Honeypot Configuration and Data Analysis

Jared Campbell
The University of Texas at Austin

David Zehden
The University of Texas at Austin

ABSTRACT

This project aims to build a honeypot server and analyze the data it collects. The goal of a honeypot server is to create vulnerable server on the open internet which we expect will be attacked by malicious actors. These attacks will be logged by the server, then we will analyze the data collected to find common trends in the attacks. Our honeypot will collect data on both an attacking machine and the attacks it directs at the honeypot. Our honeypot server is hosted on an Amazon Web Services (AWS) virtual server instance. The network for the server has been configured to allow all incoming traffic on monitored ports we expect to be targeted (such as SSH). We then analyzed the collected data using Python to find trends in attacker IP addresses, attacker operating systems, attacked ports, and timing of attacks. The honey pot was successful in collecting a large amount of attack data, averaging over 1000 unique data points per day. Our research has determined that a honeypot server is an effective tool for monitoring potential attacks on a network. We have also found honeypots to be highly configurable which allows for the collection of data specific to an organization's needs.

1. INTRODUCTION

A honeypot is a server that is made intentionally vulnerable in order to attract the attention of malicious actors. The server then logs any attempts by attackers to exploit and gain access to it. The goal is for researchers to be able to analyze common trends in the kinds of attacks used by malicious actors and even possibly discover new kinds of attacks that have never been seen before. Another use case for honeypots in an enterprise environment is to slow down attackers by allowing them to attack the non-critical honeypot instead critical infrastructure.

There are many different tools that can be used to create a honeypot. For example, the honeypot could emulate a vulnerable web app or a vulnerable end-user machine. The configuration of the honeypot depends on the goals of its creators, whether that be research or protection as described above. Our goal is to research these various tools and configurations, and to gather enough data to be able provide an analysis of current trends in attacks. Since there are so many different tools and honeypot configurations, our implementation will focus on a general honeypot meant to collect as much data as possible. The results of our research will demonstrate the effectiveness of a honeypot in collecting different types of data which could be used to improve the security of an organization and provide knowledge on general trends to the information security community.

For our approach, we have configured a honeypot as a vulnerable server using various tools which we have found to emulate common vulnerabilities and log attack data. We have also configured a front end website for the honeypot. The collected data is automatically analyzed and visualized via Python scripts, and the results are displayed on the website. The most interesting part of our design is that once the honeypot is configured, it runs continually without any need for further user interaction. The longer the honeypot runs, the more data it collects, and the more accurate trends over time become. As we were only able to run our fully configured honeypot for approximately one week, the analysis only shows common trends over a one week time period. However, given more time, the honeypot could eventually demonstrate trends over months or even years. The longer the honeypot is active, the more interesting the data becomes.

In our initial results, we were able to track the top attacker IP addresses and their country of origin, attacker operating systems, targeted ports, timing of attacks, and SSH attack data such as the top usernames and passwords of all login attempts. Interestingly, the top five countries from which attacks originated were the United States, Netherlands, Russia, China, and Romania (in order of number of detections). The two most widely used operating systems by attackers were Linux 2.2.x-3.x and Windows 7 or 8 (respectively). It is interesting that attackers are using fairly outdated operating systems (current Linux kernel is 4.14 and Windows 7 will reach end of life in January 2020).

In the future, we would like to continue to run the honeypot server in order to collect data over a larger time period. It will be interesting to see how trends change across weeks and months as well as overall long-term trends. There is also much more we could configure on the honeypot, such as making masking certain elements of the honeypot detection programs to make the honeypot seem like a more realistic target. We theorize that the more enticing the honeypot

looks to attackers, the more likely they would be to attack it and possibly even use more advanced attacks, all of which would provide us with even more detailed data.

2. MOTIVATION

Mairh, et al. discuss a variety of use-cases for honeypots, among them is the use of a honeypot in conjunction with an Intruder Detection System (IDS). The authors note that an IDS typically uses misuse and anomaly detection [1]. We hope to both implement a system similar to the one described by Mairh, et al. and perform an analysis of the data we collect from the system in accordance with the type of anomaly detection described.

Additionally, Chuvakin was able to classify individual types of attacks on his system [2]. Such a level of analysis is impressive, and we hope to achieve similar data collection, albeit in a lesser volume due to the significantly shorter time frame. Also, our analysis may be more focused on traffic metrics rather than the sort of in depth log analysis done by Chuvakin. We are more inclined towards determining if and when an attack has occurred rather than what specific attacks occurred.

3. OUR ARCHITECTURE

The honeypot is hosted on an Amazon Web Services (AWS) virtual server instance. We have had to upgrade our instance to increase it's storage and memory. The server now runs an EC2 t3a.medium instance with two processor cores, 4GB or RAM, and 30GB of SSD storage. The server runs an Ubuntu Server 16 image as its operating system.

To create the functionality of the honeypot, we have configured several tools which act as sensors or detectors. Dionaea captures malware and can simulate certain individual vulnerabilities such as SQL, FTP, SMB, and HTTP vulnerabilities. Cowrie is an SSH honeypot which logs brute force attacks and shell interaction. Snort is a network intrusion detection system which does not simulate any vulnerabilities itself, but detects and logs attacks targeted at the vulnerabilities provided by other tools. These sensors are all managed by the Modern Honeypot Network (MHN). MHN provides a single platform for managing honeypots and offers several useful analytical tools for assessing high level attack metrics as well as management tools to easily modify, add, and remove honeypot instances.

The front-end website which we use to monitor our honeypot and its sensors, as well as visualize the data they collect, was initially provided by MHN. We have modified the MHN site to fit our needs by configuring it specifically for our sensors, and adding additional visualizations and analyses.

These visualization and analyses were created using Python to graph trends. There are several Python scripts which run on the server to continually analyze the data collected by the honeypot and construct graphs and charts which are then displayed on the main front-end site.

4. SETUP AND CONFIGURATION

The three sensors used by the honeypot, Dionaea, Cowrie, and Snort, were each initially individually configured through MHN. We then manually configured the logging levels of each sensors to ensure relevant data was logged separately from error and debugging information so that it could be parsed by our Python analysis scripts.

We have also configured the AWS server's firewall and security groups. We are allowing all inbound network traffic on all ports monitored by the honeypot (SSH, FTP, HTTP(S), etc.) and denying all outbound traffic. SSH access is restricted to our own SSH keys, and the SSH port has been changed from the default of 22 to allow Cowrie to monitor attacks on port 22. Password logins are denied and root login is also denied.

For data analysis, we used the `mongoexport` command line tool to export data collected by MHN to JSON files. We used `SCP` to securely copy these files onto our local machines. Then we used Jupyter notebooks to explore data. Primarily, the `matplotlib`, `numpy`, and `pandas` libraries were used to aid the development of our findings. Additionally, we modified MHN to include a new webpage for particularly interesting findings. Such findings were generated from a daily `cron` job we set up on the server to run scripts that we developed locally.

5. RESULTS

Due to some technical issues, we had to reset our EC2 instance, thus the findings presented are those collected following the reset. As of the writing of this paper, we have made over 20,000 detections on our honeypots. Modern Honeypot Network provides some useful high level statistics about our honeypots some of which may be seen in Table 1 and Table 2. Another interesting observation is that the vast majority of our data came from Snort, as over 13,500 detections originated from our Snort honeypot.

We also found that 1433 was attacked significantly than any other port for a given 24 hour period. This is likely due to port 1433's use as the default port for many SQL servers.

IP Address (first 5 digits)	Country	Number of attacks (last 24 hours)
83.97.2	Romania	19
89.248	Netherlands	18
185.15	Russia	18
77.247	Netherlands	17
185.15	Russia	16

Table 1: Top Five Individual Detected IP Addresses

Attacked Port	Number of Attacks	Common Port Use
1433	182	SQL Server
5060	87	Clear Text SIP, VoIP
22	53	SSH
8545	17	Remote Procedure Call interface of Ethereum clients
443	16	https

Table 2: Top Five Attacked Ports

Using the PyGeoIpMap library, we were also able to plot the approximate locations of attacker IP addresses (Figure 1). Additionally, we plotted the top 20 countries by attacker, as seen in Figure 2. While IP addresses from all around the world were detected, the majority came from the United States, the Netherlands, China, Russia, and Germany. One note of importance regarding IP addresses is that the addresses we've encountered are not necessarily those of the attacker since attackers could be using VPNs.

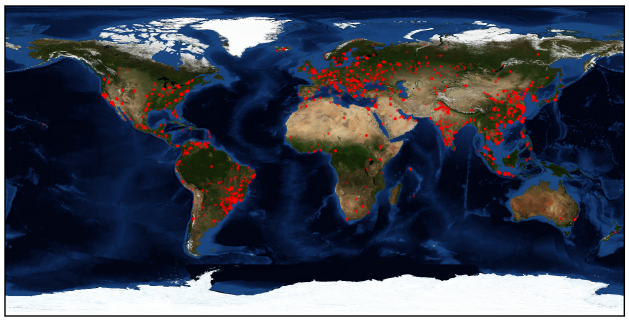

Figure 1: Approximate Locations of Attacker IP Addresses.

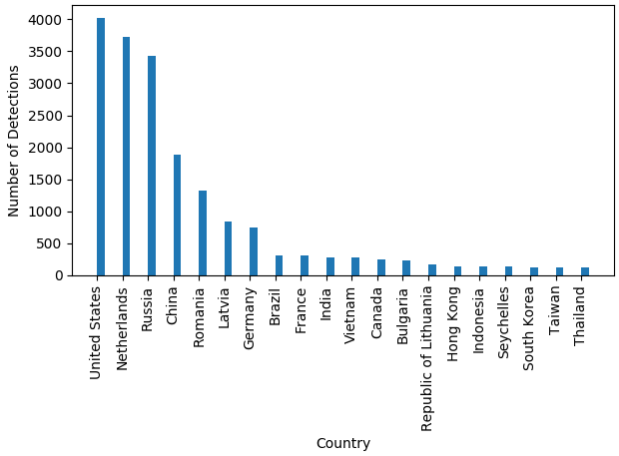

Figure 2: Top Detected Countries.

We also explored the types of operating systems detected. Most operating systems were either Linux 2.2-3.x of some barebones variety or Windows 7/8. Unfortunately for most detections we were not actually able to determine the operating system. The breakdown of detected operating systems can be seen in Figure 3.

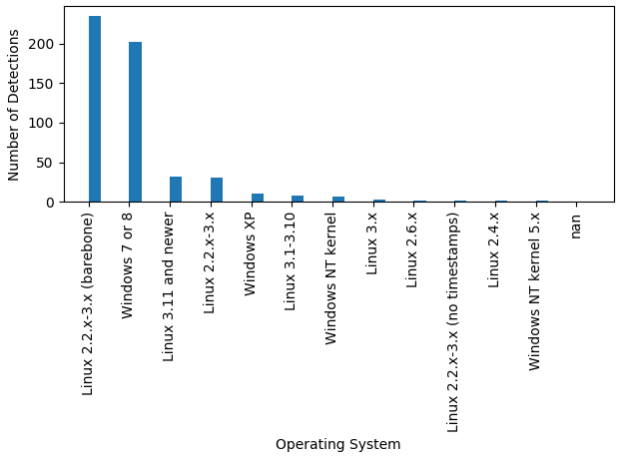

Figure 3: Top Detected Operating Systems.

The final datapoint we explored was the time of detection. The full breakdown of attacks can be seen in Figure 4. From our findings we can see that, for the most part, most detections cam during the first half of the day (relative to UTC). However, there nearly always seemed to be some detections occurring.

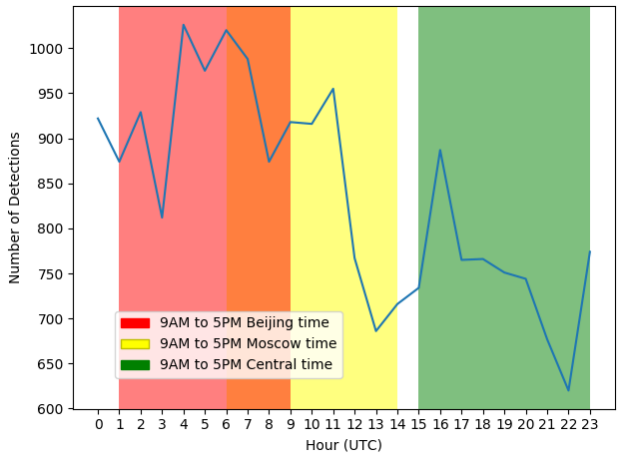

Figure 4: Detections by Time

While we were able to detect a great deal of information, it is difficult to determine the exact intent of all those we detected. In some cases, we can see that SSHing and crypto mining were attempted. In other cases, we can only see that an https request was made. That being the case, the line is blurred when attempting to divide detections into malicious and non-malicious activity.

6. CONSIDERATIONS

Since we are using AWS to host our honeypot, we must consider the configuration of the AWS instance itself. In particular, we will consider AWS firewall rules and security groups to allow all incoming connections and deny all outgoing connections. We must also consider that AWS might shutdown our server if they detect it is under attack. In that case, we will have a backup of the server prepared and it will be migrated to a private server which we control. Finally, if our honeypot does not collect enough attack data in the time period during which it is active, we will attack the server ourselves to demonstrate its capabilities.

If time permits, we have further plans to add more AWS honeypot servers and network them together with our original instance to hopefully see how an attack might traverse the network. We can also implement an intrusion detection system such as Snortif we have enough time.

7. RELATED WORK

Neil Fox's post about configuring Dionaea and Cowrie was very useful in learning how to setup the tools in our honeypot [1]. In his post he describes how to install, configure, and add improvements to Dionaea and Cowrie. He recommends implementing log rotation to prevent the bistreams logs that Dionaea uses from filling up. He also implements additional improvements to Cowrie that allow an administrator to replay the BASH sessions of an attacker once they

are inside the honeypot. We intend to implement both log rotation and replay in our own design.

Rapid7 wrote a guide for configuring honeypots on AWS [5]. Although the guide is specific to Rapid7's InsightIDR honeypots, the configuration of their AWS environment was particularly valuable. They describe in detail how to launch an EC2 instance, create security groups, and create IAM roles. We referred to this article for our own configuration of our AWS environment.

Steve Gathof also wrote an excellent article on the setup of a honeypot on an AWS EC2 instance [2]. While his work will likely be a useful reference for our architecture, it is not explicitly related to the analysis that we will be performing.

Additionally, Polyakov et al. explored both the architecture and several specific functions that a honeypot should fulfill [4]. We will try to implement the functions mentioned in the Polyakov's article, in particular, creating variable conditions for accessing the system in order to possibly filter out low level intruders.

Kuwatly et al. discuss the design of a dynamic honeypot, which is "an autonomous honeypot capable of adapting in a dynamic and constantly changing network environment" [3]. In their paper, they describe how to combine active probing, passive fingerprinting, and a network intrusion detection system with a honeypot. This allows for the dynamic honeypot to automatically update and adjust to changes in the network environment in which it is located. While our network environment is currently comprised of only a single server, we would like to eventually extend our work to a larger network of multiple honeypots. If we are able to extend our work as such, this paper will be an invaluable reference.

8. CONCLUSIONS

We hope to construct and analyze a honeypot server. We intend to perform an analysis of our findings in order to determine trends regarding the traffic our server receives. In particular, we hope to detect and examine anomalies on our system. We do expect to receive a some amount of traffic from automated sweepers and hope to attract more interesting attackers as well.

9. REFERENCES

[1] N. Fox. Setting up dionaea & cowrie with mhn. September 2019.

[2] S. Gathof. Deploying a honeypot on aws. 2018.

[3] I. Kuwatly, M. Sraj, Z. Masri, and H. Artail. A dynamic honeypot design for intrusion detection. pages 95– 104, 08 2004.

[4] V. Polyakov and S. A. Lapin. Architecture of the honeypot system for studying targeted attacks. *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*, pages 202–205, 2018.

[5] Rapid7. AWS honeypots.