to create their own desired honeypot systems without setting up distributed sensors in various geographical locations. DarkNOC [43] is designed to collect interesting traffic from different information sources, e.g., NetFlow, Snort, and Nepenthes, to analyze the data and present it to users in an efficient manner. In addition, Han *et al.* proposed the HoneyMix [44] system that treats a honeypot as a network security function and instantiates honeypots by using network function virtualization.

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

Sacrificial lamb is a normal system, but without connections to production networks, which waits to be compromised by attackers, e.g., Argos [8] and Cuckoo Sandbox [20]. It can be a commercial off-the-shelf computer, a router, or a switch, etc. The typical implementation involves loading the OS, configuring some applications, and then leaving it on the network to see what happens. Sacrificial lambs provide a mean of analyzing a compromised system down to the last byte. The analysis often requires numerous third-party tools. They also do not provide integrated traffic containment facilities, so will require additional network considerations.

Deception ports indicate simulated services disguised as well-known services on production systems. These are basically LIHs or MIHs, such as Specter [28] and Dionaea [18], which mimic various services on different ports of the system, e.g., HTTP is mimicked on port 80. These honeypots first "observe" the OS they reside on and then portray these services according to that. The basic idea is deception so that the adversaries are "stuck-up" in solving the deception from where they can be removed from the network.

Proximity decoys indicate that the decoys are deployed on the same network as production systems and possibly clone the configurations of the production systems. There will be no legal hassles in monitoring the decoys, because they are part of the same subnet as the production servers, and it is allowed to monitor any activity pertaining to your own network. Also, when some malicious attacks are detected on the production systems, it is easier to either reroute them to the honeypots, or trap them, since they are in proximity to the production systems. Honeyd [17] can use the free IP addresses of a production network to deploy and integrate the decoys into the production network, which 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," by which we mean switch on the data capture function 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, where they can use the contents of honeypots to reduce the probability of generating false alarms. Sinkholes, e.g., network telescopes [21], often uses this deployment strategy.

Redirection shield uses port redirection or traffic rerouting to forward malicious data to the honeypots. This strategy needs intrusion detection technology to evaluate the network traffic. If the traffic is interesting, it will be redirected to the honeypot shield to avoid the production system being attacked. The shield and the production network should be tightly or loosely coupled. The honeypots can reside either in the same address space as the production network or on another subnet alongside the production network, or even remotely. For example, shadow honeypots [5] following this deployment strategy use the shadow application as a shield dealing with the malicious traffic for anomaly-based detection.

7) Resource Type (R-Type): It denotes the type of information system resource available for the attacks. Most honeypots provide or emulate general resources, and are aimed at detection of more than one attack technique. Currently, many specific resource oriented honeypot systems have been proposed, which are given as follows.

1) *Web application honeypots* are tools aimed at detection of attacks on web application, e.g., Glastopf [29].
2) *VoIP honeypots* are used to capture threats in internet telephony (Voice over IP), e.g., Artemisa [45].
3) *SSH honeypots* are oriented against secure shell (SSH) attacks, e.g., Cowrie [19].
4) *Bluetooth honeypots* are aimed to capture attacks propagating through Bluetooth devices, e.g., Bluepot [46].
5) *USB honeypots* are used to investigate arbitrary malware on USB storage devices, e.g., Ghost USB Honeypot [30].
6) *SCADA/ICS honeypots* emulate industrial control system resources, e.g., Conpot [47].
7) *IoT honeypots* are used to capture attacks that target IoT devices, e.g., the IoTPOT [48].
8) *IPv6 network honeypots* are tools used to capture attacks targeting IPv6 networks, e.g., Hyhoneydv6 [49].

B. Functions of Captor

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

1) Attack Monitoring: It is aimed at logging all the intrusion events and malicious behaviors to allow further investigation. Two critical layers of data can be identified: *network activity* (every inbound and outbound connection, packet and header, 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 were found: tunneling and application-level proxying. *Tunneling* is used when some distributed decoys, such as network telescopes, darknets, and blackholes, are placed in a different location where the processing backends are. As the decoys are assigned a portion of the routed IP address space corresponding to its physical location, a tunnel mechanism based on a tunneling protocol such as 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, as the tunnel is almost invisible to "traceroute," although the tunnel will add some latency and modify the MTU. Some hybrid systems [26], [50] use GRE tunnels 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. Application-level proxies are also known as application-level gateways, and are available for common Internet services, e.g., an HTTP proxy is

TABLE I
VIRTUAL HONEYPOT INTROSPECTION SOLUTIONS

Solution	VM Hypervisor	Supported OS
Argos [8]	QEMU	Windows
Nitro [59]	QEMU	Windows
Timescope [60]	QEMU	Linux
Virtuoso [61]	QEMU	Windows, Linux, OS X, Haiku
DRAKVUF [62]	Xen	Windows
Cuckoo [20]	KVM	Windows, Linux, OS X, Android

used for Web access, and an FTP proxy is used for file transfers. Honeyd [17] provides 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 and 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 (e.g., application-level proxies can better support virus scanning) than circuit-level ones. Also, they are client-neutral and require no special software components or OS on the client computer to enable the client to communicate with servers through the proxy.

On the other hand, system activity monitoring needs to capture the malicious activity in the HIHs. Clearly the activity must be captured stealthily. Sebek [51] and Qebek [52] are examples of the first honeypot monitoring tools used to stealthily capture system activity. 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 some techniques that can detect the presence of this type of kernel module installed inside honeypots. The well-known CWSandbox [53] uses in-line code overwriting to hook the API function in order to observe malware behavior without being noticed. However, this approach still has the possibility of being detected.

In order to address this drawback, Jiang and Wang [54] proposed another monitoring approach called *"out-of-the-box,"* which uses the virtualization hypervisor to monitor the activity in guest . VMI-Honeymon [13] uses a volatility extension to call the API of the Xen Access successor LibVMI to access the memory of the guest VM. LibVMI [55] is a C library with Python bindings based virtual machine introspection that 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 were some other virtual machine introspection based approaches that could analyze malware and simultaneously make it harder for the malware to detect them, such as Livewire [56], VMScope [54], Lares [57], VMWatcher [58], etc. However, they are either not open-source software or not maintained any longer. Nevertheless, the solutions in Table I do provide maintained open-source code for particular hypervisors and OSs, as listed.

2) Attack Prevention: It is aimed at deterring or blocking intrusions. This function can be carried out by several approaches: data filtering, tarpitting, and containment.

Filtering consists of discarding the data traffic. This is typically specified by means of filtering rules. There are two filtering mechanisms: source-destination (Src-Dst) based and content-based. *Src-Ds-based filtering* examines the header information (mainly source and destination addresses, ports and protocols) of each packet to make the discarding decision. This mechanism is effective at reducing the amount of repeated traffic into nonredundant manageable data. iSinks [25] uses a filtering strategy of analyzing the connections established with the first N destination IPs per every source IP. Pang *et al.* [63] improved the filtering mechanism by taking into account the source port, destination, and connection. Bailey *et al.* [64] improved the Src-Dst-based filtering mechanism by expanding the individual darknets into multiple darknets for observing the global behavior and the source distribution. *Content-based filtering* inspects the content or payload of the packets to make the discarding decision. Bailey *et al.* [10] proposed content prevalence as a filtering mechanism by inspecting the first packet of each new payload. Content prevalence analyzes the distribution of content sequences in payloads and generates an alert when a specific piece of content sequence becomes prevalent. Similarly, IMS [24] proposed a caching mechanism to avoid recording duplicated payloads, by only recording the first payload packets in order to reduce disk utilization. A potential drawback of packet inspection based filtering is that it is unable to make a decision until the session has been established and at least the first packet of content or payload has been received. SweetBait [65] uses whitelists to filter the traffic that matches benevolent patterns in order to conduct zero-day worm detection. RolePlayer [66] 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 [5] uses a signature-based IDS to filter well-known attacks and then applies an anomaly-based IDS to filter the input into suspect traffic for further investigation.

Tarpitting consists of purposely slowing down the progress of an attack, worm propagation, or virus sprawl, etc. Collapsar's [67] tarpit module restricts an outgoing attack from a honeypot by throttling the packet rate that can be sent. Honeywall [68] is also a tarpit device that can limit the number of outgoing connections. It can block any outbound connection when it is capturing automated attacks, or when it is investigating manual attacks. It can be programmed to allow a limited number of outbound connections, such as five to ten connections per hour. However, this strict data tarpitting will raise an adversary's suspicion, as well as increase the chance of being detected, and impede data capture.

Containment is another approach to preventing an adversary from using a compromised honeypot to attack other nonhoneypot 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 limits the number of outgoing connections or discards them. Alata *et al.* [69] implemented such an outgoing connection redirection mechanism by modifying the Linux system kernel. 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.

3) Attack Detection: This function aims at detecting intrusion and generating alerts. There are two common detection approaches: signature-based and anomaly-based.

Signature-based detection identifies well-known attacks by recognizing malicious patterns. This approach is often used in production environments to discover unauthorized activity and generate alerts to administrators. Unlike the attacks captured by IDSs, which may contain false alerts, 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 a *production honeypot* and emulates well-known vulnerabilities to lure and deceive intruders so that they waste their time interacting with the honeypot. Production honeypots are often LIHs and MIHs that have little or no interaction with either the attacker or production systems in order to minimize the risk of infecting them. Furthermore, the performance and response times of production honeypots should be guaranteed and similar to production systems. For example, the production honeypot Dionaea [18] can simulate multiple well-known services to carry out signature-based detection.

Anomaly-based detection identifies unknown attacks by discovering deviations from patterns of normal behavior. Honeypots using this detection approach are always used in research environments as research honeypots. A *research honeypot* is designed to detect anomalies and investigate unknown signatures. Thus, research honeypots are often more powerful than production honeypots. HIHs and hybrid honeypot systems are always used as research ones to provide fully functional systems. A wider assortment of data can be captured to facilitate further investigation. Research honeypots are a step ahead of production ones. The signatures of new attacks discovered by research honeypots are often used to update production ones, and provide an early warning and prediction of future attacks and exploits. A number of anomaly-based detection techniques have been proposed in the context of honeypot research. For example, Argos [8] applies dynamic taint analysis [70] to detect zero-day attacks and generate new signatures; Honeycomb [71] uses the longest common substring algorithm to detect repeating patterns in order to spot worms; and Bailey's system [10] performs system behavior profiling by comparing an infected virtual filesystem with an uninfected one. In addition, some current learning techniques, such as the deep learning approach for NIDS [72], can also be used in decoys so as to acquire new detection skills for identifying unknown attacks.

Apart from the traditional IDS techniques, Sekar *et al.* [73] proposed specification-based anomaly detection using a supervised method to develop the specification, instead of unsupervised machine learning techniques. This identifies legitimate behavior and detects unknown attacks as deviations from the norm. Not only does it improve the effectiveness of anomaly detection over signature-based approaches, but also minimizes the large number of false positives produced by other anomaly-based techniques.

4) Attack Response: It relates to the measures taken to respond to attacks and adapt to intrusion events based on predefined requirements. These honeypots can take two type of reaction: traffic redirection and decoy reconfiguration.

a) Traffic redirection: It is used to control how traffic is sent to an appropriate destination. For example, hybrid honeypots redirect malicious traffic from a LIH to an isolated HIH for further investigation. We review two redirection techniques: Flow-based routing and TCP connection replaying.

Flow-based routing is where packets are routed from source to destination by selecting the path that satisfies some requirements such as QoS, load balance, security, etc. This mechanism is based on the same principles as used for normal network routing, but is applied to more specific data flows. Kohler *et al.* [74] proposed the flexible and configurable Click modular router, which is made of simple packet processing modules that are combined in a service chain in order to build complex and efficient network services that can be used to do flow-based routing. There are several cooperative honeypot systems that use the Click framework to facilitate data control. For example, the Potemkin gateway router and the GQ gateway are based on the Click modular router. 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 the fine-grained dynamic control of traffic by means of flow table entries configured on each OpenFlow switch. In the near future, programmable SDN based network architectures will increasingly take the role of data control for honeypot systems [75].

TCP connection replaying is a connection handover technique aimed at seamlessly transferring a TCP socket endpoint from one node to another. When an interesting connection is established between the attacker and a 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 [10] avoids conserving the state of every connection, since 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. The Honeybrid gateway [76] uses the connection replay mechanism to implement transparent traffic redirection between LIHs and HIHs. Furthermore, Honeybrid revealed the technical details of the gateway, which is a TCP replay proxy using libnetfilter_queue [77] to process packets. The connection handoff mechanism based on TCP replay is able to provide stealthy redirection for automated malware. In [78], Lin *et al.* proposed a transparent and secure network environment that allows automated malware to attack or propagate, but under stealthy control. Although TCP/IP stateful traffic replay can facilitate transparent TCP connection handoff, it cannot solve the identical-fingerprint problem, because the LIH and HIH have different fingerprints (e.g., IP and MAC addresses). This problem leaves the opportunity for the skilled adversary to detect the honeypot environment. VMI-Honeymon [79] provided a novel solution that retains the MAC and IP address of the original HIH for cloned HIHs but creates separate network bridges to isolate them so as to avoid address collisions. Fan *et al.* [80], [81] proposed a hybrid honeypots based traffic redirection mechanism intending for addressing the identical-fingerprint problem. Its drawbacks are that different honeypots using the identical-fingerprint have to frequently switch up and down according to research requirements, and it is hard to conduct large-scale deployment using the proposed hybrid architecture. Most recently, Fan and Fernández [82] proposed a novel SDN-based stealthy TCP connection handover mechanism that solved this problem through using different ports of an OpenFlow-based switch to isolate the honeypots whilst keeping identical-fingerprints.

b) Decoy reconfiguration: It is designed to timely adapt the decoy's state to specific events, which could be intrusion events, state variation of targets, etc. As stated, static honeypot systems lack the capability to reconfigure the decoy timely. This is a critical disadvantage when honeypots are deployed in complex and dynamic network scenarios. Several approaches have been proposed to address this problem, which can be categorized into dynamic cloning and dynamic catering.

Dynamic cloning synchronously emulates the real production targets including network topologies, OS fingerprints, services, open ports, etc. It is designed to rapidly alter the configuration and deployment by monitoring and learning the target organization's network in real time. Thus, dynamic cloning has two phases. The first phase is called network discovery, and is used to collect information about the target network. The second phase is called honeypot deployment, which deploys decoys that emulate the target systems. There are two ways to discover the targets: passive and active fingerprinting. Hecker and Hay [83] discuss both ways for network discovery and automated honeynet cloning. Passive fingerprinting tools, such as p0f [84], can sniff the traffic, determine active systems, and open ports in the target scenario, whilst making 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 [85], can discover all open ports on the target system, even if there is no production traffic to those ports, at the price of generating some extra production traffic. In [9], 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 about the organization's network. 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 [17] to represent the production systems. It also uses virtual HIHs to receive the redirected traffic from the LIHs, but the dynamic deployment of HIHs is not mentioned.

Dynamic catering is used to create honeypots that cater for certain attacks by gradually escalating the interaction level to capture malicious data, and redeploy honeypots when intrusion activity is detected. The idea is to create and deploy honeypots on demand to increase the efficiency of data capture. Potemkin [26] used dynamically created HIHs on physical servers to achieve efficient resource usage. It employs a network gateway, to which routers all over the Internet tunnel traffic. The gateway acts as an agent to send 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. When an HIH is idle, the gateway instructs the VMM to destroy it and reclaim the resources. VMI-Honeymon [79] clones VMs by restoring the memory snapshot of its configuration on a QEMU copy-on-write (qcow2) filesystem. The newly created virtual HIH runs the system and applications with the same fingerprints as the cloned one for investigating the attacks.

5) Attack Profiling: It is the extrapolation of attack information in order to analyze malicious activity, as well as unveil the intruder's motives. McGrew and Vaughn, Jr. [86] indicated that an attack profile should contain the following attributes.

1) *Motivation* describes the reason of the attack.
2) *Breadth/Depth* presents the scope of the attack and the degree of impact to the attacked system.
3) *Sophistication* shows the level of technical expertise needed to carry out the attack.
4) *Concealment* describes the measures used for hiding evidence of the attack.
5) *Attacker(s)* defines the entity(ies) behind the attack: An individual or a group of adversaries, and identifies the source of the attack, e.g., automated malware.
6) *Vulnerability* is the flaw that can be exploited by the attack.
7) *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.

Some of these attributes can be obtained directly from the captured honeypot data. For example, 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 depth of the attack. Also, the concealment and tools can be identified by observing the adversary's activity on the honeypot. Using basic statistics on the log information is called *direct information based* attack profiling. Some honeypots, e.g., Honeyd [17] and Dionaea [18], use the IP source, destination, and timestamp of an attack to describe the attack profile.

However, the other attributes have to be obtained from derived information. Motivation can be inferred from insights into the activity on the HIH. Identifying the attacker and the sophistication needs in-depth observation and forensics on the interaction between the attacker and the honeypot. Determining the vulnerabilities often needs advanced detection techniques. Therefore, *derived information based* attack profiling is much more complex, since it tries to assess and explain the fundamental cause of the attack. Basic statistics are insufficient for this. It is necessary to apply multiple approaches, e.g., association rule mining, neural networks, virtual machine introspection, etc. Currently, plenty of techniques have been suggested for analyzing malicious data: Nawrocki *et al.* [11] reviewed the different approaches for analyzing honeypot data; Egele *et al.* [87] surveyed automated dynamic malware analysis techniques and tools; Rieck *et al.* [88] presented the research

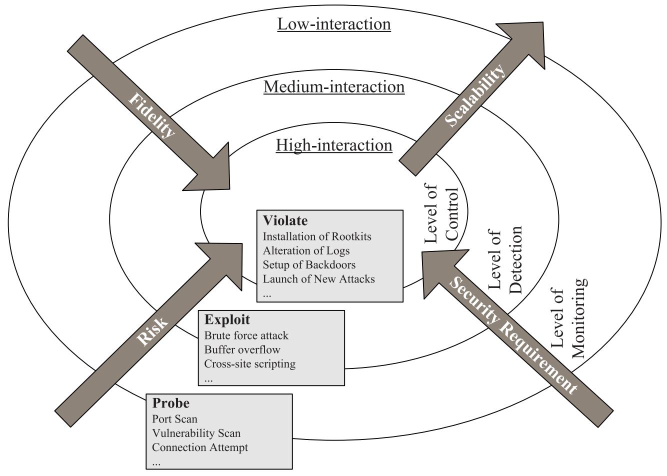

Fig. 5. Constraints between main features.

on machine learning techniques for honeypot system behavior analysis.

IV. Design Space and Taxonomy Validation

This section will introduce the design space constructed by the taxonomy, present some feature constraints that can influence the theoretical design space, proposes two subsets of features that can significantly affect the honeypot performances, and also apply these features to a number of typical honeypots with the purpose of validating the taxonomy.

A. Design Space and Feature Constraints

Depending on the classification scheme, the honeypot designer can theoretically observe at most 103 680 different combinations of features, which provides a global view of the design space of homogeneous honeypots. However, we have to note that some features are mutually exclusive (see Fig. 5) and this leads to a reduction of the design space.

From the point of view of resisting attacks, malicious data can be captured when honeypots are probed, attacked, and compromised. This corresponds exactly to the three phases of a cyber attack, i.e., probe, exploit, and violate. Attacks always begin by probing large-scale IP networks in order to find vulnerable nodes, and then exploiting the vulnerabilities to compromise the nodes. Finally, if the compromised nodes are worth utilizing, the adversary will violate them, e.g., by installing rootkits, setting up backdoors, or launching new attacks, etc. The large-scale probing will produce high network traffic, but it will not translate into useful system activity. When the vulnerable nodes are attacked, i.e., in the exploiting phase, the scale of the attack is reduced, but the data quality is enhanced, i.e., the attacking traffic will include malicious payloads. In the violating phase, only a small part of the compromised network, i.e., several specific nodes, will be involved, and the data quality becomes very high because any unauthorized system data is worth recording for further investigation. Therefore, every phase produces different data quantity and quality. The fidelity and scalability features are highly related to the three attack phases.

However, fidelity and scalability are a pair of mutually competing features in a decoy. In order to capture high quality data, the decoy has to increase the interaction level. However, a higher interaction level leads to a higher risk of being compromised, so the honeypot has to enhance the captor to protect the decoy. Consequently, higher interaction guarantees the fidelity but sacrifices scalability, which will result in failing to capture adequate data from large-scale IP networks. For example, Provos [17] showed that a 1.1 GHz Pentium III can support thousands of LIHs created by Honeyd. The performance penalty is little, though it depends on the complexity and number of simulated services available for each decoy: for 1000 templates, the processing time is about 0.022 ms per packet, whereas for 250 000 templates, the processing time increases to 0.032 ms. However, Honeyd needs to forward interesting traffic to HIHs to detect compromises and unusual activity. The HIH Argos [8] can detect zero-day attacks, but it has a large performance penalty: even in the fastest configuration, it is at least 16 times slower than the host. So there needs to be a good balance between these two features in order to optimize the use of honeypot resources. Cooperative honeypots, particularly hybrid honeypots have been developed to overcome these issues.

Also, from the above discussion, we can see that attack profiling by the captor is highly related to the fidelity of the decoy. If the captor wants to perform attack profiling based on derived information, the decoy needs a high interaction level in order to record detailed enough information (i.e., system activity) about the attack. Otherwise, if the honeypot is a LIH or MIH, the attack profiling can only use direct information.

Furthermore, adaptability is highly related to the P/V. It is observable that physical honeypots are often static, while virtualization technology has made it easy to create dynamic honeypots. The software artifice is the easiest way to enforce dynamic configuration, and at present, virtual machine based HIHs are increasingly convenient to perform dynamic configuration, e.g., Fan *et al.* [80] demonstrated that a KVM-based virtual honeypot can be activated to work in 13 s from the hibernated state and even only in 1.5 s from the suspended state. Therefore, reconfiguration of the decoy's attack response is also tightly related to the P/V.

Overall, the design space can help to predict future theoretical designs, while the constraints among the different features can provide a more practical way for the designer to implement honeypots in specific technical environments.

B. Subsets of Features Effecting Honeypot Performances

According to the above analysis and the honeypots revisit with our proposed D-C-based taxonomy in Section III, we realized that several features can actually influence the honeypot performances in practice. Thereby, we conclude two subsets of effective features for the independent and cooperative honeypots, respectively. By using these features, a honeypot designer can improve its honeypot or select the most effective one for its system.

For independent honeypots, we determine the features of class fidelity, scalability, adaptability, role, D-strategy, R-type, attack

TABLE II
SUBSET OF FEATURES FOR INDEPENDENT HONEYPOT PERFORMENCES

	Independent Honeypots											
	Specter(2003)[28]	Network Telescopes(2004)[21]	Honeyd(2004)[17]	Argos(2006)[8]	Glastopf(2009)[29]	Dionaea(2011)[18]	Artemisa(2011)[45]	Bluepot(2012)[46]	Ghost(2012)[30]	Conpot(2013)[47]	Cuckoo Sandbox(2014)[20]	Cowrie(2015)[19]
Fidelity												
Low	●	●	●	○	●	○	●	○	○	●	○	○
Medium	○	○	○	○	○	●	○	●	●	○	○	●
High	○	○	○	●	○	○	○	○	○	○	●	○
Scalability												
Unscalable	○	○	○	●	○	●	○	○	●	○	●	○
Scalable	●	●	●	○	●	○	●	●	○	●	○	●
Adaptability												
Static	●	●	○	●	○	●	○	●	●	●	●	●
Dynamic	○	○	●	○	●	○	●	○	○	○	○	○
Role												
Server	●	●	●	●	●	●	●	○	○	●	○	●
Client	○	○	○	○	○	○	○	●	●	○	●	○
D-Strategy												
Sacrificial lamb	○	○	○	●	○	○	○	○	●	○	●	○
Deception ports	●	○	○	○	●	●	○	●	○	●	○	●
Proximity decoys	○	○	●	○	○	○	●	○	○	○	○	○
Minefield	○	●	○	○	○	○	○	○	○	○	○	○
Redirection shield	○	○	○	○	○	○	○	○	○	○	○	○
R-Type												
General	●	●	●	●	○	●	○	○	○	○	●	○
Web app	○	○	○	○	●	○	○	○	○	○	○	○
VoIP	○	○	○	○	○	○	●	○	○	○	○	○
SSH	○	○	○	○	○	○	○	○	○	○	○	●
Bluetooth	○	○	○	○	○	○	○	●	○	○	○	○
USB	○	○	○	○	○	○	○	○	●	○	○	○
SCADA/ICS	○	○	○	○	○	○	○	○	○	●	○	○
Monitoring												
Network activity	●	●	●	○	●	●	●	●	●	●	○	●
System activity	○	○	○	●	○	○	○	○	○	○	●	○
Detection												
Signature-based	●	○	○	○	●	●	○	●	●	●	●	○
Anomaly-based	○	○	○	●	○	○	○	○	●	○	○	○
Profiling												
Direct info	●	●	●	○	●	●	●	●	●	●	○	●
Derived info	○	○	●	●	○	○	○	○	●	○	●	○

TABLE III
SUBSET OF FEATURES FOR COOPERATIVE HONEYPOT PERFORMENCES

	Cooperative Honeypots													
	Bailey's System(2004)[10]	Collapsar(2004)[67]	Shadow Honeypots(2005)[5]	Potemkin(2005)[26]	Gen II Honeynet(2005)[89]	Artail's System(2006)[9]	GQ(2006)[50]	SweetBait(2007)[65]	Honeybrid(2008)[76]	SGNET(2008)[90]	Li's System(2009)[91]	VMI-Honeymon(2013)[79]	IoTPOT(2015)[48]	Hyhoneydv6(2015)[49]
P/V														
Physical	○	○	●	○	●	●	○	○	○	○	●	○	○	○
Virtual														
Software artifice	●	○	●	○	○	●	●	●	●	●	●	●	●	●
VMware	●	○	○	○	●	○	●	○	●	○	●	○	○	○
UML	○	●	○	○	●	○	○	○	○	○	○	○	○	○
Xen	○	○	○	●	○	○	○	○	○	○	○	●	○	○
QEMU	○	○	○	○	○	○	○	●	○	●	○	○	●	●
KVM	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Role														
Server	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Client	○	●	●	○	○	○	○	○	○	○	●	○	○	○
D-Strategy														
Sacrificial lamb	●	●	○	●	●	●	●	●	●	●	○	●	●	●
Deception ports	○	○	○	○	○	○	○	○	○	○	○	●	●	○
Proximity decoys	●	○	○	○	●	●	○	●	●	●	●	○	○	●
Minefield	○	●	○	●	○	○	●	○	○	○	○	○	○	○
Redirection shield	○	○	●	○	○	○	○	○	○	○	○	○	○	○
R-Type														
General	●	●	●	●	●	●	●	●	●	●	●	●	○	○
IoT	○	○	○	○	○	○	○	○	○	○	○	○	●	○
IPv6	○	○	○	○	○	○	○	○	○	○	○	○	○	●
Monitoring														
App-layer proxying	○	○	●	○	○	●	○	○	○	○	○	○	○	●
GRE tunneling	○	●	○	●	○	○	●	○	○	○	○	○	○	○
In-the-box	●	●	●	○	●	●	○	○	●	○	○	○	○	○
Out-of-the-box	○	○	○	●	○	○	●	●	○	●	○	●	○	○
Prevention														
Src-Dst filtering	○	○	○	●	○	○	●	○	●	○	○	●	○	○
Content filtering	●	○	●	●	○	○	●	●	●	○	○	●	○	○
Tarpitting	○	●	○	○	●	●	●	○	●	○	○	●	○	○
Containment	●	○	○	●	○	○	●	○	○	○	○	●	○	○
Detection														
Signature-based	○	●	○	○	●	●	○	○	●	○	●	●	●	○
Anomaly-based	●	○	●	○	○	○	○	●	●	●	○	●	○	○
Response														
Flow-based routing	○	●	○	●	○	○	○	○	○	○	○	○	○	○
TCP replaying	●	○	○	○	○	○	○	○	●	●	○	●	○	○
Dynamic cloning	○	○	○	○	○	●	○	○	○	○	○	○	○	○
Dynamic catering	●	○	○	●	○	○	●	○	○	○	○	●	○	●

monitoring, detection, and profiling as the effective ones. We omit the other classes for the reasons as follows: as P/V, this class is highly consistent with class fidelity. All the LIHs and MIHs are software artifices, and all the HIHs are physical or virtualization techniques based; for attack prevention, single independent honeypots do not provide any function related to this class; for attack response, single independent honeypot systems lack function about this class. Consequently, Table II demonstrates the subset of features effecting the independent honeypot performances.

On the other hand, for cooperative honeypots, we conclude the features of class P/V, D-strategy, R-type, attack monitoring, prevention, detection, and response as the effective ones. We ignore the other classes since cooperative honeypots often include both the advantages of the LIHs/MIHs and the HIHs, so they are high-interaction (for fidelity), scalable (for scalability), dynamic (for adaptability), and can provide derived information (for attack profiling). So, Table III illustrates the subset of features effecting the cooperative honeypot performances.

Therefore, the class role, D-strategy, R-type, attack monitoring, and attack detection are the ones that include features both the independent and cooperative honeypots must take into account. Meanwhile, the above two tables shows the D-C-based taxonomy can fully classify these different types of honeypots. It can be observed that every honeypot has its own features, which could be strengths or weakness in terms of different security issues and scenarios. The strengths and weakness of a honeypot are not absolute, but they are tightly related to the security requirements.

TABLE IV
COMPARISON OF SUBSYSTEMS OF HYBRID HONEYPOTS

Hybrid honeypot	Frontend	Controller	Backend
Bailey's system	Honeyd	A central controller	VMware VM
Artail's system	Honeyd	Honeywall	Physical machine
GQ	Network telescopes	Click-based router	VMware ESX
SweetBait	Honeyd + honeycomb	–	QEMU-based Argos
Honeybrid	Honeyd	Honeybrid gateway	VMware VM
SGNET	Honeyd + ScriptGen	SGNET gateway	QEMU-based Argos
Li's system	Spamtrap + Phoneybot + Phoneytoken	–	Phoneypot
VMI-Honeymon	Dionaea	Honeybrid gateway	Xen VM
IoTPOT	Frontend responder	–	QEMU-based IoTBox
Hyhoneydv6	Honeydv6	–	QEMU VM

V. HONEYPOT DEVELOPMENT TRENDS

A. Hybrid Honeypots

According to the requirement of decoupling and achieving optimization of both fidelity and scalability, many cooperative honeypots (particularly, hybrid honeypots) have been developed (see Table III). A typical hybrid honeypot consists of three subsystems: frontends, controller, and backends. The frontends can be LIHs/MIHs or sinkholes for monitoring large-scale routed IP address spaces, the backends can be HIHs or a honeyfarm, and the controller can be, e.g., a Honeywall, Click modular router, or Honeybrid gateway. These three subsystems are in charge of providing different functions. The frontends often provide low interaction with the attacks, because their main objective is to capture network data. Also, they need to discard the uninteresting traffic for capturing fine-grained data. The controllers are used to perform data control as well as dynamic system configuration in a hidden way. The backends perform stealthy system data capture and data analysis such as digital forensics to unveil the attacker's skills, tactics, and motives. Table IV shows a comparison of the subsystems of various hybrid honeypots.

We see that Honeyd takes the role of frontend in most hybrid honeypots. The wide applications of Honeyd are probably accounted for by its advantages of lightweight design, distributed appearance, programmable components, and dynamic features. Honeyd is a virtual LIH framework that can deploy multiple decoys concurrently following a configurable network topology. Though it can only emulate LIHs, it still has following advantages.

1) Based on the OS fingerprinting database of Nmap, it can fabricate decoys with almost all the common OS fingerprints.
2) Users can implement their own fake service responses in python in order to capture data—Honeyd may even emulate a service so that it actually collects 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.

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

Most hybrid honeypots use VM to deploy their backends. The most popular hypervisors are Xen and QEMU. Many dynamic configuration and virtual memory introspection solutions have been proposed based on these hypervisors. With the evolution of QEMU-KVM, we can foresee that the KVM will be mainly responsible for the backends deployment. The following section will detail the virtual honeypot development.

B. Virtual Honeypots

The development of honeypots now relies heavily on the progress of virtualization technology. Virtual honeypots provide several valuable advantages: ease of maintenance, dynamic configuration, and antidetection.

Virtualization leads to ease of maintenance. First, by using virtualization technology, one physical machine can simultaneously host multiple virtual honeypots, which can significantly improve resource efficiency. Second, the time-consuming task of large-scale honeypot deployment is greatly decreased by using virtualization techniques that only take several minutes to deploy rather than several hours on physical machines, e.g., the physical honeypot designed by Cliff Stoll in 1986 [14].

Virtualization also facilitates dynamic configuration, which is often used to reduce the response times to specific events. As previously stated, dynamic honeypots can be used to clone production systems and to synchronize with their changes in a timely manner. They can also be used to investigate intrusions by modifying their own state according to the requirements of the attack research. For example, dynamic configuration can facilitate 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 antidetection, which is described next.

Antidetection aims to avoid the honeypot being detected. Virtualization technology provides several ways to hide both the decoy and the captor. The captor is hidden first, by virtual machine memory introspection that facilitates "out-of-the-box" monitoring. This improves the stealthy monitoring capability of HIHs. Second, 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 antidetection. For limited-function honeypots, antidetection focuses on camouflaging the fact that the decoy is a honeypot. For example, because the link latency of a Honeyd-based decoy can lead to its detection, Fu *et al.* [92] reduced this in order to camouflage the Honeyd-based decoy. Additionally, once a decoy has been detected, the inbound traffic rate by the attacker will be reduced [93], so in this case, the system can redeploy the decoy to perform antidetection.

C. Special Purpose Honeypots

An increasing number of special purpose honeypots [47]–[49], [88] have been developed. First, both independent honeypots and cooperative honeypots are focused on developing specific attacked-resource oriented honeypots. These honeypots focus on fully emulating one type of information resource so that they can obtain fine-grained data. With the rapid growth of cyberspace, both SCADA/ICS and IoT systems are faced with increasing cyber threats. Consequently, honeypots for these are now being developed. Thus, the trend in honeypot development closely follows industrial developments. Second, research honeypots, particularly for anomaly-detection and attack profiling, have become increasingly numerous. These rely on cutting-edge computer science technologies, such as machine learning, big data analysis, etc.

VI. Conclusion

As a rapidly developing technology, honeypots have become a hot research topic in the field of computer and network security. From a variety of honeypot systems, we have captured the two common essential elements: the decoy and the captor. We have highlighted the trend to decouple these two elements from tight to loose coupling, in order to reduce the risk that a change within one component will create unanticipated changes within the other.

Based on this core concept, we have proposed the D-C-based taxonomy, which helps us to investigate honeypot systems. Thanks to the taxonomy, we identified various decoy features and reviewed a large number of honeypot techniques and related cutting-edge technologies. Broadly speaking, current honeypot development have followed two approaches: independent honeypots and cooperative honeypots. On the one hand, owing to the advantages of lightweight design, low-cost development, ease of management, resource efficiency, etc., independent honeypots have steadily developed in various application scenarios, with numerous examples of specific attacked-resource oriented honeypots emerging as independent software. On the other hand, cooperative honeypots cannot only provide broader views due to their distributed and cooperative deployment in different network domains, but also create opportunities for early network anomaly detection, attack correlation, and global network status inference. Also, cooperative honeypots 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 D-C are the two fundamental elements, which originate all the development in this important area. Therefore, our work can help security researchers gain insights into honeypot research and explore the designs and application space of future honeypot systems.

Acknowledgment

The authors would like to thank Prof. D. Chadwick from the University of Kent, Canterbury, U.K., for conducting proofreading to improve the quality of this entire paper.

References

[1] S. Brandes, "The newest warfighting domain: Cyberspace," *Synesis: A J. Sci., Technol., Ethics, Policy*, vol. 4, pp. G90–95, 2013.

[2] "Internet security threat report," Symantec Corporation, USA, Tech. Rep. no. ISTR, vol. 21, Apr. 2016.

[3] G. J. Rattray, "An environmental approach to understanding cyberpower," in *Cyberpower and National Security*, Sterling, VA, USA: Potomac Books, Inc., 2009, pp. 253–274.

[4] S. Peisert, M. Bishop, and K. Marzullo, "What do firewalls protect? An empirical study of firewalls, vulnerabilities, and attacks," Univ. California Davis, Davis, CA, USA, Tech. Rep. CSE-2010-8, 2010.

[5] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis, "Detecting targeted attacks using shadow honeypots," in *Proc. 14th Conf. Usenix Security Symp.*, Berkeley, CA, USA, 2005, vol. 14, pp. 9–25.

[6] W. Fan, K. Lwakatare, and R. Rong, "Social engineering: I-e based model of human weakness for attack and defense investigations," *Int. J. Comput. Netw. Inf. Security*, vol. 9, no. 1, pp. 1–11, 2017.

[7] V. Mateos, V. A. Villagrá, F. Romero, and J. Berrocal, "Definition of response metrics for an ontology-based automated intrusion response systems," *Comput. Elect. Eng.*, vol. 38, no. 5, pp. 1102–1114, 2012.

[8] G. Portokalidis, A. Slowinska, and H. Bos, "Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," in *Proc. 1st ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst.*, 2006, pp. 15–27.

[9] H. Artail, H. Safa, M. Sraj, I. Kuwatly, and Z. Al-Masri, "A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks," *Comput. Security*, vol. 25, no. 4, pp. 274–288, Jun. 2006.

[10] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos, "A hybrid honeypot architecture for scalable network monitoring," Univ. Michigan, Ann Arbor, MI, USA, Tech. Rep. CSE-TR-499-04, 2004.

[11] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A survey on honeypot software and data analysis," *CoRR*, vol. abs/1608.06249, 2016. [Online]. Available: http://arxiv.org/abs/1608.06249

[12] L. Spitzner, "The honeynet project: Trapping the hackers," *IEEE Security Privacy*, vol. 1, no. 2, pp. 15–23, Mar. 2003.

[13] T. K. Lengyel, J. Neumann, S. Maresca, B. D. Payne, and A. Kiayias, "Virtual machine introspection in a hybrid honeypot architecture," presented at the 5th Workshop Cyber Security Experimentation Test, Berkeley, CA, USA, 2012.

[14] C. Stoll, *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. New York, NY, USA: Gallery Books, 2000.

[15] B. Cheswick, "An evening with berferd in which a cracker is lured, endured, and studied," in *Proc. Winter USENIX Conf.*, 1992, pp. 163–174.

[16] L. Spitzner, "Honeypots: catching the insider threat," in *Proc. 19th Annu. Comput. Security Appl. Conf.*, Dec. 2003, pp. 170–179.

[17] N. Provos, "A virtual honeypot framework," in *Proc. 13th Conf. USENIX Security Symp.*, Berkeley, CA, USA, 2004, pp. 1–14.

[18] "Dionaea—Catched bugs," Nov. 2011. [Online]. Available: http://dionaea.carnivore.it/.

[19] M. Oosterhof, "Cowrie—Active kippo fork," Jul. 2015. [Online]. Available: http://www.micheloosterhof.com/cowrie/.

[20] CuckooFoundation, "Cuckoo," Oct. 2014. [Online]. Available: http://www.cuckoosandbox.org/.

[21] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Network telescopes: Technical report," Univ. California at San Diego, La Jolla, CA, USA, Tech. Rep. CS2004-0795, Jul. 2004.

[22] T. CYMRU, "The darknet project," Jul. 2015. [Online]. Available: http://www.team-cymru.org/darknet.html.

[23] D. Song, R. Malan, and R. Stone, "A snapshot of global Internet worm activity," in *Proc. 14th Annu. Comput. Security Incident Handling*, Jun. 2002, pp. 1–6.

[24] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The internet motion sensor: A distributed blackhole monitoring system," in *Proc. Netw. Distrib. Syst. Security Symp.*, 2005, pp. 167–179.

[25] V. Yegneswaran, P. Barford, and D. Plonka, " On the design and use of internet sinks for network abuse monitoring," in *Recent Advances in Intrusion Detection*, E. Jonsson, A. Valdes, and M. Almgren, Eds., vol. 3224, Berlin, Germany, Springer, 2004, pp. 146–165.

[26] M. Vrable *et al.*, "Scalability, fidelity and containment in the Potemkin virtual honeyfarm," in *Proc. ACM Symp. Operating Syst. Principles*, Oct. 2005, vol. 39, no. 5, pp. 148–162.

[27] "Know your enemy: Honeynets," May 2006. [Online]. Available: http://old.honeynet.org/papers/honeynet/.

[28] L. Spitzner, "Specter: A commercial honeypot solution for windows," 2003. [Online]. Available: http://www.symantec.com/connect/articles/specter-commercial-honeypot-solution-windows/.

[29] L. Rist, "Glastopf project," 2009. [Online]. Available: http://glastopf.org/.

[30] S. Poeplau and J. Gassen, "A honeypot for arbitrary malware on USB storage devices," in *Proc. 2012 7th Int. Conf. Risks Security Internet Syst.*, Oct. 2012, pp. 1–8.

[31] N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*, 1st ed. Reading, MA, USA: Addison Wesley, Jul. 2007.

[32] 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," *Ann. Telecommun.—Annales Des Télécommun.*, vol. 64, no. 5, pp. 305–323, 2009.

[33] L. K. Yan, "Virtual honeynets revisited," in *Proc. IEEE SMC 6th Annu. Inf. Assurance Workshop*, Jun. 2005, pp. 232–239.

[34] F. Abbasi and R. Harris, "Experiences with a generation iii virtual honeynet," in *Proc. Australasian Telecommun. Netw. Appl. Conf.*, Nov. 2009, pp. 1–6.

[35] A. Capalik, "Next-generation honeynet technology with real-time forensics for U.S. defense," in *Proc. IEEE Mil. Commun. Conf.*, Oct. 2007, pp. 1–7.

[36] N. Memari, K. Samsudin, and S. Hashim, "Towards virtual honeynet based on LXC virtualization," in *Proc. IEEE Region 10 Symp.*, Apr. 2014, pp. 496–501.

[37] P. Kasza, "Creating honeypots using docker," 2015. [Online]. Available: https://www.itinsight.hu/blog/posts/2015-05-04-creating-honeypots-using-docker.html.

[38] F. Galán and D. Fernández, "Use of vnuml in virtual honeynets deployment," in *Proc. IX Reunión Española sobre Criptología y Seguridad de la Información*, Barcelona, Spain, 2006, pp. 600–615.

[39] F. Stumpf, A. Görlach, F. Homann, and L. Brückner, "Nose-building virtual honeynets made easy," in *Proc. 12th Int. Linux Syst. Technol. Conf.*, 2005, pp. 1664–1669.

[40] D. Fernández, *et al.*, "Distributed virtual scenarios over multi-host linux environments," in *Proc. 5th Int. DMTF Academic Alliance Workshop Syst. Virtualization Manage.*, Oct. 2011, pp. 1–8.

[41] W. Fan, D. Fernández, and Z. Du, "Versatile virtual honeynet management framework," *IET Inf. Security*, vol. 11, no. 1, pp. 38–45, Mar. 2016.

[42] W. Chin, E. Markatos, S. Antonatos, and S. Ioannidis, "Honeylab: Large-scale honeypot deployment and resource sharing," in *Proc. 3rd Int. Conf. Netw. Syst. Security*, Oct. 2009, pp. 381–388.

[43] B. Sobesto, M. Cukier, M. Hiltunen, D. Kormann, G. Vesonder, and R. Berthier, "DarkNOC: Dashboard for honeypot management," in *Proc. 25th Int. Conf. Large Installation Syst. Admin.*, 2011, pp. 16–16.

[44] W. Han, Z. Zhao, A. Doupé, and G.-J. Ahn, "Honeymix: Toward SDN-based intelligent honeynet," in *Proc. 2016 ACM Int. Workshop Security Softw. Defined Netw. Netw. Funct. Virtualization*, 2016, pp. 1–6.

[45] R. do Carmo, M. Nassar, and O. Festor, "Artemisa: An open-source honeypot back-end to support security in VoIP domains," in *Proc. 12th IFIP/IEEE Int. Symp. Integrated Netw. Manage. Workshops*, May 2011, pp. 361–368.

[46] 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.

[47] L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, and J. Smith, "Conpot ICS/SCADA honeypot," Nov. 2013. [Online]. Available: http://conpot.org/.

[48] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the rise of IoT compromises," in *Proc. 9th USENIX Workshop Offensive Technol.*, Aug. 2015, pp. 1–9.

[49] S. Schindler, B. Schnor, and T. Scheffler, "Hyhoneydv6: A hybrid honeypot architecture for IPV6 networks," *Int. J. Intell. Comput. Res.*, vol. 6, no. 2, pp. 562–570, 2015.

[50] W. Cui, V. Paxson, and N. C. Weaver, "GQ: Realizing a system to catch worms in a quarter million places," , Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep. TR-06-004, 2006.

[51] "Know your enemy: Sebek, a kernel based data capture tool," Nov. 2003. [Online]. Available: http://old.honeynet.org/papers/sebek.pdf.

[52] "Know your tools: Qebek—Conceal the monitoring," Nov. 2010. [Online]. Available: http://www.honeynet.org/papers/KYT_qebek.

[53] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Security Privacy*, vol. 5, no. 2, pp. 32–39, Mar. 2007.

[54] X. Jiang and X. Wang, " Out-of-the-box monitoring of VM-based high-interaction honeypots," in *Recent Advances in Intrusion Detection*, vol. 4637, C. Kruegel, R. Lippmann, and A. Clark, Eds. Berlin, Germany: Springer, 2007, pp. 198–218.

[55] LibVMIProject, "LibVMI," 2015. [Online]. Available: http://libvmi.com/.

[56] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proc. Netw. Distrib. Syst. Security Symp.*, 2003, pp. 191–206.

[57] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An architecture for secure active monitoring using virtualization," in *Proc. 2008 IEEE Symp. Security Privacy*, May 2008, pp. 233–247.

[58] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through VMM-based 'out-of-the-box' semantic view reconstruction," in *Proc. 14th ACM Conf. Comput. Commun. Security*, 2007, pp. 128–138.

[59] J. Pfoh, C. Schneider, and C. Eckert, " Nitro: Hardware-based system call tracing for virtual machines," in *Advances in Information and Computer Security*, vol. 7038, T. Iwata and M. Nishigaki, Eds. Berlin, Germany: Springer, 2011, pp. 96–112.

[60] D. Srinivasan and X. Jiang, "Time-traveling forensic analysis of VM-based high-interaction honeypots," in *Proc. 7th Int. Conf. Security Privacy Commun. Netw.*, SecureComm 2011, London, U.K., Sep. 7–9, 2011, pp. 209–226, revised selected papers.

[61] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," in *Proc. 2011 IEEE Symp. Security Privacy*, May 2011, pp. 297–312.

[62] 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 *Proc. 30th Annu. Comput. Security Appl. Conf.*, 2014, pp. 386–395.

[63] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of internet background radiation," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, 2004, pp. 27–40.

[64] M. Bailey, E. Cooke, F. Jahanian, N. Provos, K. Rosaen, and D. Watson, "Data reduction for the scalable automated analysis of distributed darknet traffic," in *Proc. 5th ACM SIGCOMM Conf. Internet Meas.*, 2005, pp. 21–21.

[65] G. Portokalidis and H. Bos, "SweetBait: Zero-hour worm detection and containment using low-and high-interaction honeypots," *Comput. Netw.*, vol. 51, no. 5, pp. 1256–1274, 2007.

[66] W. Cui, V. Paxson, N. C. Weaver, and Y. H. Katz, "Protocol-independent adaptive replay of application dialog," in *Proc. 13th Annu. Netw. Distrib. Syst. Security Symp.* (NDSS), Feb. 2006, pp. 1–15.

[67] X. Jiang and D. Xu, "Collapsar: A VM-based architecture for network attack detention center," in *Proc. USENIX Security Symp.*, 2004, pp. 15–28.

[68] "Know your enemy: Honeywall cdrom," May 2005. [Online]. Available: http://old.honeynet.org/papers/cdrom/.

[69] É. Alata, I. Alberdi, V. Nicomette, P. Owezarski, and M. Kaâniche, " Internet attacks monitoring with dynamic connection redirection mechanisms," *J. Comput. Virol.*, vol. 4, no. 2, pp. 127–136, 2008.

[70] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," in *Proc. 12th Annu. Netw. Distrib. Syst. Security Symp.*, 2005, pp. 1–17.

[71] C. Kreibich and J. Crowcroft, "Honeycomb: Creating intrusion detection signatures using honeypots," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 51–56, Jan. 2004.

[72] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. 9th EAI Int. Conf. Bio-inspired Inf. Commun. Technol. (Formerly BIONETICS)*, 2016, pp. 21–26.

[73] R. Sekar *et al.*, "Specification-based anomaly detection: A new approach for detecting network intrusions," in *Proc. 9th ACM Conf. Comput. Commun. Security*, 2002, pp. 265–274.

[74] 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.

[75] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with SDN: A feasibility study," *Comput. Netw.*, vol. 85, no. C, pp. 19–35, 2015.

[76] R. G. Berthier, "Advanced honeypot architecture for network threats quantification," Ph.D. dissertation, Univ. Maryland at College Park, College Park, MD, USA, 2009, AAI3359256.

[77] H. Welte and P. N. Ayuso, "The libnetfilter_queue project," 2014. [Online]. Available: http://www.netfilter.org/projects/libnetfilter_queue/.

[78] Y.-D. Lin, T.-B. Shih, Y.-S. Wu, and Y.-C. Lai, "Secure and transparent network traffic replay, redirect, and relay in a dynamic malware analysis environment," *Security Commun. Netw.*, vol. 7, no. 3, pp. 626–640, 2014.

[79] T. Lengyel, J. Neumann, S. Maresca, and A. Kiayias, " Towards hybrid honeynets via virtual machine introspection and cloning," in *Network and System Security*, vol. 7873, J. Lopez, X. Huang, and R. Sandhu, Eds. Berlin, Germany: Springer, 2013, pp. 164–177.

[80] W. Fan, D. Fernndez, and Z. Du, " Adaptive and flexible virtual honeynet," in *Mobile, Secure, and Programmable Networking*, vol. 9395, S. Boumerdassi, S. Bouzefrane, and R. Renault, Eds. New York, NY, USA: Springer, 2015, pp. 1–17.

[81] W. Fan, Z. Du, D. Fernández, and X. Hui, "Dynamic hybrid honeypot system based transparent traffic redirection mechanism," in *Proc. 17th Int. Conf. Inf. Commun. Security*, Beijing, China, Dec. 9–11, 2015, pp. 311–319.

[82] W. Fan and D. Fernández, "A novel SDN based stealthy TCP connection handover mechanism for hybrid honeypot systems," in *Proc. IEEE 3rd Conf. Netw. Softwarization*, Bologna, Italy, Jul. 2017, pp. 1–9.

[83] C. Hecker and B. Hay, "Automated honeynet deployment for dynamic network environment," in *Proc. 46th Hawaii Int. Conf. Syst. Sci.*, Jan. 2013, pp. 4880–4889.

[84] M. Zalewski, "pof v3," 2012–2014. [Online]. Available: http://lcamtuf.coredump.cx/p0f3/.

[85] G. Lyon, "Namp," 2015. [Online]. Available: http://nmap.org/.

[86] R. McGrew and R. B. Vaughn jr, "Experiences with honeypot systems: Development, deployment, and analysis," in *Proc. 39th Annu. Hawaii Int. Conf. Syst. Sci.*, Jan. 2006, vol. 9, pp. 220–229.

[87] 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.

[88] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *J. Comput. Security*, vol. 19, no. 4, pp. 639–668, Dec. 2011.

[89] "Know your enemy: Genii honeynets," May 2005. [Online]. Available: http://old.honeynet.org/papers/gen2/.

[90] C. Leita and M. Dacier, "SGNET: A worldwide deployable framework to support the analysis of malware threat models," in *Proc. 7th Eur. Dependable Comput. Conf.*, May 2008, pp. 99–109.

[91] S. Li and R. Schmitz, "A novel anti-phishing framework based on honeypots," in *Proc. 2009 eCrime Res. Summit*, Sep. 2009, pp. 1–13.

[92] X. Fu, B. Graham, D. Cheng, R. Bettati, and W. Zhao, "Camouflaging virtual honeypots," Dept. Comput. Sci., Texas A&M Univ., College Station, TX, USA, Tech. Rep. 2005-7-3, Jul. 2005.

[93] H. Wang and Q. Chen, "Dynamic deploying distributed low-interaction honeynet," *J. Comput.*, vol. 7, no. 3, pp. 692–698, 2012.

Wenjun Fan received the Ph.D. degree in telematics engineering from the Universidad Politécnica de Madrid, Madrid, Spain, in 2017.

He is a Postdoctoral Researcher of cyber security with the University of Kent, Canterbury, U.K. His research interests include cyber security, software-defined networks, cloud computing, and machine learning.

Zhihui Du (M'00–SM'16) received the B.E. degree from the Computer Department, Tianjin University, Tianjin, China, in 1992, and the M.S. and Ph.D. degrees in computer science from Peking University, Beijing, China, in 1995 and 1998, respectively.

From 1998 to 2000, he was with Tsinghua University, Beijing, China, as a Postdoctoral Researcher. Since 2001, he has been with Tsinghua University, as an Associate Professor with the Department of Computer Science and Technology. His research areas include high-performance computing and grid computing.

David Fernández received the M.S. degree in telecommunications engineering and Ph.D. degree in telematics engineering from the Universidad Politécnica de Madrid, Madrid, Spain, in 1988 and 1993, respectively.

He is an Associate Professor of computer networks with the Technical University of Madrid, Madrid. His research interests focus on software-defined networks, network virtualization, cloud computing datacentres technologies, and network security.

Víctor A. Villagrá received the M.S. degree in telecommunications engineering and Ph.D. degree in telematics engineering from the Universidad Politécnica de Madrid, Madrid, Spain, in 1989 and 1994, respectively.

He is an Associate Professor of telematics engineering with the Technical University of Madrid, Madrid. He authored a textbook about security in telecommunication networks. His research interests focus on network security, network management, and advanced services design.