Review and Analysis of Cowrie Artefacts and Their Potential to be used Deceptively

Warren Z Cabral, Craig Valli, Leslie F Sikos, Samuel G Wakeling

Abstract—**Honeypots are progressively becoming a fundamental cybersecurity tool to detect, prevent and record new threats and attack methodologies used by attackers to penetrate systems. The current technology is advancing rapidly; with the use of virtualisation, and most recently, virtual containers, the deployment of honeypots has become increasingly easier. A varied collection of open source honeypots such as Cowrie are available today, which can be easily downloaded and deployed within minutes—with default settings. Cowrie is a medium-interaction secure shell (SSH) and Telnet honeypot intended to log brute force and shell interaction attacks. However, the current issue with the default Cowrie configuration is that it is easily detected by adversaries using automated scripts and tools. To increase Cowrie's deceptive capabilities, it is essential to understand, modify, and leverage all capabilities of the honeypot. However, this process is complex, because there are no standard frameworks to interpret the artefacts used by the Cowrie honeypot and how these artefacts link to the type of deceptiveness presented to the cyber-attacker. Therefore, there is a need for some type of infrastructure that can interpret these basic deception techniques and tools, and later developing them into feasible cybersecurity defence mechanisms. This study pursues to develop an understanding about its capabilities, and how these capabilities can be used to bait attackers. The resulting annotations can help cybersecurity defenders better understand the effectiveness of the Cowrie artefacts and how they can be used deceptively.**

Keywords-Cybersecurity, Honeypots, Cowrie, SSH, Secure Shell, Telnet, Deception

I. Introduction

The number of cybersecurity threats and attacks is increasing at an alarming rate [1], and quite often the standard security measures, such as access control systems, intrusion detection systems, and firewalls, are insufficient to secure computers and information security systems from potential attackers [2]. Thus, we must continuously learn and analyze new threats and attack methodologies to counter and defend ourselves from cyber-attackers. Using honeypots is increasingly common for studying how attackers penetrate into information security systems [3].

Warren Z Cabral, Craig Valli and Leslie F Sikos are with the Cyber Security Co-operative Research Centre and with the School of Science Edith Cowan University, WA 6027, Australia (e-mail: {w.cabral, c.valli, l.sikos}@ecu.edu.au)

Samuel G Wakeling is with the School of Science Edith Cowan University, WA 6027, Australia (e-mail: s.wakeling@ecu.edu.au)

These are fictitious computer systems that deceive attackers by presenting themselves as actual computer systems containing sensitive information [4]. They have been in existence and are being used increasingly since the turn of this century [5].

Honeypots are purposefully designed in a way that they can be attacked by malicious attackers (whether automated or human-directed), who can gain access to seemingly sensitive information by making them believe that the system they are attacking is real [6]. Unbeknownst to the attacker, their attack pattern is recorded and can be analyzed. This enables cybersecurity analysts to learn about new attack vectors and how to better defend themselves from future attacks [7].

Honeypots are deceptive systems that make it harder for attackers to penetrate actual production systems and also reduce attacker effectiveness while making it easier for cyber-defenders to detect, understand, and reconstruct attackers' attacks and intents. It is quite clear, however, that modern-day honeypots need a certain degree of interaction and deception to be effective in cybersecurity.

This research focuses on the Cowrie honeypot, a medium-interaction secure shell (SSH) and Telnet honeypot used to record brute force attacks and SSH requests from attackers. Cowrie is an open source honeypot that functions on a Python codebase, which is maintained and publicly available on GitHub [8]. It contains a standard configuration file that encompasses artefacts/variables for its deployment. These artefacts include, but are not limited to, a hostname, IP addresses, network services, applications, and specific fingerprint information (for example, a certain SSH string returned to the connecting SSH client).

Modern honeypots such as Cowrie are still in their infancy but are moderately operative in their default configuration state. This state, however, is not a viable option for security analysts, because Cowrie can easily be detected by attackers. A honeypot is only plausible and effective if its attackers cannot realize that they are actually connecting to a honeypot [9].

A current issue encompassing Cowrie is that malicious attackers have automated scripts that can detect it [10]. Moreover, there are tools such as Sentient Hyper-Optimized Data Access Network (SHODAN), which is a scanning service that can routinely identify deployed honeypots by type, internet protocol (IP), and location [11]. For example, the Cowrie Telnet service can be detected by a simple Network Mapper (NMAP) scan [12]. A secondary issue is that there are no frameworks to interpret the different configuration options available in Cowrie. Other methods for detecting honeypots are log overflow attacks, using honeypots in virtual environments and operating system (OS) fingerprinting.

978-1-7281-5584-5/19/$31.00 ©2019 IEEE
DOI 10.1109/CSCI49370.2019.00035

Furthermore, the configuration options for most honeypots is complex and must be modified in an integrated and granular fashion to make it simpler, but at the same time achieve a high degree of deceptiveness when presented to attackers [13]. To emphasize this point, the present level of deception with Cowrie is limited due to the absence of a standard framework that could classify and make sense of the different configuration options available, and how this functionality can be modified and improved at will to increase Cowrie's deceptive capabilities [14]. Therefore, this research focuses on reviewing and analyzing the default Cowrie configuration file to understand the different types of artefacts, such as SSH and Telnet options, and other input/output variables present in the Cowrie configuration file. These artefacts can further be classified on the type of deception used by each variable, namely, mimicry, masking, and hiding. Furthermore, they can be used later to develop a framework to estimate the level of deception achieved through testing and verification of honeypot outputs using selected options.

II. LITERATURE REVIEW

The main goal of honeypots are their deceptive capabilities and the level of interaction they have with attackers.

The American Heritage dictionary of the English language [15] states that

"Deception is defined as the act of deceit. Deceit is defined as deception".

According to Cohen [16], the basic understanding behind deception is taking advantage of and exploiting cognitive networks by thoroughly exploiting identified system flaws. Analysts have made advances in pinpointing cognitive flaws and procedures for their exploitations. A straightforward approach for honeypots and decoys is by exposing vulnerability information, which might lure attackers into attacking the system's "weaknesses." In his writing, Cohen [16] quoted Torres: *"for a honeypot to work, it needs to have some honey."*

The basic requirement for honeypots is the presentation of a believable vulnerability or possible attack vector. The goal is that these vulnerabilities are exploited by attackers thinking they are actual systems. The honeypot then interacts with the attacker and records the interaction for later analysis. Information security analysts then examine these interactions to learn about new tools, tactics, and processes used by attackers. This resulting analysis then helps them create actionable cybersecurity intelligence and then remediate and secure information security of the infrastructure.

There needs to be a carefully set balance between the degree of deception and the vulnerabilities present on the honeypot. If too many vulnerabilities are present, the attacker may realize that this is a trap; if there are too few vulnerabilities, the attacker may not attack the honeypot system. Therefore, it is important to understand about the principles of deception and the different types of deceptive approaches that can be used by honeypots: mimicry, masking, and hiding.

Mimicry is a type of deception wherein the system or file is programmed to perform or copy the functions of another system. For example, we can provide a false architecture for the emulation of the Cowrie shell instance. According to Shi, et al. [17], current passive honeypots can be configured to be used entirely as "Mimicry honeypots." Though the stage for mimicry honeypots has not been defined; they can be used to perceive and adapt the change of network services to perform better concealment in order to deceive the attacker in believing the system they are attacking is the real system.

Data masking (or masking for short) is a form for data obfuscation. Obfuscation refers to making or modifying data to ensure that it is unclear or concealed [18]. The primary function of data masking is to ensure that data, which may be sensitive and can be viewed without authorization, is concealed, altered, or masked. Data masking can be used in the following types of situations: protecting data from third-party organizations, human error, and testing purposes. In the same manner data masking can be used as a tool of deception by masking information from attackers. For example, we defined configuring a variable to display a fake IP address or a fake SSH version number as masking. This masking will mislead attackers into believing they are accessing a legitimate service or system.

Data hiding is used in deception to hide variables and attributes from attackers. The key difference between hiding and masking is that in masking the data is visible whereas for the latter the data is not visible [19]. For example, passwords in Windows are masked and in Ubuntu Server are hidden.

A single type of deception technique will possibly not be the only viable means for preventing attackers from detecting honeypot services. A combination of one or more deceptive techniques must be used to achieve a feasible deployment platform for honeypots to make them act, function, and conceal themselves as a real system. Based on the level of interactions and deceptive capabilities that honeypots offer to malicious attackers, they are classified as low-interaction, medium-interaction, and high-interaction honeypots [20]. Low-interaction honeypots provide a small and limited interaction to attackers. They collect limited information, such as low-level connection logs and traffic information. However, low-interaction honeypots are easy to implement and have only a low impact on the network and system infrastructure, although, they have very poor deceptive capabilities.

The second group of honeypots are the medium-interaction honeypots. As the name implies, they provide medium interaction, which means a better chance of interaction with an attacker to collect more detailed information as compared to low-interaction honeypots. They have significantly more deceptive capabilities compared to low-interaction honeypots. Cowrie is an example of a medium-interaction honeypot. It has a medium-interaction capability in the sense that it provides some functions demonstrated by the real system, such as acting as an SSH server [21]. Medium-interaction honeypots like Cowrie use an approach called emulation. Emulation is a process through which the honeypot provides a service that acts like the expected "real" system. For example, being an SSH honeypot, Cowrie emulates the behavior of an SSH server and a Linux operating system, thereby allowing an attacker to log into the system, but every command is emulated (the attackers' commands will only be logged, not actually executed). Due to this, it is very easy for an attacker to detect whether a system is an emulated honeypot or a legitimate system [22]. As mentioned earlier by Rabadia and Valli [14], the functionality and deceptiveness of the Cowrie honeypot can be modified and improved. Additionally, it is important to know that most low-interaction and medium-interaction honeypots are open source and easily accessible on GitHub or other websites. The disadvantage of such honeypots is the time and resource requirement to configure them accordingly to be useful in attack scenarios.

The final group is the high-interaction honeypots, which provide real services, entire operating systems, and a high level of deception to attackers [23]. These are usually commercially available honeypots for which licenses must be purchased—see the Windows-based honeypot KFSensor [24], for example. They are easy to setup and configure, and they also allow attackers to have the highest level of interaction and make it possible for security analysts to gather as much attack information as possible. However, the risk factor is high, because high-interaction honeypots can be used by attackers to attack other legitimate systems [22, 25]. Therefore, it can be concluded that for a honeypot to be liable and effective, a certain degree of deception and interaction is needed.

III. Proposed Research Method

As mentioned earlier, Cowrie's configuration files contain several variables for deployment, including file locations, hostnames, server names, SSH and Telnet options, operating system information, listening addresses, and output plugins. The default configuration of Cowrie has poor deceptive capabilities, because an end-user running the honeypot with default configuration settings, i.e., making no changes to the configuration files and variables, the emulated PowerShell system and the fake file system would be susceptible for attackers to detect the honeypot with ease.

Renaming the hostname, server name, configuring the endpoints to listen on port 22, modifying the fake architecture to operate within tends to make the honeypot less deceptive but not completely viable for operational deployment. A significant number of other variables and processes must be inspected and reviewed to completely understand what each variable does and how these variables can be configured and used to potentially increase the overall deceptiveness of the Cowrie honeypot. Similarly, Cowrie's emulated PowerShell session and fake file system configuration files can be inspected, modified, and improved to strengthen its deceptive capabilities to bait attackers into believing that they are attacking a real system and not a honeypot. Cowrie is an important and viable open source honeypot, because it supports numerous output plugins and capabilities, such as the SSH and Telnet proxies, output plugins for JSON log files, Cuckoo sandbox, the ELK Stack, Graylog, Splunk, and SQL [8]. However, by default some of these functionalities are not enabled, while others are poorly configured.

When configured correctly, SSH and Telnet proxies can function as fully-fledged backend environments, as opposed to the emulated shell environment traditionally provided by Cowrie [8]. This enables Cowrie to function as a high-interaction honeypot with an actual backend environment in which attackers can execute any Linux command. The default configuration of the backend provides a pool configuration, which is not ideal. This is because a backend pool configuration uses a collection of virtual machines. According to [26], honeypots executed on virtual machines like VMWare can be detected due to

A. *The network MAC address that is assigned to the network card.* The first three octets on the vendor part of the MAC address on a VMWare virtual machine interface always has a value of the form 00-05-69-xx-xx-xx, 00-0C-29-xx-xx-xx, or 00-50-56-xx-xx-xx. This implies that if an attacker is using a Windows machine a simple `ipconfig/all` scan or `ifconfig -a` on a UNIX-like machine can detect the above MAC address.

B. *An open I/O backdoor left by the VMware developers for the runtime configuration of virtual machines.* An attacker can potentially execute assembly code to access this backdoor, check to determine whether the command was executed successfully, which confirms that they are actually attacking a VMware honeypot rather than a production system.

Therefore, it is essential to configure the environment to be fully compatible with the Cowrie honeypot executed in a simple backend configuration.

For this research study, a Cowrie instance was downloaded and set up from the official GitHub repository. The instance was executed in a Ubuntu 18.04 LTS Server virtualized using VMware Workstation 15.5 Pro. Cowrie's default configuration file is located in `/etc/cowrie.cfg.dist`. The variables of the configuration file were distinguished based on their usage characteristics. For example, the variable `arch` is used to identify the architecture or the operating system on which the honeypot is running. This variable can be modified to display or mimic a false architecture, thereby misleading attackers from detecting the real architecture. Therefore, this variable is classified as mimicry. Likewise, the variable `fake_addr` is used to disguise a fake address as the address of the incoming connection. Because this variable conceals the real IP address of the incoming connection, it is classified as masking. All the other variables of the configuration file were examined similarly to determine the functionality of each variable and what type of deception is used (mimicry, masking, or hiding). Based on this analysis, the most significant variables were identified and conjugated in Table 1. The proposed research methodology aims to modify and determine Cowrie variables to understand their functionality and create a pivot for future research to understand how these variables, files and systems can be potentially configured in regards to the research background mentioned in this paper to make the Cowrie honeypot more deceptive when presented to attackers.

IV. RESULTS

The following table reviews the most significant variables found and analyzed in Cowrie's default configuration file. These variables were categorized (based on the order as it appears in the default Cowrie configuration file) on the extent to which they can be configured and improved to increase the overall deceptiveness of the Cowrie honeypot when presented to an attacker when deployed in an ideal environment [8]. The variables have been further classified based on their deception type which are data mimicry, data masking and data hiding.

TABLE I. THE MOST SIGNIFICANT VARIABLES ANALYSED IN THE COWRIE CONFIGURATION FILE CLASSIFIED BY DECEPTION TYPE.

Variable	Usage and Example	Deception
`sensor_name`	The sensor name is used to identify the current Cowrie instance. This variable should be changed and then hidden from unauthorized users. **`sensor_name = ecuserver01`**	**Data masking/ data hiding**
`hostname`	The hostname of the Cowrie honeypot instance. The output of this variable is displayed by the shell prompt of the virtual environment. The output of this variable should be changed to mislead attackers into thinking they are attacking a real server. `hostname = cowrie_servo5`	**Data masking**
`fake_addr`	The fake IP address to be displayed as the address of an incoming connection. `fake_addr = 192.168.66.254`	**Data masking**
`internet_facing_ip`	The IP address on which the machine running the Cowrie instance is reachable from the Internet. This variable can be masked/unmasked to make it reachable to attackers. `internet_facing_ip = 9.9.9.9`	**Data masking**
`arch`	The output of this variable is used to display and act as a fake operating system. A user can modify this variable accordingly to which Linux system they want to deploy. `arch = linux-x64-lsb`	**Data mimicry**
`processes`	The location for the file that contains the PowerShell commands that can be used by the attacker during honeypot sessions. The file `cmdoutput.json` can be configured and enhanced to mimic a fully operational PowerShell session. The file is located in `share/cowrie/cmdoutput.json`	**Data mimicry**
`operating_system`	The operating system being used by the Cowrie emulation shell. The output of this variable can be modified to act as another operating system. `operating_system = GNU/Linux`	**Data mimicry**
`kernel_version`	The version number of the kernel that can be used to disguise or act as another version number to mislead attackers. `kernel_version = 3.2.0-4-amd64`	**Data mimicry**
`version`	These variable disguises the Cowrie honeypot and prevents it from getting detected by simple SSH scans. `version = SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2`	**Data masking**
`listen_endpoints`	The endpoints to listen for an incoming SSH or an incoming Telnet connection. By default, the Cowrie service runs on port 2222. To mislead attackers, it is possible to configure the service to run on port 22 via configuring iptable rulesets. `listen_endpoints = tcp:22:interface=0.0.0.0`	**Data masking**
`out_addr`	The IP address used to bind outgoing connections. This variable is used by the `wget` and `curl` commands. The output depends on which IP address the end-user wishes to bind to. `out_addr = 0.0.0.0`	**Data masking**

`filesystem`	The `filesystem` variable points to the location of the fake file system. By default, the fake file system resembles a Debian 5.0 operating system and has the ability to add or remove files. This file can be configured to improve Cowrie's emulation options. The file is stored in Python pickle format and is located at `share/cowrie/fs.pickle`	**Data mimicry**
`backend`	The backend proxy configuration for SSH and Telnet configurations. Ideally, a `simple` backend must be provided, i.e., a system provided by the end-user, such as a server to avoid honeypot detection in `pool` configuration, which uses a collection of virtual machines. `backend = simple`	**Data mimicry**
`backend_ssh_host`	The hostname displayed by the backend SSH host. The default hostname "localhost" can be changed to match an organisation's naming convention to make it look genuine to attackers. `backend_ssh_host = ecu_ssh1`	**Data masking**
`backend_ssh_port`	The port number to which the backend host will listen for SSH connections. Default SSH ports can be changed to suit the honeypot's requirements. `backend_ssh_port = 22`	**Data masking**
`backend_telnet_host`	The hostname displayed by the backend Telnet host. The default hostname "localhost" can be changed to match the organisation's naming convention to make it look genuine to attackers. `backend_telnet_host = ecu_telnet1`	**Data masking**
`backend_telnet_port`	The port number to which the backend host will listen for Telnet connections. The default Telnet ports can be changed to suit the honeypot's requirements. `backend_telnet_port = 23`	**Data masking**
`ciphers`	The command for stating the cipher algorithms used. The output of this variable can be configured to include/exclude necessary ciphers for operation. `ciphers = aes256-cbc,aes192-cbc,aes128-cbc`	**Data masking/ data hiding**
`macs`	The command for stating the MAC (Message Authentication Code) algorithms used. The output of this variable can be configured to include/exclude necessary MACs. `macs = hmac-sha2-512,hmac-sha2-384,hmac-sha2-56,hmac-sha1,hmac-md5`	**Data masking/ data hiding**

a. The variables analyzed as seen in Cowrie's default configuration type. The variables have been categorized based on deception type which are data mimicry, data masking and data hiding.

V. FUTURE RESEARCH

At the time of writing, research is being pursued on the numerous tools and strategies used by attackers to detect honeypots, including Cowrie. The results arising from this investigation will be used to develop a pivot for further research on the following:

a. The different types of artefacts, such as SSH and Telnet options, and other input/output variables present in the Cowrie configuration file.
b. How these variables can be used and modified to increase the deceptiveness of the Cowrie honeypot when presented to attackers.
c. To create a platform for the automated deployment of the Cowrie honeypot with proposed configurations.

VI. SUMMARY AND CONCLUSION

This paper summarizes information about the Cowrie honeypot and the different types of deceptive features available, together with their advantages for being used as tools for tricking attackers into attacking fake (honeypot) information security infrastructures.

To make it harder for attackers to detect that they are connecting to a Cowrie honeypot, the default configuration file must be modified. However, this is a non-trivial task, because at the time of writing no standard framework is available that can process the artefacts or variables used by the honeypot and how these artefacts link to the type of deceptiveness presented to the cyber-attacker. This urges the development of an infrastructure that can interpret these basic deception techniques and tools (which can later be implemented in efficient cybersecurity defence mechanisms), which is the very purpose of this research paper. Further work in this field will advance the outcomes of this research to increase our understanding of deception techniques, and can be used as a pivot for planning, configuring, and testing the default Cowrie configuration file to increase its deceptive capabilities when presented to attackers.

References

[1] G. K. Sadasivam, C. Hota, and B. Anand, "Honeynet Data Analysis and Distributed SSH Brute-Force Attacks," in *Towards Extensible and Adaptable Methods in Computing*, S. Chakraverty, A. Goel, and S. Misra Eds.: Singapore: Springer, 2018, pp. 107-118. doi:10.1007/978-981-13-2348-5_9.

[2] C. Wang and L. Zhuo, "Cyber Deception: Overview and the Road Ahead," *IEEE Security & Privacy,* vol. 16, no. 2, 2018, doi: 10.1109/MSP.2018.1870866.

[3] P. Sokol, J. Mísek, and M. Husák, "Honeypots and honeynets: issues of privacy," *EURASIP Journal on Information Security,* vol. 2017, p. Article 4, 2017, doi: 10.1186/s13635-017-0057-4.

[4] N. Bhagat and B. Arora, "Honeypots and Its Deployment: A Review," V. Rathore, M. Worring, D. Mishra, and S. Maheshwari, Eds., 2019: Singapore: Springer in Emerging Trends in Expert Applications and Security, pp. 505-512, doi: 10.1007/978-981-13-2285-3_59.

[5] L. Spitzner, "Honeypots: Catching the insider threat," in *Proceedings of the 19th Annual Computer Security Applications Conference.* 2003: IEEE, pp. 170-179. doi:10.1109/CSAC.2003.1254322.

[6] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection*. Upper Saddle River, NJ: Addison-Wesley, 2008.

[7] T. S. Toland, S. Kollmannsperger, J. B. Brewton, and W. B. Craft, "Using Sports Plays to Configure Honeypots Environments to form a Virtual Security Shield," in *Computer and Network Security Essentials*, K. Daimi Ed.: Cham: Springer, 2018, pp. 189-204. doi:10.1007/978-3-319-58424-9_11.

[8] M. Oosterhof, "Cowrie SSH and Telnet Honeypot: github," 2019. [Online]. Available: https://github.com/cowrie/cowrie.

[9] M. Dornseif, T. Holz, and U. S. Müller, "Honeypots and limitations of deception," 2005.

[10] M. Rajain, "Honeybee: Application that will detect honeypots!," 2018. [Online]. Available: https://github.com/mohitrajain/honeybee.

[11] J. Matherly, "Shodan - (Sentient Hyper-Optimised Data Access Network)," 2019.

[12] T. Nishinaga. "Cowrie telnet service detected by nmap 7.70: github." 2018. [Online]. Available: https://github.com/cowrie/cowrie/issues/962.

[13] C. Kraenzel, J. R. Linton, and R. Mani, "Dynamically Configuring A Honeypot," ed: Google Patents, 2019.

[14] P. Rabadia and C. Valli, *Finding evidence of wordlists being deployed against SSH honeypots – implications and impacts*. (2014). Edith Cowan University, Research Online, Perth, Western Australia. doi: 10.4225/75/57b3e7d5fb882.

[15] The American Heritage dictionary of the English language, *The American Heritage dictionary of the English language*. Houghton Mifflin Harcourt, 2016.

[16] F. Cohen, "The use of deception techniques: Honeypots and decoys," *Handbook of Information Security,* vol. 3, no. 1, pp. 646-655, 2006.

[17] L. Shi, L. Jiang, D. Liu, and X. Han, "Mimicry honeypot: a brief introduction," in *Proceedings of the 8th International Conference on Wireless Communications, Networking and Mobile Computing*, 2012: IEEE, pp. 1-4, doi: 10.1186/s13635-017-0057-4.

[18] X. Sun, Y. Wang, J. Ren, Y. Zhu, and S. Liu, "Collecting internet malware based on client-side honeypot," in *Proceedings of the 9th International Conference for Young Computer Scientists*, 2008: IEEE Computer Society, pp. 1493-1498, doi: 10.1109/ICYCS.2008.257.

[19] Apigee. "Data masking and hiding." 2019. [Online]. Available: https://docs.apigee.com/api-platform/security/data-masking.

[20] S. Z. Melese and P. S. Avadhani, "Honeypot System for Attacks on SSH Protocol," *International Journal of Computer Network and Information Security (IJCNIS),* Journal Article vol. 8, no. 9, pp. 19-26, 2016, doi: 10.5815/ijcnis.2016.09.03. Modern Education and Computer Science Press.

[21] M. T. Qassrawi and Z. Hongli, "Deception methodology in virtual honeypots," in *Proceedings of the 2nd International Conference on Networks Security, Wireless Communications and Trusted Computing*, 2010, vol. 2: IEEE, pp. 462-467, doi: 10.1109/NSWCTC.2010.266.

[22] F. Zhang, S. Zhou, Z. Qin, and J. Liu, "Honeypot: a supplemented active defense system for network security," in *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2003: IEEE, pp. 231-235, doi: 10.1109/PDCAT.2003.1236295.

[23] K. Papazis and N. Chilamkurti, "Detecting indicators of deception in emulated monitoring systems," in *Service Oriented Computing and Applications*, vol. 13, no. 1: London: Springer, 2019, pp. 17-29. doi: 10.1007/s11761-018-0252-2.

[24] KeyFocus, "KFSensor: Advanced Windows Honeypot System," 2019. [Online]. Available: http://www.keyfocus.net/kfsensor/.

[25] A. Almutairi, D. Parish, and R. Phan, "Survey of high interaction honeypot tools: Merits and shortcomings," in *Proceedings of the 13th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting, PGNet2012. PGNet*, 2012.

[26] S. Innes and C. Valli, "Honeypots: How do you know when you are inside one?," in *Proceedings of the 4th Australian Digital Forensics Conference*, Edith Cowan University, Perth Western Australia, 2006, doi: 10.4225/75/57b131f0c7052.