A HoneyNet Environment for Analyzing Malicious Actors

Daniel N. Gisolfi; Michael Gutierrez; Tyler V. Rimaldi;
Casimer DeCusatis, *Fellow, IEEE*; and Alan G. Labouseur
Marist College
School of Computer Science and Mathematics
Poughkeepsie, NY 12601
{Daniel.Gisolfi1, Michael.Gutierrez2, Tyler.Rimaldi1, Casimer.DeCusatis, Alan.Labouseur}@Marist.edu

***Abstract*—A honeypot is a web application or other resource that is deceptively constructed to log the actions of its users, most (but not all) of whom can be assumed to be malicious actors. A honeynet is a network of honeypots. Thanks to their interconnectedness, honeynets allow for vast amounts of data to be collected for analysis. In this paper we discuss how we came to build a honeynet, its design and implementation, and a few insights gained by analyzing attack data gathered from it.**

I. Introduction

The frequency of cyber attacks have been increasing in recent years [1]. As more devices become compromised and infected, more data is lost or exposed to malicious actors. The brilliant interconnectedness of Internet of Things (IoT) devices promises to be a cheap but effective addition to malicious users' arsenals. In 2016, a botnet resulting from the Mirai Malware on IoT devices perpetrated a distributed denial of service (DDoS) attack and took over household cameras [2]. Symantec states that there has been a 600% increase in IoT attacks during 2017 [1]. In May 2018, the University of Vermont became a target. The school quickly noticed the intrusion and alerted its students, faculty, and staff to change their NetID passwords [3]. Fortunately, UVM did not have its data exposed or stolen. In summary, it's pretty bad out there, and getting worse.

Yet there is hope. By developing and deploying a honeynet, we hope to collect vast amounts of attack data. Utilizing this data, we aim to discover attackers' strategies, motives, and investments – thus providing insight on how to prevent or mitigate similar attacks as they occur in the near and distant future.

Key contributions of this paper include the following:

traces the evolution of a honeynet
describes the architecture of a honeynet
discusses preliminary data analysis from a honeynet
demonstrates honeynets as a valuable tool for providing insight on cyber attack data

The remainder of this paper is organized as follows: Section II discusses our prior work and introduces the concept of a honeynet. Section III describes our honeynet implementation. Section IV provides a preliminary analysis of attack data. Section V concludes with a plans for future work.

II. Background

We have come to develop our honeynet through the natural (for us) evolution of our cyber security research that began with using graph analytics to examine data we were collecting from individual SSH and SDN honeypots.

A. Evolution from Prior Work

G-star Studio [4] is a web-based front end to G*, the Dynamic Graph Database [5]. Both make up part of the analytic core of our cyber security research. Soon after making G-Star Studio available on the public Internet, we observed a number of unauthorized connection attempts to its Application Programming Interface (API). These attacks specifically targeted G-star's REpresentational State Transfer (REST) API. We noticed that our VM ran out of disk space because the G-star API log file grew to tens of gigabytes. Looking at the huge log file, we realized we had inadvertently invented an API honeypot and Pasithea [6] was born.

Once we were working with three individual honeypots – an SSH honeypot called LongTail [7], an SDN honeypot mimicking a software defined network controller and administrative system called Dolos, and our new "accidental" REST API honeypot now called Pasithea – we found ourselves considering two steps forward: developing a high interaction honeypot and connecting our existing honeypots in a network. . . a honeynet.

B. Low Interaction and High Interaction Honeypots

Generally stated, a honeypot is a web application or other resource (a "system") that is deceptively constructed to log the actions of its users, most (but not all) of whom can be assumed to be malicious actors. Such a tool is classified under one of two categories: low interaction or high interaction.

Low interaction honeypots emulate certain vulnerabilities within a system [8], [9]. Essentially, this kind of honeypot includes a subset of existing vulnerabilities a system may possess. Because these vulnerabilities are emulated, it does not put the actual system at risk, as it restricts the mobility of an attacker. While low interaction honeypots collect detailed attack data, the range of data they can collect is limited because these honeypots do not give attackers mobility throughout the system. As such, this type of honeypot does not collect diverse data. Instead, it only collects data with respect to the specific points in the system or vulnerabilities it emulates. For example,

a web application may have a login screen or an API help page listing its commands. These resources may not have any functionality at all. In fact, they may simply return (perhaps random) errors. Essentially, this wastes attackers' time and skills while logging their actions, thus enabling us to learn from their (attempted) exploits.

High interaction honeypots, on the other hand, let an attacker exploit many emulated vulnerabilities within a system. These systems generally contain many links and layers, thus resembling a large infrastructure [8], [9]. By encouraging the attacker to take control of the entire system or large parts of it, these honeypots allow the attacker to gain mobility throughout the system, all while their activities are being logged. For example, once a malicious actor "hacks" the credentials of a login screen or uses data from an API help page to execute commands, the responses from a high interaction honeypot lead the malicious actor to more resources and other parts of the system. While this still wastes attackers' time and skills, we gain additional data by logging more of their attack exploits and also the data they supply in using the system (e.g., search terms), the paths they take through the system, and the techniques they employ to move from resource to resource.

There are many ways to construct a high interaction honeypot. One way is to take several low interaction honeypots and link them together to form a honeynet.

C. Our HoneyNet

Our honeynet is currently in development, the details of which follow in Section III. However, we have built and deployed an alpha version that includes four interconnected honeypots: the Longtail SSH honeypot, the Dolos SDN/admin honeypot, the Pasithea REST API honeypot, and our newly constructed high interaction REST API honeypot called Peitho. To analyze all of the data we're collecting, we use a message queue to send log files to a database and also to LCARS [10], our Lightweight Cloud Application for Real-time Security, an analysis and visualization tool. This tool enables us to perform graph analyses and visualizations, hive plot visualizations, and relational analyses, all of which help us explore correlations, frequencies, and outliers in the cyber attack data.

III. HONEYNET CONSTRUCTION

As mentioned earlier, before we constructed our initial honeynet we had deployed each of our individual honeypots as separate, low-interaction entities. Our fleet consisted of the Longtail SSH honeypot, the Dolos SDN/admin honeypot, and the Pasithea REST API honeypot. Longtail, our SSH honeypot, was constructed with C and Perl. Dolos, our SDN/admin honeypot, was constructed using Flask, a lightweight Python web framework [11]. Pasithea, our REST API honeypot, was constructed with NanoHTTPD, a lightweight Java web-server [12].

A. A New Honeypot as an Entry Point

Also noted earlier, low interaction honeypots do not provide much functionality for the attacker and are therefore limited in the data they can collect. Nonetheless, they have been successful at gathering substantial attack data. Despite this success, we wanted to develop something more interactive that would enable us to collect even more data. Therefore, we have begun transforming our individual honeypots into an interconnected honeynet. To facilitate this and to provide a high interaction entry point to our honeynet, we created a high interaction REST API honeypot called Peitho. It builds on the techniques of its low interaction predecessor (Pasithea) and provides high interaction features such as a login screen, a file directory and retrieval system, two reroute methods, and an interactive help page. It is currently serving as the entry point to our honeynet.

B. Deploying the HoneyNet

The alpha version of our honeynet is hosted in the Marist College Enterprise Computing Research Lab (ECRL) utilizing an IBM server cluster. Currently, each honeypot in our honeynet is scattered across multiple TCP ports on a single network with a public IP address.

Our honeynet lures attackers in the following manner: as attackers find interesting and useful data on one honeypot, depending on their skills, they will be able to piece that information together, or take the bait, to gain access to other honeypots. This can be thought of as a kind of cognitive or motor skill development technique often performed with developing babies, like placing the right shapes into the right holes. Just as doctors watch and record their infant patients place shapes into holes, our honeynet watches (and logs) each and every step taken by the attackers. This enables us to collect detailed data so that we can analyze their strategies and motives, and develop better-tailored bait for the future.

In order to scatter our honeypots, we use Docker, a containerization platform [13]. We host each honeypot in its own Docker container. Containerization allows each honeypot to run in a standalone, dedicated Unix environment (Ubuntu in this case). This creates a modular system that allows us to add and remove honeypots on the fly and also allows for easy honeypot management and deployment. Furthermore, containers cannot reach each other unless they are translated to the proper sub-network of the virtual machine (VM). Containerization creates another level of security, as these containers exist independently from one another on the Docker sub-network of the VM.

To allow attackers to reach these containers, we have mapped the ports of the honeypot VMs to Docker sub-network ports located on the host machine. Such translation creates a safety net against various types of cyber attacks on the VM. For example, a DDoS attack on one honeypot will not affect any of the other honeypots because they each reside on their own individual containers. Fig. 1 provides an overview of our honeynet architecture.

C. Moving Through the HoneyNet

Cyber attackers can reach our honeynet on port 80. This gives attackers access to one of our entry points, Pietho, our newly-created high interaction REST API honeypot. It contains special features such as an administrative login page, a file directory and retrieval system, and two reroute methods. Each of these functions rely on either an HTTP GET or POST request. If attackers break through the administration login layer, they will reach the file directory. The file directory layer

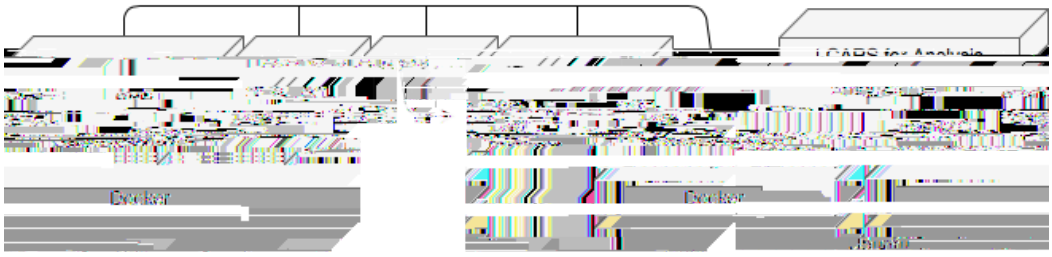

Fig. 1. Honeynet Architecture

contains data about what files are accessible and provides the file type and file path. Using the file path, an attacker can attempt to figure out the existence of our file retrieval system. If an attacker discovers this system they will be able to retrieve files by using the file path. Currently, there exists a file that contains the SSH login credentials for our SSH honeypot, Longtail. Using that data, attackers can link to Longtail. This serves as a terminating point and will deny access to the attacker while logging each move the attacker makes regardless of what credentials or tactics are used.

If attackers do not find or choose not to visit Longtail, they can visit the REST API help screen. From the REST API help screen they can use one of two reroute methods. Depending on which reroute method is requested, the attacker will either be linked to Pasithea, our low interaction REST API honeypot, or to Dolos, our SDN/admin honeypot.

If the attacker chose to access the REST API honeypot, Pasithea, they would be presented with a 404 error screen. This honeypot is able to take any type of request, regardless of the HTTP method. However, our honeypot has been strategically designed to model G* Studios's API. This allows our honeypot to be unidentifiable and indistinguishable from a normal HTTP server [6].

If an attacker takes the system admin direction, Dolos, our SDN honeypot, they will be prompted to enter login credentials. The credentials are intentionally made to be simple and could be brute forced quite easily (in our opinion). Once attackers successfully gain access, they will be shown a list of contents pulled from a small PostgreSQL database. We are currently filling this database with data that will lead to a future honeypot susceptible to SQL injection attacks, which of course, we will monitor and log. At the moment, we have placed fake user and administrator data in this database to give attackers incentives to consider SQL injection attacks. We are still considering what data would be most appropriate to store in this database as bait.[1] Fig 2 provides an overview of the honeynet as it stands today.

D. Activity Tracking and Logging

To track activity in our honeynet, each of our honeypots creates log entries that follow a common in-house log schema that highlights all of the pertinent attack data we receive. These log entries are stored in two forms: as a text log file entry and as a row in a table located in a PostgreSQL database.

First, requests are logged in a text file that is later transferred out of its Docker container for persistent storage. Then, using an instance of RabbitMQ [14], a message queue running in another Docker container, we send the log data to our honeynet queue. Once in the queue, the data is pulled, parsed, and inserted into a table located in our PostgreSQL database [15]. This database serves as redundant storage for all honeypot data. It also supports queries for analysis (via, among other tools, LCARS). This keeps our attack data organized, safe, and readily available for analysis.

Using our database in conjunction with LCARS enables us to visualize attack data and to perform graph and relational analysis. One of the most powerful methods of exploring the collected data is by generating a hive plot [16]. Using hive plots enables us to explore correlations, frequencies, and outliers that may have gone unnoticed in traditional-style visuals. With our honeynet and analytic software, we are able to provide cyber security experts with key insights on attackers' strategies, motives, and investments. In addition, we will be able to use our attack data to add depth to our honeynet as we continue to learn from our attack data.

IV. Preliminary Analysis

We have found, even at this early stage, multiple types of recurring attacks that include attempts to kill a PHP5 hash function and CGI (Common Gateway Interface) attempts to access Apache files. Additionally, we have noticed that some attackers use HTTP requests to attempt to load a resource, usually popular sites such as bing.com and twitter.com, by

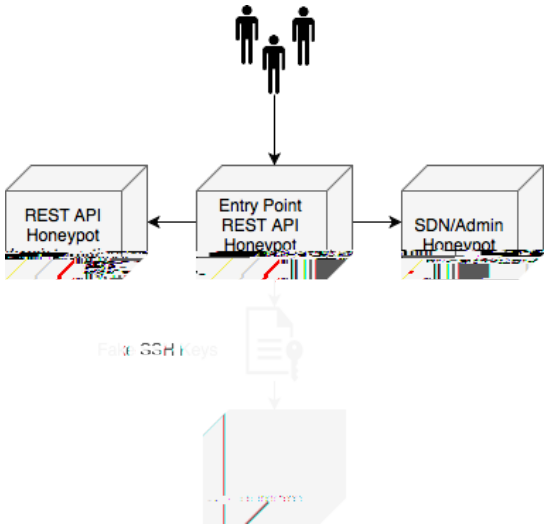

Fig. 2. Paths through the Honeynet

[1]We are open to suggestions and would love to hear from you.