Probabilistic Estimation of Honeypot Detection in Internet of Things Environment

Oleg Surnin*, Fatima Hussain†, Rasheed Hussain*, Svetlana Ostrovskaya*,
Andrey Polovinkin*, JooYoung Lee*, and Xavier Fernando †

* Institute of Information Systems, Innopolis University, Innopolis, Russia.
Email: {o.surnin,r.hussain,j.lee}@innopolis.ru
† Ryerson University, Canada.
Email: fatima.hussain@ryerson.ca

***Abstract*—With the emergence of the Internet of Things (IoT) and the increasing number of resource-constrained interconnected smart devices, there is a noticeable increase in the number of cyber security crimes. In the face of the possible attacks on IoT networks such as network intrusion, denial of service, spoofing and so on, there is a need to develop efficient methods to locate vulnerabilities and mitigate attacks in IoT networks. Without loss of generality, we consider only intrusion-related threats to IoT. A honeypot is a system used to understand the potential dynamic threats and act as a proactive measure to detect any intrusion into the network. It is used as a trap for intruders to control unauthorized access to the network by analyzing malicious traffic. However, a sophisticated attacker can detect the presence of a honeypot and abort the intrusion mission. Therefore it is essential for honeypots to be undetectable. In this paper, we study and analyze possible techniques for SSH and telnet honeypot detection. Moreover, we propose a new methodology for probabilistic estimation of honeypot detection and an automated software implemented this methodology.**

***Index Terms*—Internet of Things, SSH Honeypots, Intrusion, Security, Attacks**

I. Introduction

Internet of things (IoT) networks employ IoT devices that usually have open network ports for interaction between the physical and virtual worlds. This phenomenon makes the network vulnerable to potential cyber attackers [1]. On the other hand, the nature of cyber crimes have been drastically changed and now the target is IoT devices rather than traditional PC or mobile phones. According to the findings of an ISACA research report on the state of cybersecurity (2017)[1], the majority of the surveyed organizations expect to be hit with cyber attacks. Among them, 46% are confident that they defend against such attacks. One of the most popular methods to protect important data and resources is set up different kinds of honeypots used as a security control mechanism for capturing and analyzing unauthorized network activity.

Honeypots can provide services from setting up a single dummy system to the creation of the entire network with a wide range of services and operating systems [2]. These are used to emulate various network services and develop a trap or loophole in the system; such that attackers try to compromise it and get noticed. The main goal of honeypots is to monitor activities and log the received data. Afterwards, this information is used to design a network security system and to prevent future intrusions [3]. Even though honeypot is a passive approach, it can still efficiently find zero-day exploit attempts at the early stage of massive attacks.

Honeypots can be divided into two broad categories: research and production honeypots. Production honeypots are easily deployable and are used to detect and mitigate attacks and improve the network security. While research honeypots are complex and have the ability to capture more detailed information such as, but not limited to, tactics, techniques, and procedures used by hackers and investigate the threats. Without loss of generality, in this work we focus on honeypots used in the Internet of Things (IoT) networks. IoT honeypots are heterogeneous in nature, and monitor specific services like video streaming and private protocols [4] in addition to remote management and web services like HTTP and FTP. To this end, various IoT honeypots like IoTPOT, SIPHON, and multipurpose honeypot are available in the market [5]. These are based on architectures like MIPS, ARMS, PowerPC focusing on various protocols like HTTP, SSH, Telnet and so on.

Honeypots are designed to be exploited, hacked, infected with malware, and generally abused by a malicious third party. However, if attackers are able to detect that system under attack is dummy and not real, the role of honeypot is compromised. Therefore, it is very important to make honeypot detection as hard as possible. Our main objective is to estimate the probabilistic detection of IoT honeypots.

1) Research Motivation and Contribution: Despite all of the advantages, honeypots still have problems that are often exploited by the attackers. What happens if the attackers are able to detect that they are in a honeypot instead in a real system? How developers and system users can prevent such detection? This research intends to answer these questions by studying, analyzing and testing possible techniques for honeypot detection, in a real environment. Our research is focused on SSH honeypots and associated challenges, specifically, with their implementations. Summary of our contributions is listed as below:

[1]https://cybersecurity.isaca.org/csx-resources/state-of-cyber-security-2017

978-1-5386-9223-3/19/$31.00 ©2019 IEEE

- analyze and select suitable methods for honeypot detection,
- develop a proof-of-concept for automated honeypot detection,
- provide recommendations for honeypot developers and users.

The rest of the paper is organized as follows. Section II presents related work and section III describes research methodology and implementation of our honeypot detection mechanism. It includes both the modification of existing implantation followed by its automation. Experimentations and analysis of captured data are discussed in section IV. Conclusions are drawn in section V.

II. Related work

Honeypots do not only allow to simulate multiple virtual hosts, numerous TCP/IP stacks and network topologies; but also provide several features to set up real servers, identify cyber attacks and assign hackers a passive-fingerprint. Since honeypots are widely spread over the entire network, their detection becomes very important for both the attackers and users. There are many methods to analyze the creditability of a honeypot. However, most of them are focused on the detection of a virtual environment and low-level interaction honeypots. For instance, Mukkamala et al. [6] proposed network-level honeypot detection technique, that uses timing analysis of ICMP ECHO requests. Another research presented in [7] describes honeypot detection model based on analysis of the essential honeypot detection characteristics. While a deceptive system to escape from honeypot hunting is proposed in [8]. This system encompasses several detection methods for both low and high-level interaction honeypots, deployed in the virtual and physical environment. In [9], the authors developed a method to prevent honeypot detection in order to allow honeypots to bypass botnet defence. This defence is created to detect honeypots and also prevent them from joining botnets. The proposed method increases the chance of adding honeypots to a botnet and thus help to analyze and detect the botnet attacks followed by their destruction. In the same spirit, authors in [10] proposed a mechanism to use honeynet securely in a botnet, which uses sensor technology for honeypot detection. Sensor-based botnet defence system is designed to discover a node that does not pass malicious traffic as honeypots do. Authors solved this issue by allowing malicious traffic to propagate without modifying the other honeypot nodes in the honeynet.

Fingerprinting is a technique used by attackers to detect whether a system is a honeypot or not. Many attempts are made by the research community to make the honeypot more complex in order to reduce the chance of being discovered by the attackers, through fingerprinting. One such attempt is performed by authors in [11], where at first threat modelling is performed to analyze the potential security threats of the fingerprinting results. Afterwards, they enhanced its deception capability by modifying system configuration and custom scripts. Use of sandbox mechanism is another approach to make honeypot detection harder for the attackers. It provides an isolated environment for analysis of botnet. Researchers in [12] used this approach to develop a honeynet to detect and analyze a centralized botnet.

Despite all of the above research work, there is a lack of research which emphasizes specific honeypot implementations and associated weaknesses. Our aim is to fill this gap as it can make honeypot less discoverable for the attackers and pave a path for a stronger and secure network.

III. System Model

In this section, we present our detection model and various honeypot detection techniques. Our system is divided into four stages as shown in Figure 1.

A. Research Methodology

- The first stage is a preparation. We investigated and researched existing works related to honeypot detection. We choose two popular medium-level interaction SSH and Telnet honeypots [13] and [14] for our analysis.
- In the second stage, the most suitable methods and models of honeypot detection are determined by exploring their source code. In addition, we deployed and tested selected honeypots approaches. Afterwards, we compare our introduced enhancements with existing honeypots techniques.
- The third stage is focused on automation of honeypot's detection techniques. We developed and implemented an open-source software for honeypot detection, as described in section IV.
- In the fourth stage, experiments are conducted followed by systematization and analysis of the obtained data.

B. Honeypot detection Model

We use honeypot detection model proposed in [7]. A quadruple $< S, M, R, f >$ represents the honeypot detection model, where S is the set of systems waiting for detection, M is the set of the detection methods, R is the set of detection results, and f is the judging function of the honeypot.

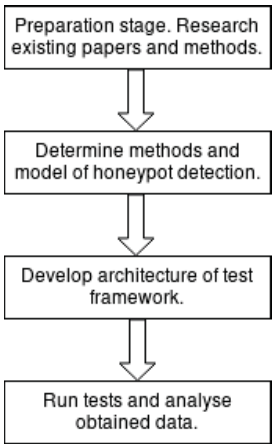

Figure 1. Phases of research methodology

1	Check the result of ping utility. The ping utility from honeypot always returns the same result even if a domain name is incorrect
2	Check the default version of OpenSSH
3	Check the default host name
4	Check authorization with one of the default passwords

Table I: Methods for default honeypot configuration

For each detection system $s \in S$, for any detection methods $m \in M$, detection results into $m(s) \in \{0, 1\}$, if $m(s) = 0$, it implies the system S does not have the characteristic m of the honeypot; if $m(s) = 1$, it means that the system S has the characteristic m of the honeypot. The detection result set $R = \{m(s) = [0|1]|s \in S, m \in M\}$ is the result of using the detection methods set M to detect system S. The weighed value of each method is used to describe the importance of the honeypot detection methods. If the weighed value is λ_m for $m \in M$, then we have $\sum_{m \in M} \lambda_m = 1$ and judging function is given as follows:

$$f(s) = \sum_{m \in M} \lambda_m \times m(s) \in [0, 1]$$

C. Honeypot detection methods

We selected honeypots presented in [13] and [14] for our research since they are the most popular medium level honeypots. The following features are provided by these honeypots by default:

- the fake file system with the ability to add/remove files,
- the ability to add fake file content,
- supporting for SSH exec commands,
- storing the session logs,
- and many more additional commands.

Honeypot detection methods have few common features and are typically divided into three categories. To this end, we also developed three groups for honeypot identification methods:

- default configuration of honeypots,
- base Linux functionality, and
- atypical behavior of system.

In the following, we explain each group of honeypots.

1) Default Configuration: In this group, default parameters of honeypots are analyzed. Some parameters cannot be changed from configuration files, and one way to change them is to rewrite the source code. *Detection methods 1-4* are presented in Table I.

2) Linux Based: This group contains methods based on Linux functionality as listed in Table II. These methods are discussed as follows:

Detection method 5: The ¨$?¨ command should return an exit status of the last command. If we pass a random sequence of character into bash interpreter, the status of the last command is not zero. The implementation of honeypots ignores this global variable and zero is always returned.

5	Check returning value of command that leads to fatal error
6	Check loop construction
7	Check inode number of folders/files
8	Check standard error output redirection
9	Check mkdir command with –parent flag
10	Check wget command with –help flag
11	Check the result of 'cd ∼/' command

Table II: Methods based on Linux functionality

Detection method 6: Loop constructions of the bash interpreter are the base components. We check result of *"for i in 1..5; do echo kxe; done"* command. Bash should return five lines with incrementing number from 1 to 5. Implementation of honeypots ignore this command and return an error of syntax.

Detection method 7: The "ls -il /" command shows the inode number of files from the root directory. In general for *nix file systems, with each file or directory, there is an associated inode. The inode is typically represented as an integer number. If this command is passed to the interpreter, it provides an empty output.

Detection method 8: Redirection of error output is the basic feature of bash interpreter. We make a fake command with redirect output file and check the contents of the redirected file. If the file exists and the file contains some error, the test is failed. Otherwise, the test is passed.

Detection method 9: The mkdir command creates a directory. The mkdir command provides an option to create all the parent folders on the fly if they do not already exist. The honeypot implementation does not support this flag. However, this option is the base functionality of the mkdir command.

Detection method 10: Some implementations of honeypot do not support –help option of the wget command.

Detection method 11: The "cd /" command redirects to a home directory of the login user. However, honeypot implementations return a syntax error.

3) Atypical Behavior: This method checks the atypical behaviour of the server, as listed in Table III. Honeypot has a number of features which are unlikely to be encountered in a real system. We used these features for detection.

Detection method 12: A real control system does not have a large number of open ports. If the number of ports is more than 5, the test is not passed.

Detection method 13: In a real system, all files are saved even user closes the session. Honeypot does not save a file after users close ssh session.

Detection method 14: Cowrie honeypot checks a result on VirusTotal after curl or wget command. We can use this mechanism to detect cowrie. Our approach is to implement a simple HTTP server which can process random URLs. We generate a random unique URL and make a request using curl or wget command to the honeypot. Finally, we check the generated URL on VirusTotal. If this URL already exists, the test is passed.

12	Check the number of open ports on the server
13	Check newly created file after re-login
14	Check the result on VirusTotal after curl or wget execution

Table III: Methods based on atypical behavior

In addition, we have a metric for checking the previous history of IP honeypots. These test results are stored in the database in form of a pair of IP address and associated calculated score.

IV. Experimental Results and analysis

1) Testing environment: The testing environment used for our experimentation is shown in Fig. 2. We use docker container to run honeypots to prevent possible attacks on our real servers. Cowrie has an officially available image in the docker hub repository. Kippo image is also obtained from docker hub. We use two machines with different IP addresses for our tests. We do not deploy all honeypot on one machine as we have assumed that one machine cannot have more than 5 open ports. There are 6 machines in our test environment and two groups of honeypots; one with default configurations and second with modified ones. Analysis of the first group shows that honeypots with default configuration can easily be detected. Analysis of the second group shows that even well-configured honeypots are still detectable. Also, we deploy a clear SSH server to detect false positive rates of our methods.

2) Testing Software Development: An application is developed using Python 3 language. It takes an IP address and port of the target system as command line parameter. Test framework is run using the command: python3 honeytector/app.py, and afterwards an HTML report is generated by the applications. This report lists total score along with results of test methods, as shown in Fig. 3.

The architecture of the developed application in form of UML class diagram is shown in Fig. 4. Our main goal for this application development is to add new detection methods. All detection methods should inherit from IMetrics interface. Metrics are grouped into Collector class. MetricsCollector class contains instances of all collectors. First, the main loop of the application gets a metrics list with a result testing from MetricsCollector class. This list is passed to MathModel class, where coefficients are calculated for every metrics. Finally, DocumentGenerator makes an HTML report with testing results. Our proposed architecture is easier to understand and research community can conveniently add new methods of honeypot detection to our designed model.

Honeypot	Configuration	Probability
Cowrie (HP1)	Default	1
Cowrie (HP3)	Modified	0.79
Cowrie (HP5)	Modified	0.64
Kippo (HP2)	Default	0.89
Kippo (HP4)	Modified	0.79
Kippo (HP6)	Modified	0.64
real SSH (SSH1)	Default	0.15
real SSH (SSH2)	Default	0

Table IV: Probabilistic estimation of honeypot detection

	HP1	HP2	HP3	HP4	HP5	HP6	SSH1	SSH2
1	+	+	+	+	+	+	-	-
2	+	+	-	-	-	-	+	-
3	+	+	-	-	-	-	-	-
4	+	+	+	+	-	-	-	-
5	+	+	+	+	+	+	-	-
6	+	+	+	+	+	+	-	-
7	+	+	+	+	+	+	-	-
8	+	+	+	+	+	+	-	-
9	+	+	+	+	+	+	-	-
10	+	+	+	+	+	+	-	-
11	+	+	+	+	+	+	-	-
12	+	+	+	+	-	-	+	-
13	+	+	+	+	+	+	-	-
14	+	-	-	-	-	-	-	-

Table V: Detailed results

3) Experimentation Results: Results of our experimentation are represented in Table IV and Table V. These results confirm that the proposed methodology can successfully detect honeypots. Even with modified configuration, honeypot will be identified by our software. '+' sign is placed to signify that a honeypot is detected by our software, while '-' sign signifies that the system looks real. Also, we can see, changes in configuration decrease the detection probability, but it is not enough to hide the typical honeypot behaviour.

Additionally, it can be clearly seen that the defined set of methods allows distinguishing honeypots from real systems with a high degree of probability.

4) Recommendations: We have the following recommendations based on experimental results:

- We should not use hard-coded values for utilities; if hard-coded values are returned after a command is called, this command is identified as fake. For example, the "ps aux" command should return a list of the running processes. If the list of the process is hard-code value and this list is returned every time whenever a user uses "ps aux" command, it leads to suspicious behaviour.

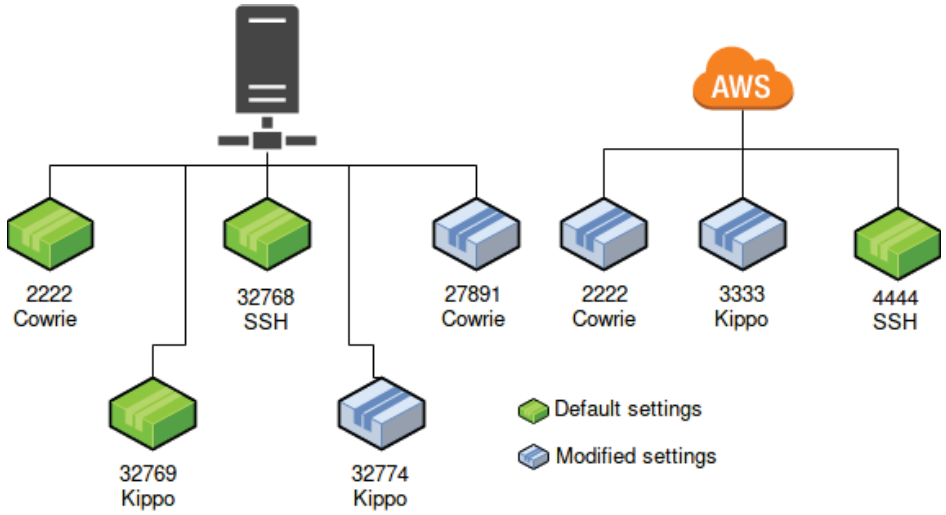

Figure 2. Testing environment

Metric name	Result
Ping	-
OpenSSH version	-
Host name	+
Authorization	-
Command return value	+
Loop	+
Inode number	+
Error output redirection	+
mkdir	-
wget	+
cd	-
Open ports	+
New File	-
VirusTotal	-
Honeypot probability is: 82%	

Figure 3. Example of the generated report

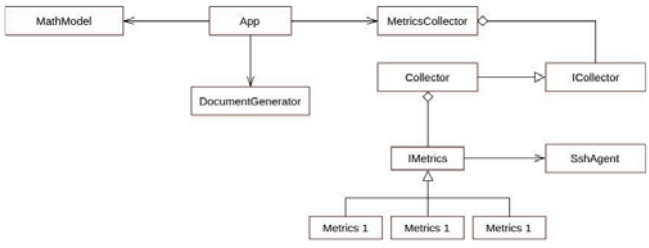

Figure 4. The architecture of the developed application

- We should always check that the host requested through ping command exists.
- We must not delete files created by attackers.
- We should send files requested by attackers to a sandbox and after some delay or provoked user command, send these files to VirusTotal. If the delay is not specified, our detection method identifies the given service as a honeypot.
- It is not recommended to run many services on the honeypot machine since a real control system does not have a large number of open ports. If a machine has a lot of services, then it is most probably a honeypot.
- Emulated Linux utilities should support full functionality from the origin.

V. Conclusions and Future Work

Honeypot detection technology is often used in network attacks for analyzing, whether the attacked system is a honeypot or not. This is done by gathering the responding information through certain operation, to avoid the goal to know the aim, methods and tools of the attack. By analyzing the medium-level interaction SSH honeypots, our work presents a probabilistic estimation of honeypot detection.

In contradiction to previous researches, our work is focused on software architecture and implementation problems of the specific honeypots. We showed that the majority of currently used honeypots can be easily detected and avoided by attackers. Also, we can conclude that the only way to fix honeypot weaknesses is to completely change their implementations.

In addition, we implemented an open-source software[2] based on our methodology. It is a fully-functioning prototype, which can allow honeypots detection with a certain degree of probability. The open-source approach provides the opportunity for developers to extend our software with additional methods.

[2]https://gitlab.com/legik/honeypot

The main goal for the future work is to extend and improve the designed software with machine learning techniques, that will help to define more accurate coefficients. Furthermore, the proposed detection model can be extended to support other types of honeypots.

REFERENCES

[1] H. Semic and S. Mrdovic, "Iot honeypot: a multi-component solution for handling manual and mirai-based attacks," *IEEE Telecommunications forum TELFOR*, 2017.

[2] W. Fan, Z. Du, D. Fernandez, and V. Villagra, "Enabling an anatomic view to investigate honeypot systems: A survey," *IEEE System Journal*, 2017.

[3] R. M. Campbell, K. Padayachee, and T. Masombuka, "A survey of honeypot research: Trends and opportunities," *IEEE International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015.

[4] S. Dowling, M. Schukat, and H. Melvin, "A zigbee honeypot to assess iot cyberattack behaviour," *IEEE Irish Signal and System Conference*, 2017.

[5] Anirudh, A. Thileeban, and D. J. Nallathambi, "Use of honeypots for mitigating dos attacks targeted on iot networks," *IEEE International Conference on Computer, Communication, and Signal Processing (ICCCSP)*, 2017.

[6] R. B. S. Mukkamala, K. Yendrapalli, "Detection of virtual environments and low interaction honeypots," *Information Assurance and Security Workshop, 2007.IAW'07.IEEE SMC*, 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4267547/

[7] D. Wenda and D. Ning, *A Honeypot Detection Method Based on Characteristic Analysis and Environment Detection*. New York, NY: Springer New York, 2012, pp. 201–206. [Online]. Available: https://doi.org/10.1007/978-1-4419-8849-2_26

[8] S.-J. K. Lai-Ming Shiue, "Countermeasure for detection of honeypot deployment," *Proceedings of the International Conference on Computer and Communication Engineering*, 2008.

[9] M. Al-Hakbani and M. Dahshan, "Avoiding honeypot detection in peer-to-peer botnets," *IEEE Conference on Engineering and Technology (ICETECH)*, 2015.

[10] C. Costarella, S. Chung, B. Popovsky, and D. Dittrich, "Hardening honeynets against honeypot-aware botnet attacks." [Online]. Available: www.tacoma.washington.edu/sites/default/files/sections/.../C_Costarella.pdf

[11] R. Dahbul, C. Lim, and J. Purnama, "Enhancing honeypot deception capability through network service fingerprinting," *Journal of Physics Conference Series*, 2017.

[12] M. S. A. Gregio, L. Duarte, and A. Montes, "Botnet detection and analysis using honevnet," *ICoFC S*, 2007.

[13] desaster et al., "Kippo - ssh honeypot." [Online]. Available: https://github.com/desaster/kippo

[14] M. Oosterhof, "Cowrie ssh/telnet honeypot." [Online]. Available: https://github.com/micheloosterhof/cowrie