# Observing Attacker Behavior in Different Types of Systems

HACS200 - Fall 2019

Group 2I
Jason Hu, Nicole Kaff,
Mausam Patel, and Emily Son

## Executive Summary

In order to inform decisions about how to protect a specific type of system from cyber attacks, we must first know if attackers change their behavior based on their virtual environment. With this research project, we seek to identify some of these behavior differences, should they exist, by answering the question: *Do differences in engagement, duration and/or interest exist among attackers in various environments?* In the context of this question, we define engagement as executing commands on a system, duration as the length of a continuous session on a system, and interest as logging into a system more than once. The null hypothesis for this experiment is: *There is no significant difference in engagement, duration, or interest exhibited by attackers between different environments.* We also define alternative hypotheses for each behavior aspect which are given by: *Attackers exhibit significantly higher levels of engagement in a personal computer than in a web server, database, or IoT device environment; Attackers spend a significantly longer duration in a personal computer than in a web server, database, or IoT device environment; Attackers exhibit less interest in an IoT device than in a database, web server, or personal computer environment.*

For this experiment, we constructed and deployed four honeypots, each emulating a different type of system. Specifically, we replicated the architecture of a database system, a web server, an IoT device, and a personal computer. Over the course of five weeks, we collected the session duration, commands run, and attempts to return of attackers in over 5,601 separate sessions across our four honeypots. Analysis upon this data allowed us to reject the null hypothesis and revealed that the environment of the system influences how much an attacker engages with the system and how likely they are to return.

# 1 Introduction

As modern cyber technology progresses and threats become ever more complex, many institutions, public and private, are struggling with the question of how to maintain a balance between user ease of access and security. Users expect faster and more transparent means of entry while increasingly persistent and sophisticated attacks demand for new means of authentication. In order to address this problem regarding the appropriate level of security, we investigate how attacker behavior changes according to the services running on a machine. Using observations from attacker behavior in four different types of systems over the span of 5 weeks, we attempt to establish differences between the attackers to each of our systems. Distinct threats require different methods of prevention. By recognizing which actions an attacker is likely to execute within a system, system administrators would be able to determine the level of security needed while limiting the damage done to fast and efficient user authentication.

# 2 Background Research

## 2.1 Vulnerable Environments

The prevalence of cyber attacks is not limited in scope to any one particular type of system. With the rise of the Internet of Things (IoT), once ordinary objects are now deeply embedded in an ever expanding global network of electronic devices. At the same time, there has been an influx of cyber attacks targeting these devices, with reports indicating the discovery of over 60,000 new pieces of IoT malware in the first quarter of this year (McAffee Labs, 2019). Because they are generally not under human surveillance, operate on a wireless network, and lack the power and sophistication to support a security system, IoT devices are particularly

susceptible to attacks by their very nature, with a study revealing that 70% of the most common IoT devices contain vulnerabilities (Hewlett-Packard, 2014). IoT devices are further appealing to attackers because they facilitate novel ways of wreaking havoc on victims, such as through manipulation of home security systems, thermostats, and other appliances (Abomhara & Koien, 2015).

More traditional specialty systems, such as databases, are also frequent targets of cyber attacks. Often ripe with personal or financial information, they make highly valuable targets, and recent highly publicized data breaches of companies like Equifax and Uber demonstrate the severity of such occurrences. According to an article by Shulman, databases are most vulnerable due to issues such as weak authentication and communication protocols, SQL injection, and backup data exposure (2006). Web servers are another specialty system that are common targets of cyber attacks ranging from DOS attacks to website defacement. Research by Symantec found that the top three most commonly compromised categories of web servers were dynamic DNS servers, gambling servers, and hosting servers (O'Gorman, 2019).

Personal computers are another at-risk category. Because many PC users lack technical skills and knowledge of cyber threats, this category of systems comprises a large portion of vulnerable computers. Vulnerabilities in personal computers often stem from user negligence, such as phishing incidents and failure to update software. When attackers have access to these systems, they can combine their collective resources to carry out larger attacks against other parties, including spam attacks or distributed denial of service attacks (Wash, 2010).

## 2.2 Attacker Behavior

Because these system environments each offer attackers different stimuli, we believe that by implementing four unique architectures in our honeypots we can elicit different patterns of behavior from attackers. This assumption is informed by existing research into hacker behavior theory. In his paper "A Conceptual Model of Hacker Development and Motivations," Van Beveren proposes that a hacker's interaction with a system is dependent on four factors: their perception of control on the system, their ability to focus on the attack, the ability of the environment to arouse their curiosity, and the intrinsic interest that they have towards the attack (2001). When all of these factors are present, the attack will presumably be the most severe. Of these factors, we are most interested in the first and third, namely the perception of control on the system and the ability of the environment to arouse an attacker's curiosity. In terms of our research, these factors are dependent on the computer environment, which can be manipulated on our end through the implementation of varying honeypot architectures.

## 2.3 Related Works

Existing literature indicates the feasibility of observing attacker behavior in simulated environments. Amongst these efforts, IoT honeypots appear to be a particularly popular choice. Researchers at cybersecurity company Symantec recorded nearly 60,000 total attacks on their IoT honeypot in 2018 and found that a majority of attacks utilized the antiquated Telnet protocol (O'Gorman, 2019). Other IoT honeypots include those by Minn et al. who studied malware families that targeted their honeypot, Dowling et al. who investigated attacks on a honeypot using the ZigBee protocol (often used by IoT devices), and Luo et al. who provide an IoT honeypot framework that incorporates machine learning techniques (2016; 2017; 2017).

IoT devices are not the only type of system with a precedent of being made into a honeypot decoy. In 2004, the NATO Research and Technology Organisation deployed a honeypot that emulated the functions of a database and in their work present the complexities involved in doing so (Čenys et al.). More recently, Rahmatullah et al. investigated security vulnerabilities in web servers by deploying a low interaction web server honeypot (2016). While we borrow from this existing research the concept of emulating a particular type of environment, our project varies from these previous works by emulating four unique environments simultaneously with a particular interest in observing behavioral differences between them.

### 3.1 Original Experiment Design

In the original design proposal, in order to determine whether differences existed in behavior among attackers in different environments, we defined three main factors that would be measured: engagement, duration, and interest. Engagement was quantified by the number of unique commands run by each attacker as well as whether or not they attempted to download malware. Duration was defined as the length of time an attacker remained in the honeypot before fully disconnecting. Interest was determined as whether or not an attacker attempted to reconnect to the honeypot after disconnecting from the honeypot. In order to observe the differences exhibited by attackers across various systems, four types of environments were selected. The four types of systems chosen were an elasticsearch database, an Apache web server architecture, an IoT camera, and a MacOS personal computer environment. Snoopy logger would be used to detect when an attacker managed to connect or disconnect from the honeypots as well as monitor which commands were run by each attacker. When an attacker disconnects all sessions from the

honeypot, the host would output the data collected by the Snoopy logger to a file on the host and recycle the honeypot immediately, ensuring that the honeypot would be ready for the next attacker. In the case that the honeypots or the host were taken down, a cron job would be implemented that would keep track of the health of the machine every hour.

### 3.2 Design Changes

Over the course of the project, many changes were made to the initial experiment design. In terms of the research questions, the three determinants of attacker behavior themselves remained unchanged, however, the indicators used to quantify them were modified. Originally, whether or not an attacker had attempted to download malware would factor into how we had defined engagement, however, it was later decided that the variable would not be used because it would be difficult to determine the statistical significance of the combination of the two variables used to define engagement, which were numerical and categorical in nature. In addition, while monitoring the data, it was observed that very few attackers were attempting to download malware to the honeypots, making any statistical statement about the differences between the distributions weak. Additionally, instead of quantifying engagement as the number of unique commands executed by an attacker, we decided to count the total number of commands run. The output of commands is often reliant on the current state of the machine, so the same command could function differently in different cases, which makes counting the number of unique commands weaker as an indicator of attacker engagement.

During the initial phase where the architecture of the honeypot and host network was being built, several issues became apparent. The first of which was that it would be difficult to emulate a MacOS system well without intruding upon proprietary software we did not have

access to. Instead of emulating, it was decided that the personal computer honeypot would attempt to emulate the system architecture of a Linux operating system. Another issue was that, due to the complexity of the honeypot environments that were designed, it would be difficult to recreate the architecture of the honeypots after the recycling process. In order to solve the issue, separate snapshots were taken of each honeypot in a fully functioning state. The snapshots could later be restored automatically, expediting the recycling process.

After monitoring the data being collected following deployment, we noticed many potential attackers being denied access to the honeypots. Initially, we had assumed that making the login passwords to the containers match the most common passwords for their corresponding types of environments would allow attackers to login easily. However, after seeing very little traffic to the honeypots during the first week of data collection, we changed the configuration of the honeypots to accept the first password any attacker attempted to login with. Another issue that was found regarding the low amount of attackers we were detecting was that attackers would often connect to the honeypots and not leave for upwards of ten hours, thus diluting the data by blocking other attackers during that time. In order to resolve the issue a timer was implemented in the recycling script that would limit how long an attacker could remain in a honeypot to an hour before disconnecting all their sessions.

Regarding data collection, after monitoring the iptables forward chain during deployment, we noticed that the chain would be refreshed after rebooting the host, resetting the number of packets and bytes the system had detected from blocked ip addresses back to zero. The problem was dealt with by implementing logic into the recycling script that would, instead of immediately blocking disconnected attackers, first send them to a logging chain that would

keep track of attackers that had tried to return to the honeypots, allowing the data to be stored separately.

### 4.1 Research Question

Our research aims to answer the question: *Do differences in engagement, duration and/or interest exist among attackers in various environments?* In the context of the research, an attacker is defined as any malicious actor that attempts to authenticate on one of the four honeypots. Furthermore, we quantify each behavioral quality in a way that can be statistically measured. Because executing commands is how a user interacts with a system, we quantify engagement as the total number of commands run by an attacker during their session. Duration is defined by the time elapsed between the start of an attacker's first ssh connection and their last disconnection. Furthermore, interest is determined by whether or not an attacker attempts to return to the honeypot after the conclusion of their session, as we believe this to be indicative of how appealing the environment is. We observe these aspects of attacker behavior and collect corresponding data in four high-interaction honeypots, each simulating one of four environments: a database, web server, Internet of Things device, and a personal computer. By deciphering behaviors in these different environments, we hope to be able to provide recommendations for how security can be tailored to the specific type of system in an authentic environment.

### 4.2 Hypotheses

We expect to see differences in attacker behavior between the four honeypots and environments, based on the existing research on attacker behavior theory and proofs of concepts for varying types of honeypots and environments.

Our three alternative hypotheses are as follows:

*$H_1$: Attackers exhibit significantly higher levels of engagement in a personal computer than in a web server, database, or IoT device environment.*

Since personal computers more often store private information about an individual and their life, we expect to see an increased number of attempts to run various commands in the personal computer environment to attempt to gather information about the individual or exploit their system.

*$H_2$: Attackers spend a significantly longer duration in a personal computer than in a web server, database, or IoT device environment.*

Due to the valuable and sensitive information a personal computer could hold about the profile of the owner of the computer, we believe an attacker would spend the most time in the personal computer environment, especially since each personal computer is different in their file structure and naming conventions.

*$H_3$: Attackers exhibit less interest in an IoT device than in a database, web server, or personal computer environment.*

Considering the limited functionality of IoT devices in comparison to more complex systems, such as databases, web servers, and personal computers, we suspect that attackers will be generally less interested in this type of environment.

Our null hypothesis is:

*$H_o$: There is no significant difference in engagement, duration, or interest exhibited by attackers between different environments.*

Despite all the factors represented by attacker behavior theory and concepts about different environments, there could exist no significant difference in the engagement, duration, or interest exhibited by attackers across the different environments.

# 5 Experiment Design

## 5.1 Container Set Up

We have been provided four containers to use during the research process, with four distinct IPs. As previously mentioned, the four IP addresses were ultimately placed in the same subnet 128.8.238.0/16 to eliminate another variable between the four honeypots. The network will be configured as per the recommendations of the advising staff (refer to **Appendix C**). These four containers were set up to emulate four different environments: a web-server, an Internet of Things device, a database server, and a personal computer.

### 5.1.1 Database

The first honeypot environment simulates a database environment, built with a MySQL database and populated with fake financial data, excluding any personally identifiable information. Tables were created in the database with 1000 randomly generated data entries for each table to populate the database. Although we cannot collect the commands an attacker might have run inside of the MySQL database itself since MySQL provides its own shell and command line, we collected their commands that were run inside of the container instead.

### 5.1.2 Web Server

The second honeypot resembled a web-server and hosted a simple Apache web server. A fake website was set up to mimic a e-commerce website and included elementary pages such as “About Us” “Locations,” “Contact Us,” and “Careers” pages. These pages were filled using fake

phone numbers, addresses, and location titles, and included input elements to resemble interactive components on a web page such as text fields and buttons on the "Contact Us" pages. The site also included a menu bar, and pages were linked to one another. Stylesheets (CSS) files were utilized to design the web pages to further mimic a real company website. The stylesheets and HTML files were placed in the var/www/html directory, which only existed when Apache was installed onto the container. At the end of every attacker session, the recycling script ensured that the HTML and CSS files were reset to their original structure and content in the event that an attacker altered the site. The ability to interact with the actual site was tested by manually checking the site by entering the IP address in the URL bar, and we confirmed that the display, design, and links on the page worked as intended.

### 5.1.3 IoT Device

The third honeypot represented an Internet of Things (IoT) device. More specifically, it represented a basic CCTV camera that leveraged an Apache web server. We relayed live stream footage from the University's surveillance system as recommended by the advising staff. This footage was accessed on the A. James Clark School of Engineering's Engineering Information Technology website (https://eit.umd.edu/webcams). The site displays live public footage of the outside of the Kim Engineering building, the Clark Building, and McKeldin Mall. For the purposes of this project, the footage from the IP camera displaying the live footage from the Kim Engineering building was relayed. The display url was extracted by inspecting and viewing the page source of the UMD feed. We discovered the University used an AXIS Q1615 Network Camera, and our honeypot would resemble this type of camera. We relayed the footage coming from the University's IP camera from IP 129.2.122.140 into the container and masked it with the

container's IP address. A script and html file were written to allow the container to properly relay and format the footage to mimic its source display everytime the container was recycled. The commands to install and uninstall Apache were included in this script, as well as commands to copy over the HTML file located on the proxmox that masked the CCTV footage.

### 5.1.4 Personal Computer

The fourth honeypot emulates a personal computer environment, which we defined with a filesystem that mirrors a Linux system. In the list of directories in the home directory of the container, there is another directory we created named "home" which holds subdirectories for each user of the personal computer: Ashley, Jack, Karen, and Robert. Each user directory consists of the same subdirectories: Applications, Desktop, Documents, Downloads, Movies, Music, Pictures, Public. Because these subdirectories are common ones found on many personal computers, we chose to populate these with fake file data through a Python Faker Library and Python scripts. The Desktop and Downloads directories consist of a variety of fake files with different attachments, the Documents directory is made up of .docx file names, the Movie directory stores .mov files, the Music directory holds .mp3 files, and the Pictures directory lists .jpg files. This file and directory structure is mimicked for each user of the personal computer.

## 5.2 Firewall

In addition to implementing the permanent firewall rules provided by the Division of IT, we configured our container scripts to dynamically adjust the firewall at the start and end of an attacker's session. To do this we utilized two firewall chains: the FORWARD chain and a LOGGING chain, the latter of which was of our own creation. When our recyclings scripts detect that an attacker has compromised a container, two rules are inserted at the top of the

FORWARD chain. One rule blocks all traffic to that container while the other rule allows only that attacker's IP address to access that container. An attacker is permitted to create multiple concurrent sessions, but once their last session is disconnected, the two rules are removed from the FORWARD chain and a new rule is inserted that will redirect all traffic from that IP address to a chain called LOGGING. This rule is also appended to the end of the firewall script so that on reboot, attackers will still be properly blocked. More details on host reboot are discussed in **Section 5.4**.

The LOGGING chain was created to facilitate the logging of packets from users that have previously had a session in a particular container and are attempting to re-enter that same container. In this chain there is a logging rule for each container that creates a log entry with the container number and the source IP address. These log messages can then be viewed with the *dmesg -H -w* command. There is a fifth rule in this chain at the bottom that promptly drops all packets after they have been logged by the appropriate container's logging rule. In summary, this chain allows us to log the occurrence of attacker's attempting to return to the same container without actually letting them make a connection. Attackers are, however, permitted to enter a container that they have not previously entered into. In other words, an attacker may have one session in all of our environments. More information on attacker overlap between containers can be found in **Section 3 of Appendix C**.

### 5.3 Recycling

There were four recycling scripts, titled recycling10X.sh, where X corresponds to the container number, each set to constantly run in the background and monitor the auth.log files of their corresponding containers. One such script can be seen in full in **Appendix C**. Whenever an

attacker made a valid connection to the honeypots for the first time, the scripts would create iptables rules that would block all other attackers, create a file that would store all of the lines outputted to the container's auth.log file, and enter the current UNIX timestamp, so that the data could be processed afterwards. The scripts kept track of the number of sessions that attackers had created, adding one for every connection and subtracting one for every disconnect. When the attacker had fully disconnected from the honeypot, either by willingly disconnecting and having their session counter reach zero, or being forcefully disconnected after being connected for an hour, the recycling scripts would enter the current UNIX timestamp into their corresponding data files and make the appropriate changes the firewall rules as described in **Section 5.2**. Afterwards, the scripts would call separate scripts that would reset the honeypots for the next attacker.

The restarting scripts, titled restart10X.sh, would first kill the tail process run by the cycling script, then stop, unmount, and rollback their appropriate containers to snapshots we had manually taken of the honeypots in an unaltered state. The containers would then be mounted to the host and started up. After finishing the restart process, the script would call the original recycling script, restarting the process and getting ready for the next attacker.

### 5.4 Host Monitoring and Setup

This experiment was conducted on a Proxmox VM Host equipped with 16GB of RAM and 32GB of storage. To monitor the health of the system, a bash script was created to track the level of available RAM, available disk space, system load, and incoming and outgoing traffic on the host as well as on each of the containers (see health.sh in **Appendix C**). This script was configured in the host's crontab to execute once every hour, and utilizes a google spreadsheet API to push all of the collected data to a health log. Using the google scripting interface, an

alerting script was created to read the latest set of health data from the spreadsheet every two hours. If either the Proxmox host or the containers were found to be operating at unsafe levels, an email would be sent to all members of the group alerting us of the particular issue.

Also running on the Proxmox host was a script to monitor the occurrence of attacker compromises to the four containers. This script read from a file containing a list of attacker IP addresses and their associated container and sent this information to another external google sheet (see compromise_monitor.sh in **Appendix C**). Again using the google sheets interface, a script was created to send an email to the group indicating which container had been compromised and by which IP address. This script was configured to execute each time the sheet was updated, which enabled us to track compromises in near-real time. See **Appendix A** for the drawbacks of this script pertaining to trigger quotas.

In order to ensure that our host configuration would persist between reboots (should one be necessary), we configured /etc/rc.local to start up all the necessary scripts and restore the honeypots and firewall. In this script, which runs automatically when the Proxmox host starts up, unmounts all the containers, runs the firewall script provided by DIT, implements the prerouting and postrouting rules, remounts the containers, and runs the recycling and monitoring scripts. As previously mentioned, we add our logging rules to the firewall script to ensure that attackers will still be blocked from containers they have already been in even after a system reboot. The entire contents of rc.local can be viewed in **Appendix C**.

### 5.5 Backups

By default, all of our collected attacker data was stored directly on the Proxmox host. To ensure that the integrity of our research would not be compromised due to loss of data, regular

internal and external backups were made. Internal backups were made by creating a local copy of the data in a distinct Data Backups directory on the host. Backups were distinguished by a different letter (BackupA, BackupB, etc.) and in total we had nine data backups. In the backups, all files were renamed to have a .txt extension so that they could be portable onto other systems. Then the files were secure copied from the proxmox host to a physical machine. From here they were also copied to a secondary external hard drive and uploaded onto Google Drive. Then the original data files on the Proxmox host were deleted so that subsequent backups would not contain overlapping data. Following this standard procedure for all of our backups, there were ultimately four locations in which the attacker data was stored: the Proxmox host, a physical computer, a physical external hard drive, and on the Google Drive.

### 5.6 Limitations

Because this research project was sanctioned by the University of Maryland, we were required to operate within the confines of the Division of Information Technology's policies. These constraints manifest themselves in the implementation of certain firewall policies that could potentially have impacted the volume and diversity of the data collected. We were also prohibited from using any personally identifiable information, regardless of whether it was real or synthetic, and it is possible that this detracted from the appeal of our honeypots to prospective attackers. It is also worth noting that the public IP addresses used in the project are associated with the university and that this may be known by any potential attacker. Furthermore, we were subject to time limitations due to the nature of the assignment. The project was implemented over a semester-long period, and with several of these weeks necessary for the initial construction and configuration of the honeypots as well as for statistical analysis, our data

collection window was limited in length. All of these factors are important considerations in evaluating our ability to answer our research question during this project.

As for the architecture of our honeypots, we were limited in regards to how accurately we could emulate the four environments that we have selected. Ultimately our honeypots were only able to give the appearance of being a particular kind of server, and we had no way of knowing what any actual hacker actually believed. It's also possible that a significant portion of our traffic was from automated attackers that indiscriminately performed their behavior regardless of what illusioned environment they found themselves in. This may be a setback in the sense that we won't be gaining insights into the behaviors of solely human attackers. However, this simultaneously can work to our advantage by representing an attacker population that is perhaps more akin to the threats that real systems are facing every day.

## 6 Data Collection

Over the course of the five weeks that the honeypots were deployed, we recorded 5,601 compromises across our four containers. Our database honeypot had 1,339 attacks, our web server had 1,553 attacks, our IoT device had 1,389 attacks, and our personal computer had 1,320 attacks. As previously mentioned in **Section 4**, we are interested in three attributes of attacker behavior towards our environments. These are engagement, interest, and session duration and we quantify them as number of commands run, attempting to return to a container, and time elapsed between first connection and final disconnection.

For each of these compromises, we collected starting and ending timestamps and commands run during the session. This data was stored in a file with the IP address of the

attacker as it's name in a specific directory on the proxmox host for that container's data. This naming convention was ideal for us as it allowed us to quickly generate a list of all IP addresses that had compromised a particular container and was not an issue since an attacker could only access each container once. As previously discussed in **Section 5.2**, we also kept a log of attackers attempting to return to a container.

### 6.1 Commands Run

We changed Snoopy's configuration to log additional descriptive information about each command run in order to differentiate between the commands run by the attacker versus the automatically generated commands run by the OS in the background. Collection of these entries from Snoopy Logger was implemented in the recycling script, which was already following the auth.log file to which Snoopy Logger writes its output. Each entry produced by Snoopy Logger was appended to the respective attacker's session data file. An example of a typical Snoopy logger entry is shown below. Note that it includes the command and its arguments as well as the IP address of the user.

```
Nov 9 21:37:35 CT102 snoopy[638]: date: 2019-11-09T21:37:35+0000 cmdline: sleep 15s cwd: /root login: root
filename: /bin/sleep env_VAR: (undefined) env_all: XDG_SESSION_ID=19676,SHELL=/bin/bash,
SSH_CLIENT=89.222.181.58 45478 22,USER=root,MAIL=/var/mail/root,PATH=/usr/local/sbin:
/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games,PWD=/root,LANG=C,SHLVL=1,HOME=/ro
ot,LOGNAME=root,SSH_CONNECTION=89.222.181.58 45478 172.20.0.3 22,XDG_RUNTIME_DIR=
/run/user/0,_=/bin/sleep
```

### 6.2 Timestamps

Collection of timestamp data was also implemented in the recycling script. When an attacker connected to a container, the unix timestamp was then pushed to the data file corresponding to that attacker's session record. Likewise, when the attacker disconnected their last session, the unix timestamp was pushed to the very end of the data file. A complete session's

log thus contained a list of all the commands run during the session sandwiched between the session start time and session end time. We elected to use a unix timestamp, generated with the command *date '+%s'*, instead of a traditional timestamp to make it easier to calculate the difference between the start and end times of the session (i.e. the session duration). Furthermore, the timestamp was generated on the Proxmox host rather than on the respective containers to prevent inconsistencies that might have resulted from an attacker altering the time on a container.

### 6.3 IP Tables Logs

As previously stated in **Section 5.2**, our firewall was configured with a logging chain that logged the occurrence of attackers attempting to reconnect to a container that they had previously been in before dropping the packets completely. These log messages could then be viewed with the *dmesg -H -w* command. A script called returning_attackers.sh (in **Appendix C**) was written to separate these log messages from other types of log messages and to sort the log messages by their respective containers. This script followed the output of the dmesg command and appended the log entries to a file for the appropriate container's returning attackers log. This script ran continuously in the background during deployment so that no log messages were missed. An example of a typical IP tables log entry is shown below. As we can see, the entry identifies the particular container that the attacker tried to access, as well as the source IP address of the packet passing through the logging chain.

```
[Nov16 09:09] CONTAINER 101: IN=enp4s1 OUT=vmbr0 MAC=52:54:00:20:47:18:18:66:da:aa:25:d3:08:00
SRC=113.23.26.24 DST=172.20.0.2 LEN=52 TOS=0x08 PREC=0x20 TTL=101 ID=5362 DF PROTO=TCP
SPT=42233 DPT=22 WINDOW=8192 RES=0x00 SYN URGP=0
```

### 6.4 Preprocessing

Before any analysis could be performed on our collected data, we conducted some minor preprocessing to transform it into usable data points and move it to an external location. This was accomplished primarily through the use of bash scripts designed to parse the data files. To obtain the session durations, a script called time_parser.sh (included in **Appendix C**) was written to read the first and last lines of each attacker session data file in a directory, which were the starting unix time and ending unix time, respectively, and output the difference between these two times. The complete lists of durations, separated by container, were then copied to an external spreadsheet. Likewise, a script called command_parser.sh (also included in **Appendix C**) was created to count the number of lines in each attacker session data file that contained the phrase "SSH_CONNECTION." This keyword was chosen because it was present in all the snoopy logger entries that contained commands run by the attacker, as opposed to commands run in the background by the OS that snoopy logger also logged to our data files. The complete lists of the number of commands run, separated by container, were also copied to an external spreadsheet.

Our approach to parsing the IP tables logs was different as we sought to obtain four lists, one for each container, of the IP addresses that had attempted to return to a particular container. Our IP tables logs were separated by container but not by IP address, so a script was written for each container called cont10X_return_parser.sh (included in **Appendix C**) that pulled the IP address from each entry and sent it to a separate list for that particular container. This list of IP addresses was sorted and then duplicates were eliminated so that all that remained was a list of all the IP addresses that attempted to return to a particular container. This list was then compared

to the list of all IP addresses that had accessed that container, with the difference in number between the two lists being the number of attackers that did not make any attempt to reconnect to the particular honeypot environment.

## 7 Data Analysis

After collecting and processing the command, timestamp, and iptables logging data generated by attackers across the four honeypots, there were three data points for each attacker.

The first, which counted the total number of commands run by each attacker, was separated into four distinct groups based on their corresponding honeypot, which were each represented by a single discrete numerical variable that measures the attackers' engagement. In order to test whether significant differences existed between the four groups of data, a one way analysis of variance (ANOVA) test was used. For the final command data, which has sessions with zero commands run removed, each group was collected independently from the others, matching the first of the assumptions made by a one way ANOVA test. The second and third assumption, which state that variable residuals should be normally distributed for the data should the variances of the groups should be equal, are not true for the data (see **Appendix B.1**), however, one-way ANOVA is a robust test in regard to violations of assumptions of normality (Kirk, 1988). The resulting one-way ANOVA, run with a 0.05 alpha level, produced an $F$ value of 10.874 for a critical $F$ value of 2.609 as well as a $P$-value of 4.337E-07. Because the $F$ value is greater than the corresponding critical value, the null hypothesis can be rejected. The standard error for the four samples is .437. The mean number of commands run for the personal computer honeypot was at least 3.83, or 8.76 standard errors, greater than the means of the other

honeypots, meaning that there is evidence that attackers run more commands within personal computer environments than other environments.

The second, which measured the difference between timestamps recorded at the beginning and end of each attacker session to measure how long each attacker remained in the honeypots, was similarly separated into four distinct groups based on their corresponding honeypot. Each data point was represented by a single continuous numerical variable. A one-way ANOVA test could also be used to test whether significant differences existed between the four groups of duration data. The final duration data were approximately normally distributed and each group was collected independently from the others, matching the first two assumptions made by a one-way ANOVA test. The last assumption, that the variances of the populations are equal, is not true for the data, with the variance for the web server environment data being approximately 28% less than the other environments, however, the loss in power of the ANOVA test with unequal variances sharply decreases as the sample size decreases (Tiku, 1971). For the duration data, the resulting one-way ANOVA test, run with an alpha level of 0.05, produced an $F$ value of 1.682 for a critical $F$ value of 2.606 as well as a $P$-value of 0.169. In this case, because the $F$ value for the ANOVA test is less than the critical value needed to reject the null hypothesis, the null hypothesis cannot be rejected, meaning that there is no evidence within the data that the length of time an attacker spends within a system depends on the environment for the system.

The final data point, which recorded whether or not attackers who had already connected and disconnected from the honeypots had tried to reconnect to the honeypots, was also separated into four distinct groups based on the honeypot the attackers tried to access, however, instead of

being a discrete or continuous numerical variable that could be tested using a one-way ANOVA test, the returning attacks data is a categorical variable. In order to test for differences among the four groups of categorical data, a chi-square test for goodness of fit was used. The variable that measures whether an attacker attempts to reconnect is categorical and each value in the contingency table is at least 5, matching two of the three assumptions made by a chi-square goodness of fit test. However, the data is only a simple random sampling when considering attacks to the University of Maryland network. The resulting chi-square test, run with an alpha level of 0.0.5, produced an $F$ value of 23.439 for a critical $F$ value of 7.815 as well as a $P$ value of 3.27E-05. Because the $F$ value exceeds the needed critical $F$ value needed to reject the null hypothesis, the null hypothesis can be rejected, meaning that there is evidence that the content of a system affects the likelihood of an attacker to return to the system.

# 8 Conclusion

## 8.1 Results

The findings of this study have important implications within the field of cyber security and provide several insights into how specific types of systems can be secured. Distinct threats require different methods of prevention. By recognizing which actions an attacker is likely to execute within a system, system administrators would be able to determine the level of security needed while limiting the damage done to fast and efficient user authentication. As demonstrated by the data, there is evidence that attackers run more commands within personal computer environments than other environments. This indicates that threats to personal computers should not be disregarded. Security professionals should treat personal computers with the same caution

used for more complex or commercial systems. The data also demonstrates that there is evidence that the content of a system affects the likelihood of an attacker to return to the system. System administrators may wish to invest a varying level of security depending on the type of the system. What types of specific content of the system would pose a greater risk and require a greater investment remain to be identified, and are highlighted as a future opportunity for research below.

### 8.2 Future Work

In future iterations, or in future work related to the differences in attacker behavior within different types of environments, there are multiple opportunities to expand upon this research project. For instance, both Snoopy and Man in the Middle software may be used to collect more data on the behavior of the attackers and the commands run. This would offer other areas of analysis and perhaps other interesting insights into attacker behavior such as navigation between directories. Furthermore, with more time and resources, there is an opportunity to create more realistic and interactive environments for each of the honeypots. With access to paid web servers, and a more detailed breakdown of the configuration of the environments, such as the IP camera, more convincing environments may be developed. Designated ports may also be introduced as a part of a more detailed, expert configuration. As previously mentioned, while the data suggests there is a difference in interest depending on the type of environment, focus on analysis of what aspects of the data itself causes attackers to wish to return is outside the scope of the current project. We propose this as an opportunity for further research with interest measured against variable of what data is used instead of the type of the environment.

# Appendix A

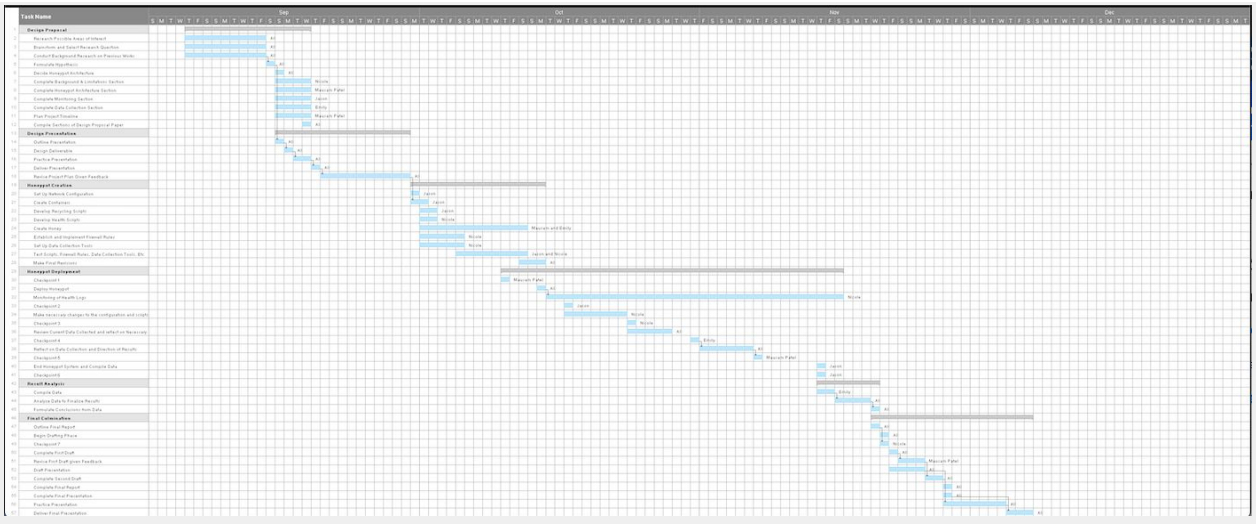
## A.1 Lessons Learned

Throughout the course of this project, we learned crucial lessons about not only about cybersecurity research, but also the dynamics of working on a research team to accomplish a long-term goal and complete a project together. For example, we learned valuable lessons in project management in the context of a research project. We learned that while it is crucial to effectively plan the milestones of the project and deliverables by using a gantt chart as a tool, it is also important to actually use and refer to it. In terms of project management, we generally divided up the work fairly and in a way that played to certain team member's strengths but also allowing them room to contribute in other areas as well. However, we did learn that although we assigned certain team members to a specific role at the beginning, we ended up changing around the roles a little based on how the research was going and what needed to be done, creating more flexibility and efficiency.

We also learned that research is unexpected, and the team is not abstained from responsibility during the data collection period. We faced setbacks and challenges, and it was our responsibility to act accordingly in a timely manner. By staying vigilant, we were able to address problems early on, and prevented serious data collection problems. This was especially important due to the short timeline of the project, we simply could not afford to lose weeks of data. We learned that some ideas, while interesting, were too ambitious for the time and resources given, and that we should be prepared to eliminate some aspects of our original plans as new information arises.

**project management stuff??? we should put the gantt chart.**

**Gantt Chart**



## A.2 Project Feedback

There were several things that we appreciated about how this project was conducted. First, we liked having access to the ACES Honeypot Project Wiki and we found this to be a valuable resource as we were setting up our host and honeypot environments. We also appreciated that there was a large teaching staff and that we could communicate with them conveniently on slack. Furthermore, we also liked that we were allowed the opportunity to work in our groups on the project for a majority of the class time each week after deployment, as this ensured that we could meet at least once as a group each week without having to coordinate schedules.

There were a few areas that we identified as areas that could be improved and we would like to offer some potential suggestions. First, we felt that the first two weeks of class would have been better spent working on the project proposal. Although we recognize the value of the

project management material covered during this time, we felt spending an entire week on it was not necessary and that it cost us time that we could have spent becoming acquainted with our groups and brainstorming research questions. As a result, when we were tasked with writing the entire project proposal in one week, as well as creating and rehearsing a presentation, we felt very pressed for time and moderately overwhelmed. Getting our groups and the project proposal assignment the first week of class would have greatly alleviated this stress as well as allow the work to be of even greater quality.

We also thought that the checkpoints could be more structured. Although we like the concept of the checkpoints (getting one-on-one time with the instructors and TAs), there were times where we were unsure what to expect from a checkpoint in the sense that we sometimes felt the staff guided the checkpoints but other times we were expected to lead. Furthermore, we frequently found that information in the executive summary had to be repeated during the checkpoints, which may have taken away time from discussing other important issues, and that sometimes we were told by the staff that they would follow up with us on a particular point but they never did. We understand that there were many groups and that it is unreasonable to expect all of the staff to remember everything each week, so we thought one solution might be to assign a certain number of groups to each TA. This would allow that particular TA to become more familiar with the individual research questions and can give each group more customized help. Of course, one drawback would be that the TAs each have different strengths and specialties so it could be hard to divide up the groups in this manner. We also thought it would be very helpful to have one TA remain in the classroom during the checkpoints to assist groups as if it were office hours.

### A.3 Surprises

After we received a new IP address from the teaching staff, modified the honeypots to accept any password on the first attempt, and implemented a session timer, the number of sessions that we were collecting increased considerably. Our group was quite astonished by the sheer volume of the data, and at one point we even ran out of storage on the Proxmox host. However, with such a large quantity of data, we were surprised that a relatively low number of attackers actually ran commands. Only 38% of attackers ran commands in the database honeypot, 31% in the IoT device, and 34% in the personal computer. The web server honeypot did, however, see 51% of it's attackers run commands during their session. The most interesting session that we found was when a particular attacker attempted to access a website called minexmr.com, which appeared to be a website associated with mining.

### A.4 Pitfalls and Failures

During the first week of deployment we ran into several issues that could have been avoided had we taken additional precautions. First, we did not think to ask the teaching staff for a fourth IP address in the same subnet as the other three. Unlike other groups, it was especially important for us to have all of our IP addresses in the same subnet because we needed to ensure that all of our honeypots had an equal chance of being reached by any particular attacker. We also did not realize that the default Snoopy Logger output did not provide us enough information to be able to distinguish between commands run during the attacker's session and commands run in the background, and having to reconfigure Snoopy Logger to allow us to do so cost us some valuable deployment time and data. Another issue we ran into was that during the peak of our

deployment time, we used up our Google Scripts quota for sending email alerts. However, at that point in deployment we were confident enough in our infrastructure that we determined the email alerts for compromises were no longer needed.

# Appendix B

## B.1 Results of Statistical Tests

| Container | Count | Attackers that Attempted to Return | Attackers that Ran Commands | Avg. # Commands | Avg. Duration (sec) |
|---|---|---|---|---|---|
| **Database** | 1339 | 905 (67.6%) | 503 (38%) | 52 | 356 |
| **Web Server** | 1553 | 1080 (69.5%) | 787 (51%) | 56 | 325 |
| **IoT Device** | 1389 | 872 (62.8%) | 432 (31%) | 54 | 362 |
| **PC** | 1320 | 934 (70.8%) | 444 (34%) | 61 | 383 |

### Duration Statistics

| *Database* | |
|---|---|
| Mean | 356.2323 |
| Standard Error | 19.58393 |
| Median | 137 |
| Mode | 2 |
| Standard Deviation | 716.6219 |
| Sample Variance | 513547 |
| Kurtosis | 13.26124 |
| Skewness | 3.637504 |
| Range | 3659 |
| Minimum | 0 |
| Maximum | 3659 |
| Sum | 476995 |
| Count | 1339 |

| *Web Server* | |
|---|---|
| Mean | 324.7721 |
| Standard Error | 15.40772 |
| Median | 152 |
| Mode | 2 |
| Standard Deviation | 607.1891 |
| Sample Variance | 368678.7 |
| Kurtosis | 19.32636 |
| Skewness | 4.238587 |
| Range | 3659 |
| Minimum | 0 |
| Maximum | 3659 |
| Sum | 504371 |
| Count | 1553 |

| *IoT Device* | |
|---|---|
| Mean | 361.9309 |
| Standard Error | 20.23326 |
| Median | 126 |
| Mode | 2 |
| Standard Deviation | 754.0791 |
| Sample Variance | 568635.2 |
| Kurtosis | 11.66618 |
| Skewness | 3.480562 |
| Range | 3658 |
| Minimum | 1 |
| Maximum | 3659 |
| Sum | 502722 |
| Count | 1389 |

| *Personal Computer* | |
|---|---|
| Mean | 383.1576 |
| Standard Error | 20.9143 |
| Median | 137 |
| Mode | 2 |
| Standard Deviation | 759.8544 |
| Sample Variance | 577378.6 |
| Kurtosis | 11.2618 |
| Skewness | 3.400997 |
| Range | 3654 |
| Minimum | 1 |
| Maximum | 3655 |
| Sum | 505768 |
| Count | 1320 |

### Commands Run Statistics

| *Database* | |
|---|---|
| Mean | 52.93638171 |
| Standard Error | 0.80122706 |
| Median | 60 |
| Mode | 60 |
| Standard Deviation | 17.96964928 |
| Sample Variance | 322.9082952 |
| Kurtosis | 2.94911801 |
| Skewness | -2.191450669 |
| Range | 75 |
| Minimum | 2 |
| Maximum | 77 |
| Sum | 26627 |
| Count | 503 |

| *Web Server* | |
|---|---|
| Mean | 56.12579 |
| Standard Error | 0.489107 |
| Median | 60 |
| Mode | 60 |
| Standard Deviation | 13.72117 |
| Sample Variance | 188.2704 |
| Kurtosis | 9.462541 |
| Skewness | -3.33792 |
| Range | 75 |
| Minimum | 2 |
| Maximum | 77 |
| Sum | 44171 |
| Count | 787 |

| *IoT Device* | |
|---|---|
| Mean | 53.96528 |
| Standard Error | 1.034286 |
| Median | 60 |
| Mode | 60 |
| Standard Deviation | 21.49723 |
| Sample Variance | 462.131 |
| Kurtosis | 48.10382 |
| Skewness | 2.674205 |
| Range | 309 |
| Minimum | 2 |
| Maximum | 311 |
| Sum | 23313 |
| Count | 432 |

| *Personal Computer* | |
|---|---|
| Mean | 59.95721 |
| Standard Error | 1.378953 |
| Median | 60 |
| Mode | 60 |
| Standard Deviation | 29.05635 |
| Sample Variance | 844.2713 |
| Kurtosis | 262.0656 |
| Skewness | 14.83524 |
| Range | 589 |
| Minimum | 2 |
| Maximum | 591 |
| Sum | 26621 |
| Count | 444 |

## Analysis of Variance Test Results - Total Commands Run

Anova: Single Factor

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| Database | 503 | 26627 | 52.9363817 | 322.908295 |
| Web Server | 787 | 44171 | 56.1257942 | 188.270415 |
| IoT Device | 432 | 23313 | 53.9652778 | 462.131042 |
| Personal Computer | 444 | 26621 | 59.9572072 | 844.271302 |

ANOVA

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 13327.9646 | 3 | 4442.65486 | 10.8743725 | 4.33744E-07 | 2.60901864 |
| Within Groups | 883271.177 | 2162 | 408.54356 | | | |
| Total | 896599.141 | 2165 | | | | |

## Analysis of Variance Test Results - Duration

Anova: Single Factor

SUMMARY

| *Groups* | *Count* | *Sum* | *Average* | *Variance* |
|---|---|---|---|---|
| Database | 1339 | 476995 | 356.232263 | 513546.957 |
| Web Server | 1553 | 504371 | 324.772054 | 368678.653 |
| IoT Device | 1389 | 502722 | 361.930886 | 568635.23 |
| Personal Computer | 1320 | 505768 | 383.157576 | 577378.648 |

ANOVA

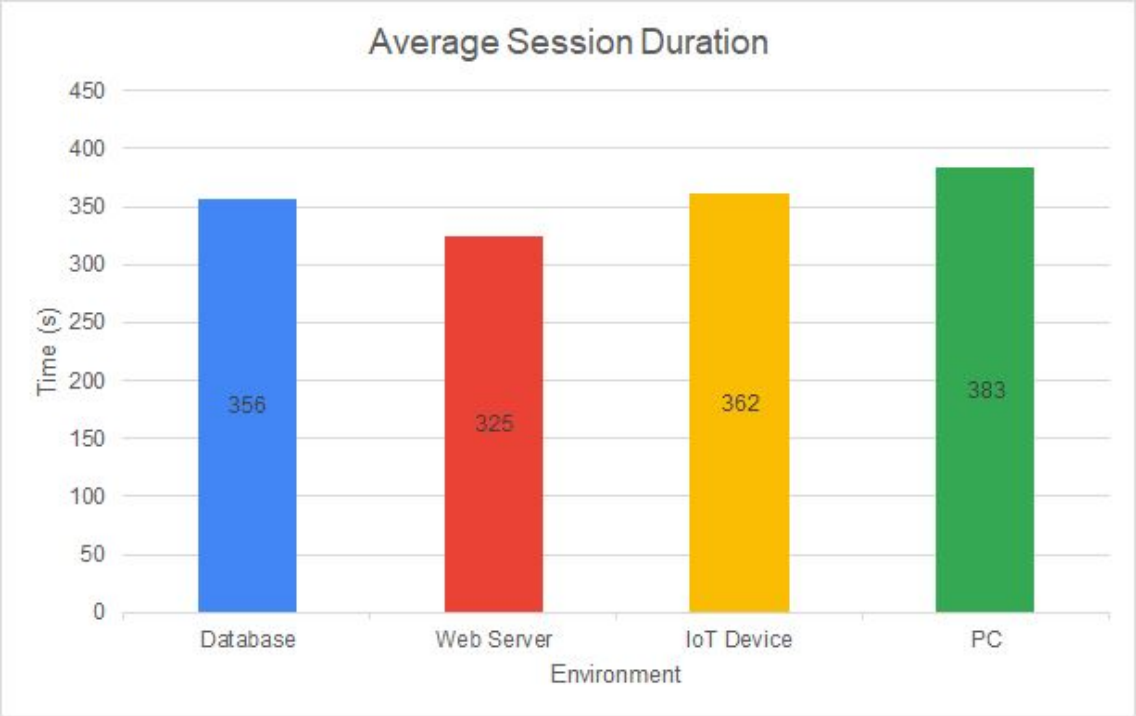
| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 2533936.09 | 3 | 844645.363 | 1.68229151 | 0.16852785 | 2.60649571 |
| Within Groups | 2810143235 | 5597 | 502080.263 | | | |
| Total | 2812677171 | 5600 | | | | |

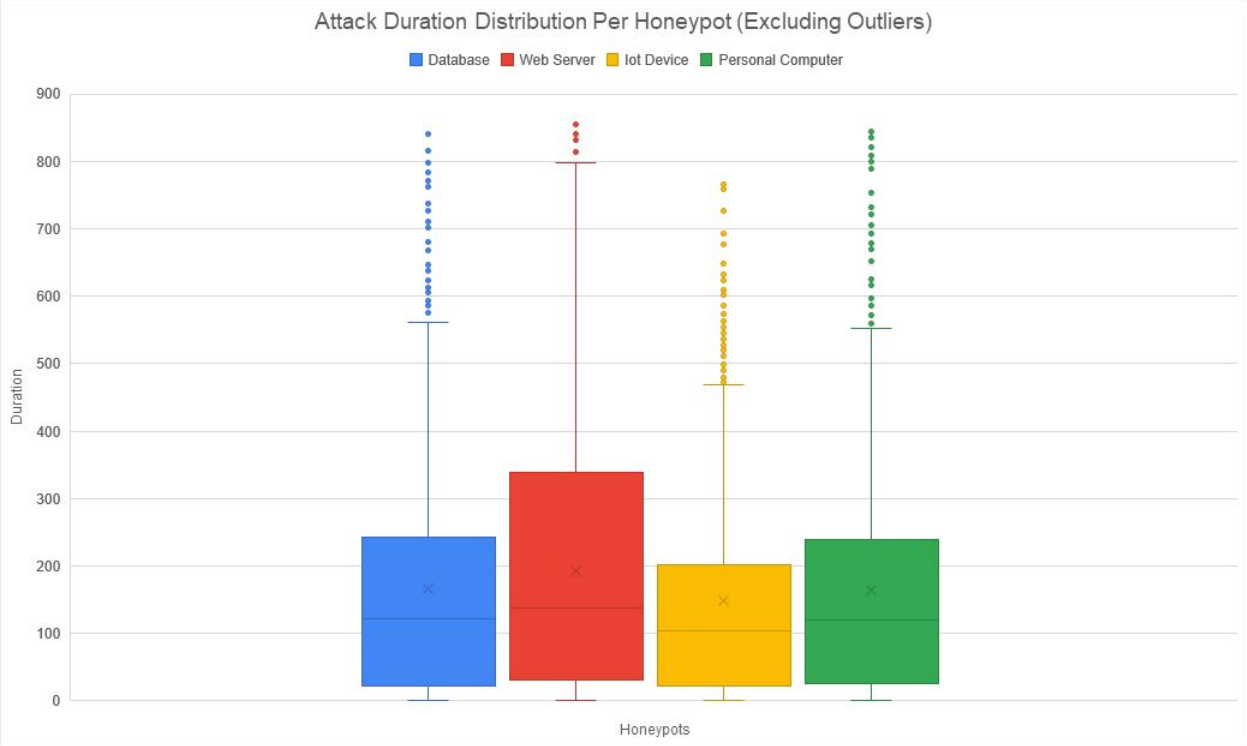
## Chi-Square Test Results - Attempts to Return

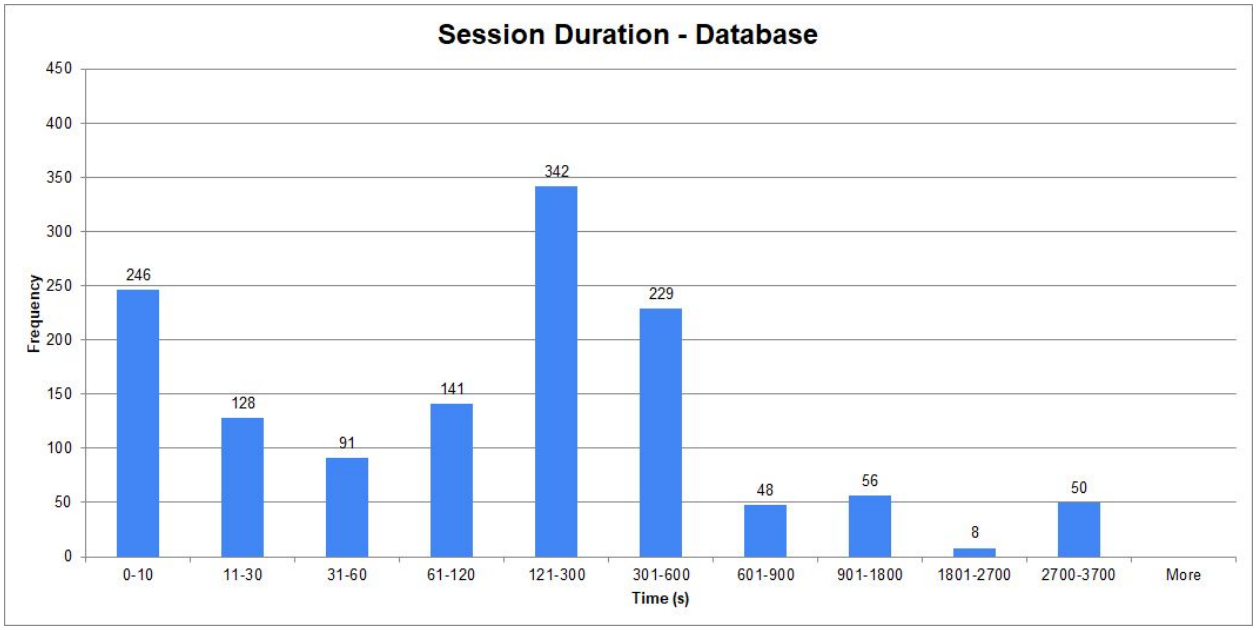
| **Environment** | **Total Number of Attackers** | **Number of Attackers that Attempted to Return** | | **Ratio** |
|---|---|---|---|---|
| Database | 1339 | 905 | 434 | 67.59% |
| Web Sever | 1553 | 1080 | 473 | 69.54% |
| IoT Device | 1389 | 872 | 517 | 62.78% |
| Personal Computer | 1320 | 934 | 386 | 70.76% |
| | | 3791 | 1810 | |

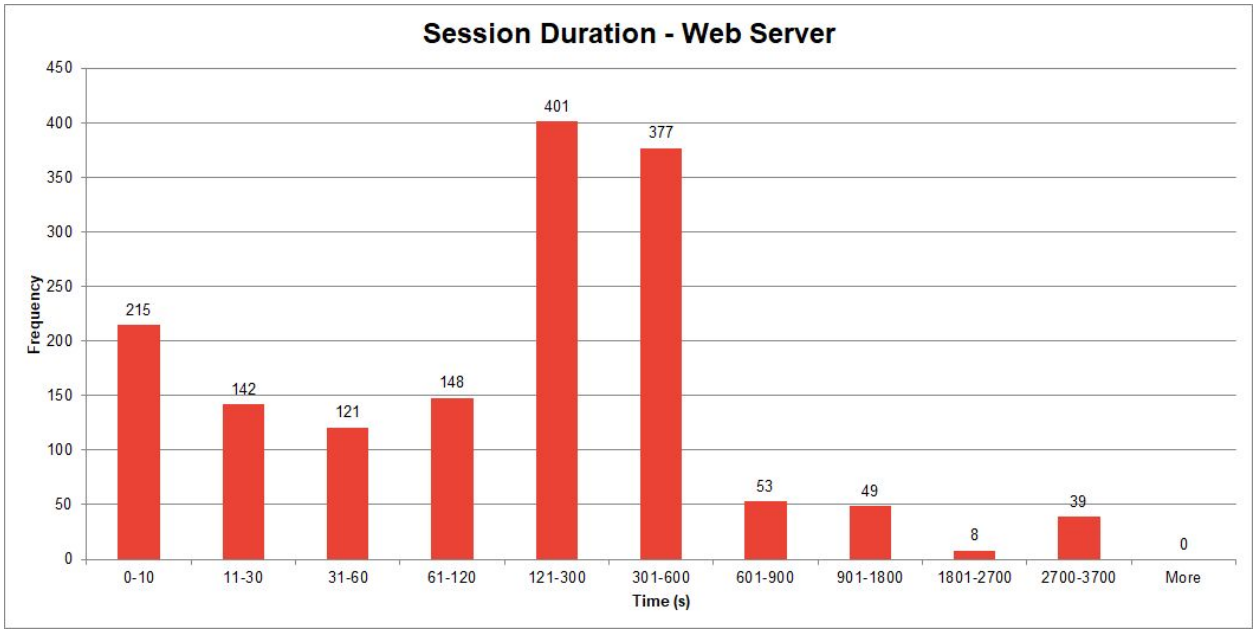
| **Environment** | **Total Number of Attackers** | Returned Expected | | |
|---|---|---|---|---|
| Database | 1339 | 906.2933405 | 432.7066595 | |
| Web Sever | 1553 | 1051.137833 | 501.8621675 | |
| IoT Device | 1389 | 940.1355115 | 448.8644885 | |
| Personal Computer | 1320 | 893.4333155 | 426.5666845 | |
| | | P-value | 3.27142E-05 | |

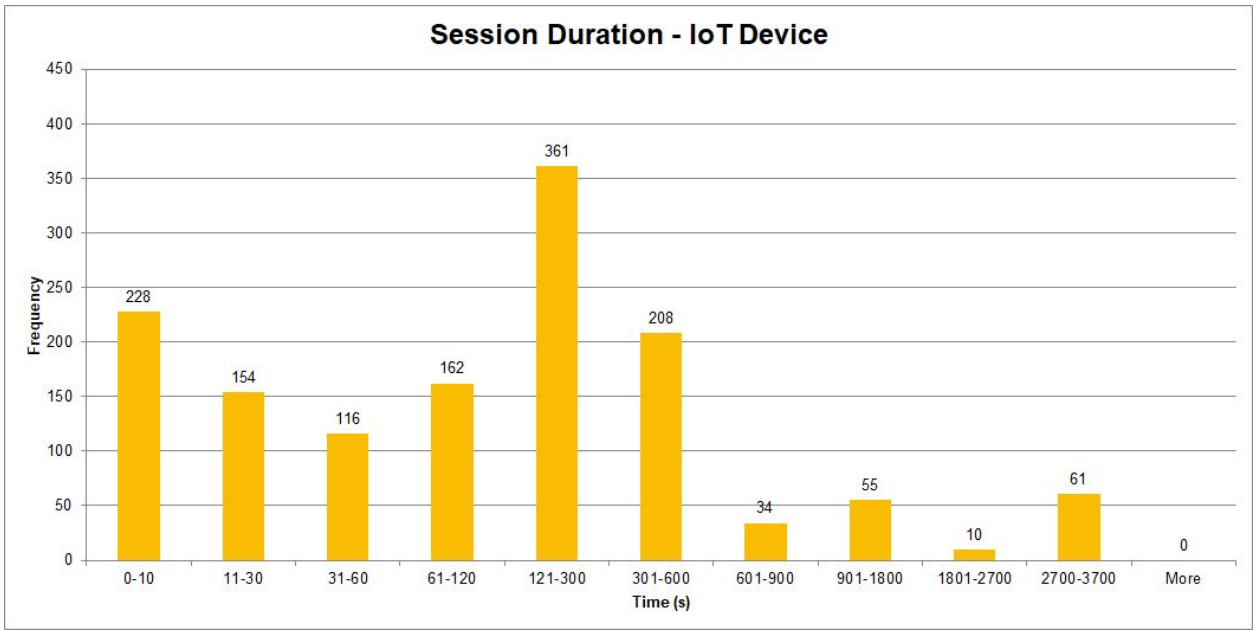
## B.2 Figures

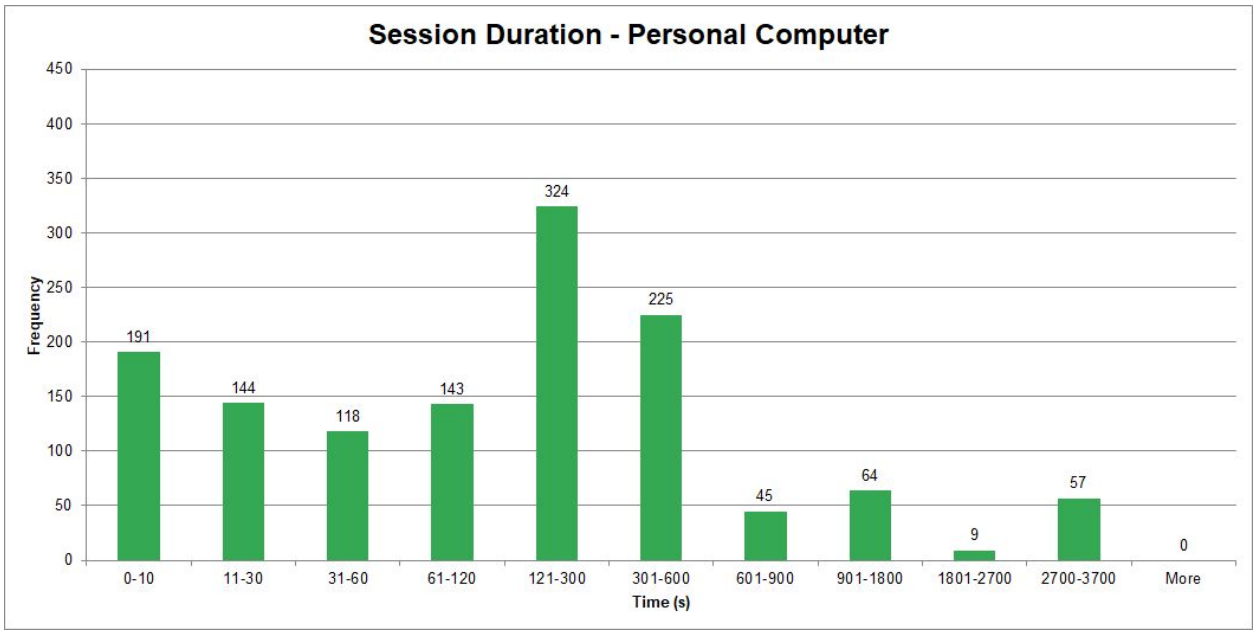

Average Session Duration
450
400
350
300
250
200
150
100
50
0
Time (s)
356
325
362
383
Database
Web Server
IoT Device
PC
Environment

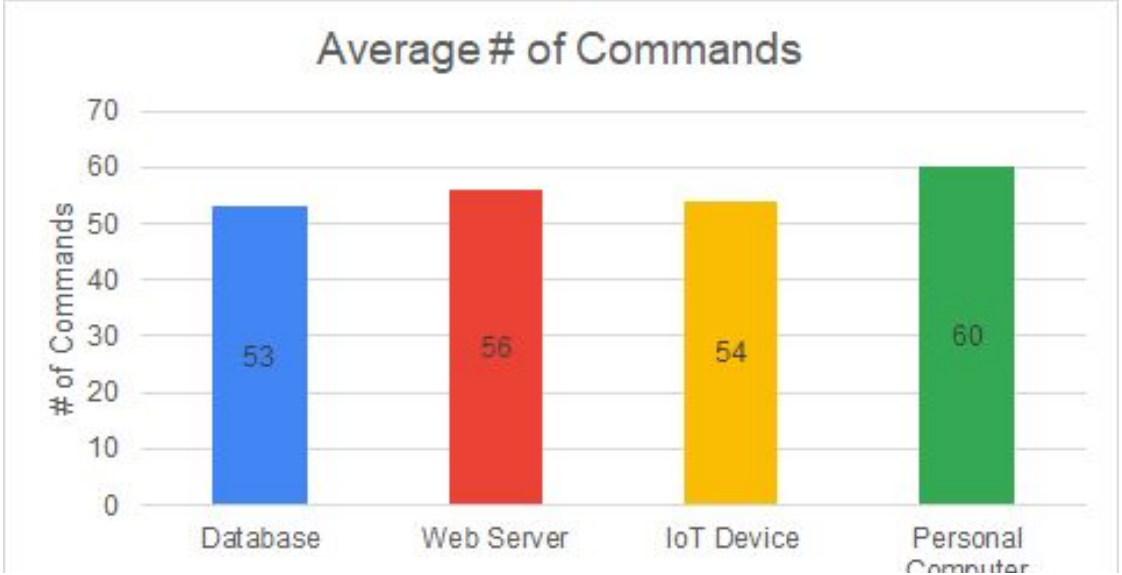

Attack Duration Distribution Per Honeypot (Excluding Outliers)
Database
Web Server
Iot Device
Personal Computer
900
800
700
600
500
400
300
200
100
0
Duration
Honeypots

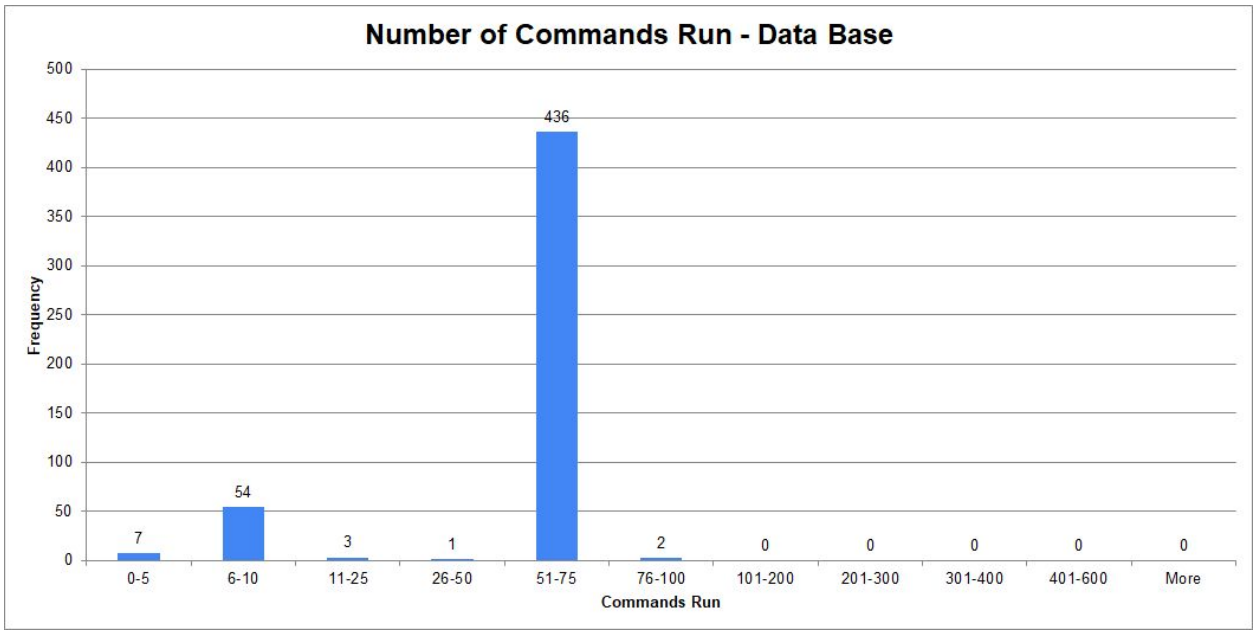

Session Duration - Database
Frequency
Time (s)
0-10
11-30
31-60
61-120
121-300
301-600
601-900
901-1800
1801-2700
2700-3700
More
246
128
91
141
342
229
48
56
8
50

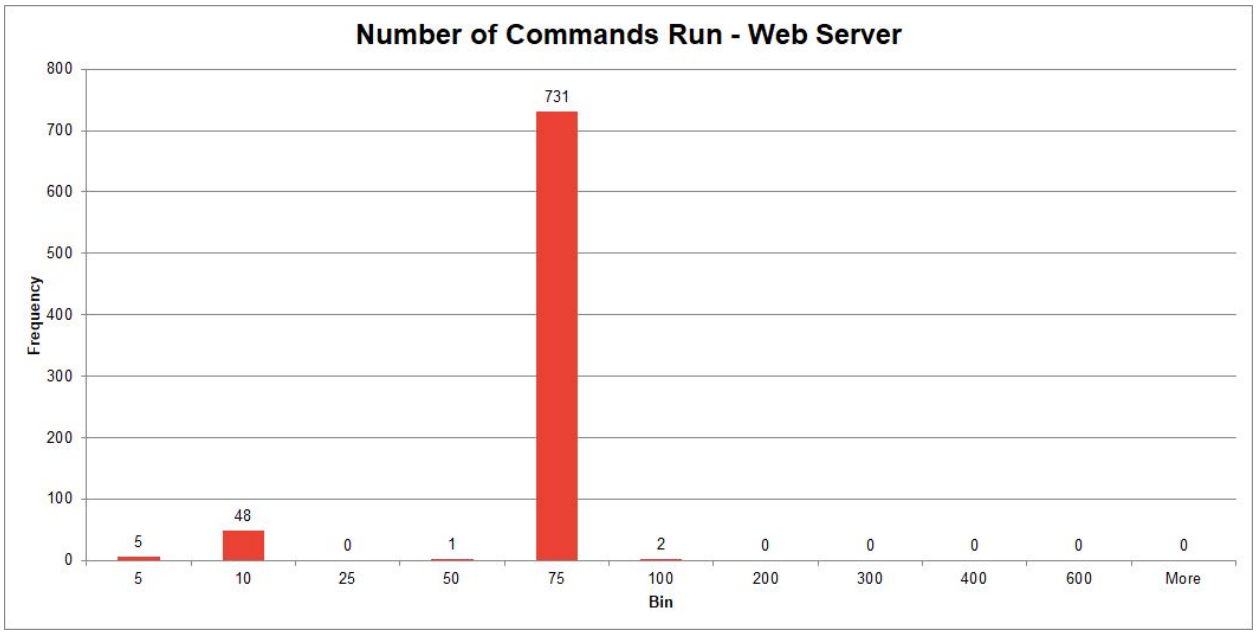

Session Duration - Web Server
Frequency
Time (s)
0-10
11-30
31-60
61-120
121-300
301-600
601-900
901-1800
1801-2700
2700-3700
More
215
142
121
148
401
377
53
49
8
39
0

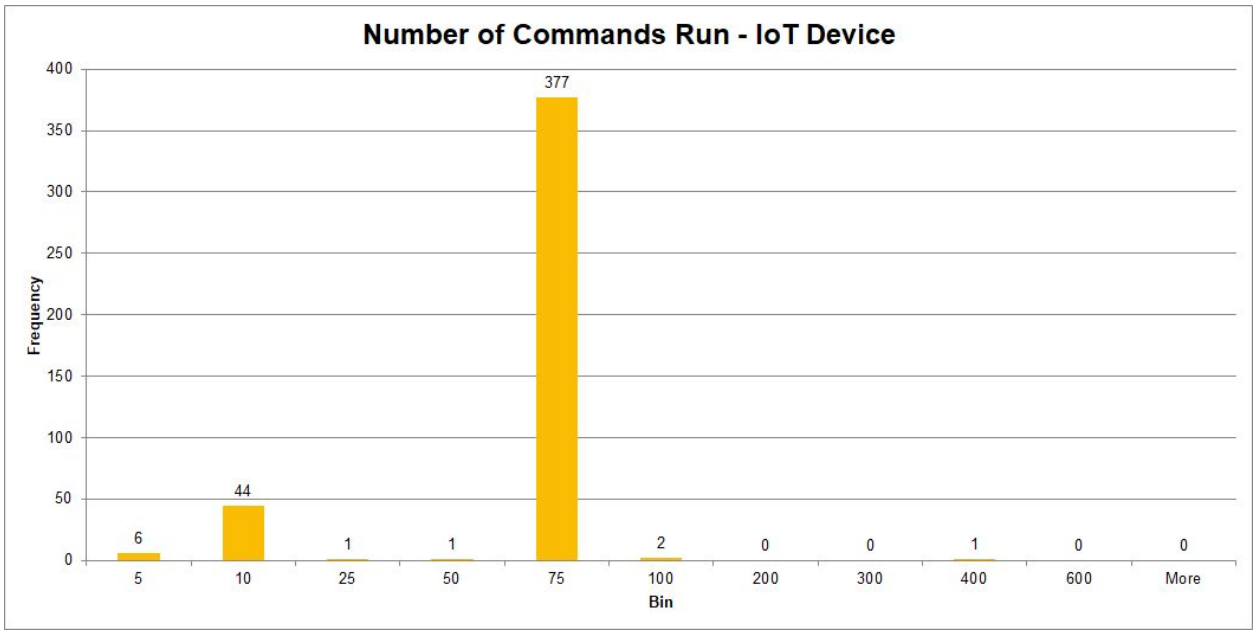

Session Duration - IoT Device
Frequency
Time (s)
0-10
11-30
31-60
61-120
121-300
301-600
601-900
901-1800
1801-2700
2700-3700
More
228
154
116
162
361
208
34
55
10
61
0

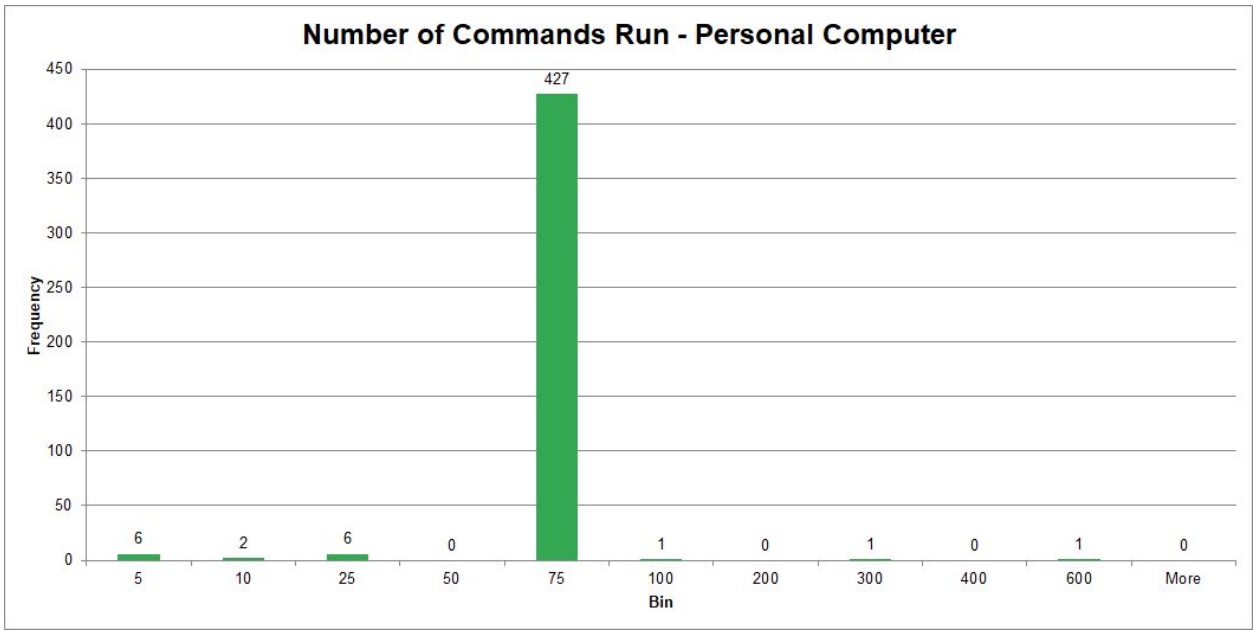

Session Duration - Personal Computer
Frequency
Time (s)
0-10
11-30
31-60
61-120
121-300
301-600
601-900
901-1800
1801-2700
2700-3700
More
191
144
118
143
324
225
45
64
9
57
0

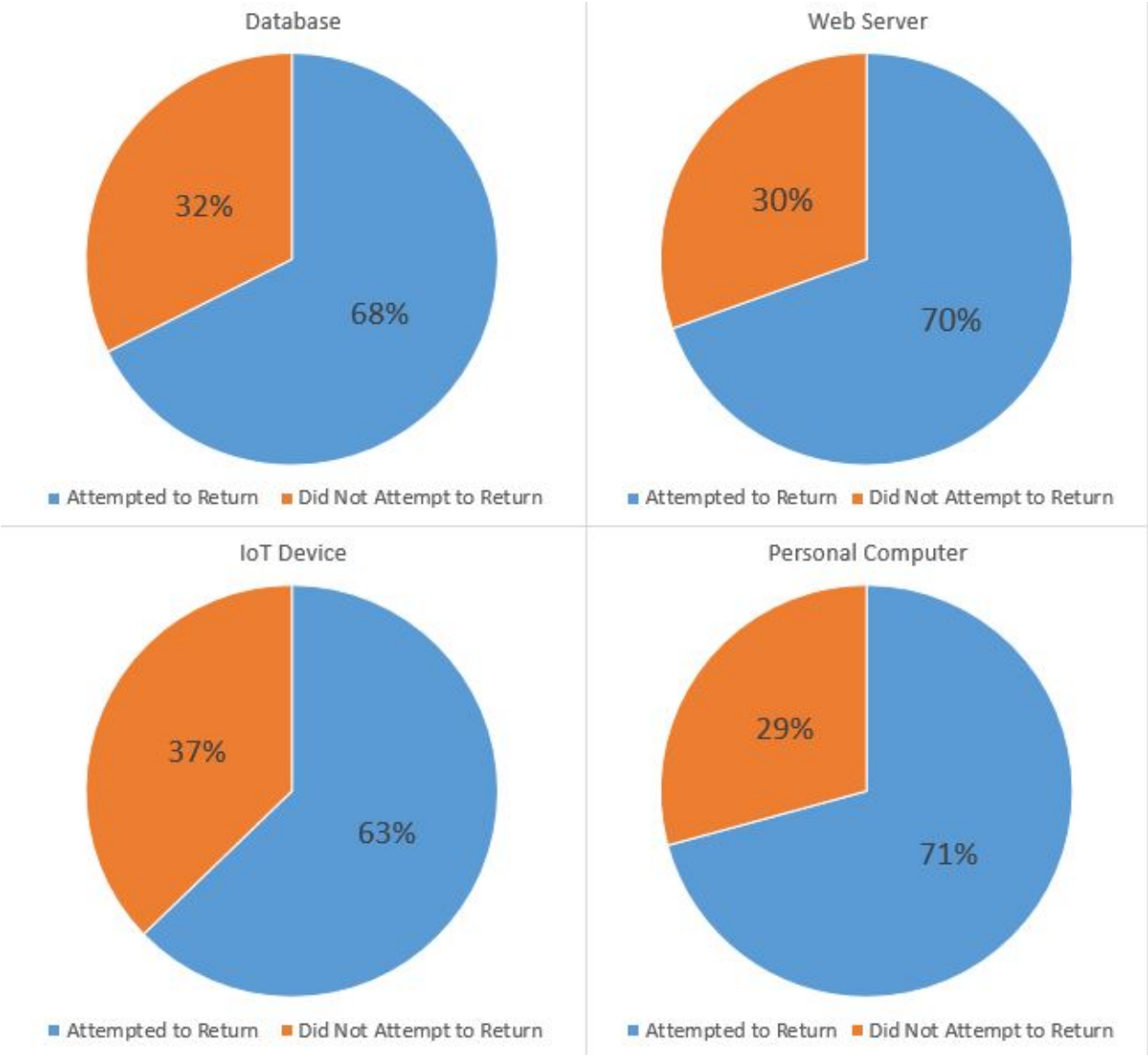

Average # of Commands
# of Commands
Database
Web Server
IoT Device
Personal Computer
53
56
54
60


Number of Commands Run - Data Base
Frequency
500
450
400
350
300
250
200
150
100
50
0
436
54
7
3
1
2
0
0
0
0
0
0-5
6-10
11-25
26-50
51-75
76-100
101-200
201-300
301-400
401-600
More
Commands Run


Number of Commands Run - Web Server
Frequency
800
700
600
500
400
300
200
100
0
731
48
5
0
1
2
0
0
0
0
0
5
10
25
50
75
100
200
300
400
600
More
Bin


Number of Commands Run - IoT Device
Frequency
Bin
5
10
25
50
75
100
200
300
400
600
More
6
44
1
1
377
2
0
0
1
0
0

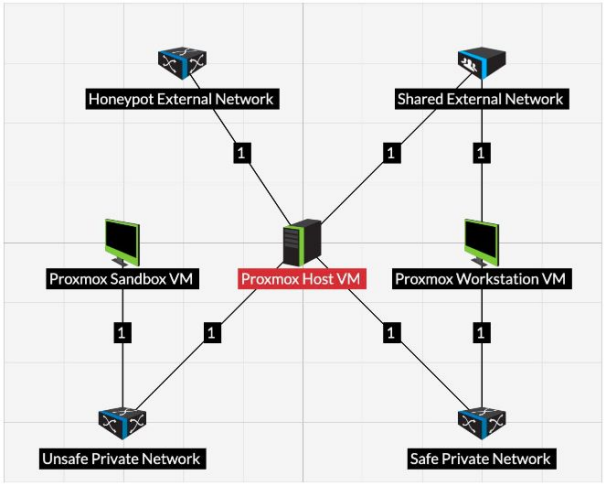

Number of Commands Run - Personal Computer
Frequency
Bin
5
10
25
50
75
100
200
300
400
600
More
6
2
6
0
427
1
0
1
0
1
0




## Appendix C

### C.1 Network Configuration

The topology set up was provided by the HACS200 teaching staff.




## C.2 Scripts

### recycling10X.sh

```
#!/bin/bash

ip=0
sessions=0
pid=0
filename="/root/Honeypot_Project/Attacker_Data/cont101/"
unix_time=0
time_diff=0

tail -F -n 1 /var/lib/lxc/101/rootfs/var/log/auth.log | while read a; do
        status=$(echo $a | awk '{print $6}')
        closed_status=$(echo $a | grep 'pam_unix(sshd:session): session closed' | awk '{print $8}')
        snoopy=$(echo $a | grep 'snoopy')

        time_diff=$(echo `date +%s`)
        time_diff=$((time_diff-unix_time))

        if [[ $unix_time -ne 0 && $time_diff -ge 3600 && $time_diff -le 99999 ]]
        then
                unix_time=0
                echo "KICKED" >> "$filename"
                /bin/date '+%s' >> "$filename"

                sed -i '$d' ~/Honeypot_Project/firewall/firewall_rules.sh
                echo "/sbin/iptables --table filter --insert FORWARD 1 --in-interface enp4s1
--out-interface vmbr0 --source $ip --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump
LOGGING2" >> ~/Honeypot_Project/firewall/firewall_rules.sh
                echo "exit 0" >> ~/Honeypot_Project/firewall/firewall_rules.sh
                /usr/sbin/pct exec 101 -- killall sshd

                iptables --table filter -D FORWARD --in-interface enp4s1 --out-interface vmbr0 --source
$ip --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump ACCEPT
                iptables --table filter -D FORWARD --in-interface enp4s1 --out-interface vmbr0 --source
0.0.0.0/0 --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump DROP
                iptables --table filter --insert FORWARD 1 --in-interface enp4s1 --out-interface vmbr0
--source $ip --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump LOGGING2
```

```
                pid=$(ps -aux | grep "[tail] -F -n 1 /var/lib/lxc/101" | awk '{print $2}' | head -n 1)
                exec ~/Honeypot_Project/restart101.sh $pid &
                exit 0
        fi

        if [[ "$status" = "Accepted" ]]
        then
                sessions=$((sessions+1))
                ip=$(echo $a | awk '{print $11}')
                echo `date` >> "/root/Honeypot_Project/debug.txt"
                echo $ip >> "/root/Honeypot_Project/debug.txt"
                echo $sessions >> "/root/Honeypot_Project/debug.txt"
                if [[ sessions -eq 1 ]]
                then
                        echo "Container 101 compromised by $ip" >>
/root/Honeypot_Project/Alerts/compromises
                        unix_time=$(echo `date +%s`)

                        # create a new data file with the ip address as the name
                        filename="/root/Honeypot_Project/Attacker_Data/cont101/$ip"
                        /usr/bin/touch "$filename"

                        # push the current unix time (on the Proxmox machine) to the file
                        # this marks the start time
                        /bin/date '+%s' >> "$filename"

                        # configure the firewall to only accept traffic on this container from this ip
                        iptables --table filter --insert FORWARD 1 --in-interface enp4s1 --out-interface
vmbr0 --source 0.0.0.0/0 --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump DROP
                        iptables --table filter --insert FORWARD 1 --in-interface enp4s1 --out-interface
vmbr0 --source $ip --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump ACCEPT
                fi
        elif [[ ! -z "$snoopy" && $sessions -ge 1 ]]
        then
                # if the line from auth.log contains "snoopy" then we only send it if it's not automated
                echo $a >> "$filename"

        elif [[ "$closed_status" = "closed" && $sessions -ge 1 ]]
        then
                sessions=$((sessions-1))
                # ip=$(echo $a | awk '{print $8}')
                echo `date` >> "/root/Honeypot_Project/debug.txt"
                echo $ip >> "/root/Honeypot_Project/debug.txt"
                echo $sessions >> "/root/Honeypot_Project/debug.txt"
                if [[ sessions -eq 0 ]]
                then
                        # push the unix time to the data file
                        # this marks the end of the attacker's connection and the completion of the data
file
                        /bin/date '+%s' >> "$filename"

                        sed -i '$d' ~/Honeypot_Project/firewall/firewall_rules.sh
                        echo "/sbin/iptables --table filter --insert FORWARD 1 --in-interface enp4s1
--out-interface vmbr0 --source $ip --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump
LOGGING2" >> ~/Honeypot_Project/firewall/firewall_rules.sh
                        echo "exit 0" >> ~/Honeypot_Project/firewall/firewall_rules.sh
```

```
                        iptables --table filter -D FORWARD --in-interface enp4s1 --out-interface vmbr0
--source $ip --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump ACCEPT
                        iptables --table filter -D FORWARD --in-interface enp4s1 --out-interface vmbr0
--source 0.0.0.0/0 --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump DROP
                        iptables --table filter --insert FORWARD 1 --in-interface enp4s1 --out-interface
vmbr0 --source $ip --destination 172.20.0.2 --protocol tcp --destination-port 22 --jump LOGGING2

                        pid=$(ps -aux | grep "[tail] -F -n 1 /var/lib/lxc/101" | awk '{print $2}' | head
-n 1)
                        exec ~/Honeypot_Project/restart101.sh $pid &
                        exit 0
                fi
        fi
done
exit 0
```

## restart10X.sh

```
#!/bin/bash
kill $1
sleep 5
pct stop 101
pct unmount 101
pct rollback 101 snap1
pct mount 101
pct start 101
exec ~/Honeypot_Project/recycling101.sh

exit 0
```

## health.sh

```
#!/bin/bash

date=$(/bin/date +"%m-%d-%Y %T")

# Get health stats of the host
host_available_mem=$(/usr/bin/free | awk '/Mem:/ {print $7}')
host_available_disk=$(/bin/df | head -n8 | tail -n 7 | awk 'BEGIN {total=0} {total+=$4} END {print
total}')
host_system_load=$(/usr/bin/uptime | cut -d 'v' -f 2 | cut -d ' ' -f 4)

# Get health stats of the UMD Network Interface in KB
host_inc_traffic=$(/sbin/ifconfig enp4s1 | awk '/RX packets/ {print $5}')
host_inc_traffic="$(($host_inc_traffic/1024))"
host_out_traffic=$(/sbin/ifconfig enp4s1 | awk '/TX packets/ {print $5}')
host_out_traffic="$(($host_out_traffic/1024))"

# Get health stats for container 101
c101_available_mem=$(/usr/sbin/pct exec 101 /usr/bin/free | awk '/Mem:/ {print $7}')
c101_available_disk=$(/usr/sbin/pct exec 101 /bin/df | tail -n 7 | awk 'BEGIN {total=0} {total+=$4} END
{print total}')
c101_system_load=$(/usr/sbin/pct exec 101 /usr/bin/uptime | cut -d 'v' -f 2 | cut -d ' ' -f 4)
c101_inc_traffic=$(/usr/sbin/pct exec 101 /sbin/ifconfig eth0 | awk '/RX bytes/ {print $2}' | cut -d ':'
-f 2)
c101_inc_traffic="$(($c101_inc_traffic/1024))"
```

```
c101_out_traffic=$(/usr/sbin/pct exec 101 /sbin/ifconfig eth0 | awk '/TX bytes/ {print $5}' | cut -d ':'
-f 2)
c101_out_traffic="$(($c101_out_traffic/1024))"

# Get health stats for container 102
c102_available_mem=$(/usr/sbin/pct exec 102 /usr/bin/free | awk '/Mem:/ {print $7}')
c102_available_disk=$(/usr/sbin/pct exec 102 /bin/df | tail -n 7 | awk 'BEGIN {total=0} {total+=$4} END
{print total}')
c102_system_load=$(/usr/sbin/pct exec 102 /usr/bin/uptime | cut -d 'v' -f 2 | cut -d ' ' -f 4)
c102_inc_traffic=$(/usr/sbin/pct exec 102 /sbin/ifconfig eth0 | awk '/RX bytes/ {print $2}' | cut -d ':'
-f 2)
c102_inc_traffic="$(($c102_inc_traffic/1024))"
c102_out_traffic=$(/usr/sbin/pct exec 102 /sbin/ifconfig eth0 | awk '/TX bytes/ {print $5}' | cut -d ':'
-f 2)
c102_out_traffic="$(($c102_out_traffic/1024))"

# Get health stats for container 103
c103_available_mem=$(/usr/sbin/pct exec 103 /usr/bin/free | awk '/Mem:/ {print $7}')
c103_available_disk=$(/usr/sbin/pct exec 103 /bin/df | tail -n 7 | awk 'BEGIN {total=0} {total+=$4} END
{print total}')
c103_system_load=$(/usr/sbin/pct exec 103 /usr/bin/uptime | cut -d 'v' -f 2 | cut -d ' ' -f 4)
c103_inc_traffic=$(/usr/sbin/pct exec 103 /sbin/ifconfig eth0 | awk '/RX bytes/ {print $2}' | cut -d ':'
-f 2)
c103_inc_traffic="$(($c103_inc_traffic/1024))"
c103_out_traffic=$(/usr/sbin/pct exec 103 /sbin/ifconfig eth0 | awk '/TX bytes/ {print $5}' | cut -d ':'
-f 2)
c103_out_traffic="$(($c103_out_traffic/1024))"

# Get health stats for container 104
c104_available_mem=$(/usr/sbin/pct exec 104 /usr/bin/free | awk '/Mem:/ {print $7}')
c104_available_disk=$(/usr/sbin/pct exec 104 /bin/df | tail -n 7 | awk 'BEGIN {total=0} {total+=$4} END
{print total}')
c104_system_load=$(/usr/sbin/pct exec 104 /usr/bin/uptime | cut -d 'v' -f 2 | cut -d ' ' -f 4)
c104_inc_traffic=$(/usr/sbin/pct exec 104 /sbin/ifconfig eth0 | awk '/RX bytes/ {print $2}' | cut -d ':'
-f 2)
c104_inc_traffic="$(($c104_inc_traffic/1024))"
c104_out_traffic=$(/usr/sbin/pct exec 104 /sbin/ifconfig eth0 | awk '/TX bytes/ {print $5}' | cut -d ':'
-f 2)
c104_out_traffic="$(($c104_out_traffic/1024))"

# Send data to google sheet
log -k /root/Honeypot_Project/Health_Logs/key -s
https://docs.google.com/spreadsheets/d/1pXi_Qk1LvyuC077TRnM-sJzXs39-wawryjo_0ALNVWE/edit#gid=0 -d
"$date,$host_available_mem,$host_available_disk,$host_system_load,$host_inc_traffic,$host_out_traffic,$c10
1_available_mem,$c101_available_disk,$c101_system_load,$c101_inc_traffic,$c101_out_traffic,$c102_available
_mem,$c102_available_disk,$c102_system_load,$c102_inc_traffic,$c102_out_traffic,$c103_available_mem,$c103_
available_disk,$c103_system_load,$c103_inc_traffic,$c103_out_traffic,$c104_available_mem,$c104_available_d
isk,$c104_system_load,$c104_inc_traffic,$c104_out_traffic"
```

## compromise_monitor.sh

```
#!/bin/bash
echo "" >> /root/Honeypot_Project/Alerts/compromises
tail -f  -n 1 /root/Honeypot_Project/Alerts/compromises | while read a ; do
        if [[ ! -z "$a" ]]
        then
```

```
                log -k /root/Honeypot_Project/Health_Logs/key -s
https://docs.google.com/spreadsheets/d/15X2eUXw4kMgymK6IDudzgCfu3gAZo044OvnIMavHRgk/edit#gid=0 -d "$a" -w
Sheet2
        fi
done

exit 0
```

## /etc/rc.local

```
#!/bin/sh -e

/sbin/iptables --table filter --insert FORWARD --destination 172.20.0.0/16 --source 0.0.0.0/0 --jump DROP

/usr/sbin/pct unmount 101 || true
/usr/sbin/pct unmount 102 || true
/usr/sbin/pct unmount 103 || true
/usr/sbin/pct unmount 104 || true

/root/Honeypot_Project/firewall/firewall_rules.sh || true
/sbin/iptables --table filter --insert FORWARD --destination 172.20.0.0/16 --source 0.0.0.0/0 --jump DROP
/sbin/iptables --table nat --append PREROUTING --destination 128.8.238.99 --jump DNAT --to-destination
172.20.0.2 || true
/sbin/iptables --table nat --append PREROUTING --destination 128.8.238.113 --jump DNAT --to-destination
172.20.0.3 || true
/sbin/iptables --table nat --append PREROUTING --destination 128.8.238.84 --jump DNAT --to-destination
172.20.0.4 || true
/sbin/iptables --table nat --append PREROUTING --destination 128.8.238.126 --jump DNAT --to-destination
172.20.0.5 || true
/sbin/iptables --table nat --append POSTROUTING --source 172.20.0.2 --jump SNAT --to-source 128.8.238.99
|| true
/sbin/iptables --table nat --append POSTROUTING --source 172.20.0.3 --jump SNAT --to-source 128.8.238.113
|| true
/sbin/iptables --table nat --append POSTROUTING --source 172.20.0.4 --jump SNAT --to-source 128.8.238.84
|| true
/sbin/iptables --table nat --append POSTROUTING --source 172.20.0.5 --jump SNAT --to-source 128.8.238.126
|| true

sleep 10

/usr/sbin/pct mount 101 || true
/usr/sbin/pct mount 102 || true
/usr/sbin/pct mount 103 || true
/usr/sbin/pct mount 104 || true

nohup /root/Honeypot_Project/recycling101.sh > /root/recyc101.out 2>&1 &
nohup /root/Honeypot_Project/recycling102.sh > /root/recycl02.out 2>&1 &
nohup /root/Honeypot_Project/recycling103.sh > /root/recyc103.out 2>&1 &
nohup /root/Honeypot_Project/recycling104.sh > /root/recyc104.out 2>&1 &
nohup /root/Honeypot_Project/returning_attackers.sh >> /root/returnattack.out 2>&1 &
nohup /root/Honeypot_Project/Alerts/compromise_monitor.sh >> /root/compmonitor.out 2>&1 &

exit 0
```

## returning_attackers.sh

```
#!/bin/bash
```

```
# first clear the kernel ring buffer
dmesg --clear

# now follow the kernel ring buffer for new logged dropped packets
dmesg -H -w | while read a; do

        cont101=$(echo $a | grep 'CONTAINER 101')
        cont102=$(echo $a | grep 'CONTAINER 102')
        cont103=$(echo $a | grep 'CONTAINER 103')
        cont104=$(echo $a | grep 'CONTAINER 104')

        if [[ ! -z "$cont101" ]]
        then
                echo $a >> /root/Honeypot_Project/Attacker_Data/return_attacks/cont101
        elif [[ ! -z "$cont102" ]]
        then
                echo $a >> /root/Honeypot_Project/Attacker_Data/return_attacks/cont102
        elif [[ ! -z "$cont103" ]]
        then
                echo $a >> /root/Honeypot_Project/Attacker_Data/return_attacks/cont103
        elif [[ ! -z "$cont104" ]]
        then
                echo $a >> /root/Honeypot_Project/Attacker_Data/return_attacks/cont104
        fi

done

exit 0
```

## time_parser.sh

```
#!/bin/bash

count=0

for f in *.txt
do
        # Print ip address
        # echo "$f" | cut -d '.' -f 1,2,3,4

        # Compute session duration
        start_time=$(head -n 1 $f)
        end_time=$(tail -n 1 $f)
        elapsed_time=$((end_time - start_time))
        echo "$elapsed_time"

        count=$((count + 1))
done

echo "Total files processed: $count"

exit 0
```

## command_parser.sh

```
#!/bin/bash
```

```
countCommandsperSess=0

for f in *
do
        countCommandsperSess=`cat $f | grep 'SSH_CONNECTION' | cut -d ':' -f '8' | wc -l`
        echo "$countCommandsperSess"
done

exit 0
```

### cont10X_return_parser.sh

```
#!/bin/bash

# Write all of the IP addresses to a text file
cat cont101.txt | while read line
do
        echo $line | cut -d ' ' -f 8 | cut -d '=' -f 2 >>
        /root/Data_Backups/Return_IP_Lists/cont101_returns.txt
done

exit 0
```

### Personal Computer Desktop and Downloads Script

```
import subprocess
from faker import Faker
fake = Faker()
for x in range(10):
       fakeFile = ' ' + fake.file_name(category=None, extension=None)
         subprocess.call(["touch", fakeFile])
```

### Personal Computer Movies Script

```
import subprocess
from faker import Faker
fake = Faker()
for x in range(37):
   fakeFile = ' ' + fake.word(ext_word_list=None)  + '.mov'
   subprocess.call(["touch", fakeFile])
```

### Personal Computer Music Script

```
import subprocess
from faker import Faker
fake = Faker()
y = 10
for x in range(73):
    fakeFile = ' track' + str(y)  + '.mp3'
    y = y + 1
    subprocess.call(["touch", fakeFile])
```

### Personal Computer Downloads Script

```
import subprocess
from faker import Faker
```

```
fake = Faker()
for x in range():
        fakeFile = ' ' + fake.word(ext_word_list=None) + “.docx”
   subprocess.call(["touch", fakeFile])
```

### Personal Computer Pictures Script

```
import subprocess
from faker import Faker
fake = Faker()
y = 100
for x in range(166):
        fakeFile = ' IMG_0' + str(y) + '.jpg'
        y = y + 1
        subprocess.call(["touch", fakeFile])
```

### WebServer Set-Up Script

```
root@proxmox:~/Honeypot_Project# cat honey102.sh
#!/bin/bash

pct exec 102 -- sudo service apache2 stop
pct exec 102 -- sudo apt-get purge apache2 apache2-utils apache2.2-bin apache2-common
pct exec 102 -- sudo apt-get autoremove
pct exec 102 -- sudo rm -rf /etc/apache2
echo -e "Y" | pct exec 102 -- apt-get remove --purge php* libapache2* apache2*

pct exec 102 -- sudo apt update
echo -e "Y" | pct exec 102 -- sudo apt-get install apache2

cp /root/Honeypot_Project/html/.  /var/lib/lxc/102/rootfs/var/www/html -r
```

### IP Camera Set-Up Script

```
pct exec 103 -- sudo service apache2 stop
pct exec 103 -- sudo apt-get purge apache2 apache2-utils apache2.2-bin apache2-common
pct exec 103 -- sudo apt-get autoremove
pct exec 103 -- sudo rm -rf /etc/apache2
echo -e "Y" | pct exec 103 -- apt-get remove --purge php* libapache2* apache2*

pct exec 103 -- sudo apt update
echo -e "Y" | pct exec 103 -- sudo apt-get install apache2


cp /root/Honeypot_Project/iot/index.html  /var/lib/lxc/103/rootfs/var/www/html
```

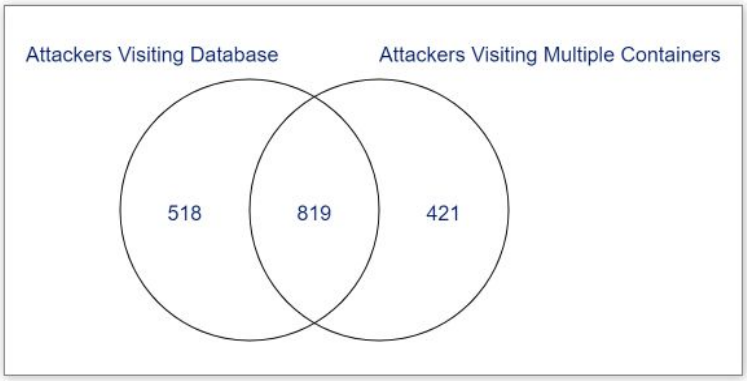
## C.3 Attacker Overlap

Although not incorporated into our analysis of the data, the occurrence of attacker overlap between containers is worth consideration. Attacker overlap refers to an attacker visiting

more than one of our honeypots throughout the course of our experiment. Although an attacker was allowed only one session on a container, they could visit as many as all four honeypots. In total we had 1,240 attackers visit more than one honeypot, and of these attackers, 257 visited all four. Our database honeypot had 518 unique attackers that did not visit any of the other three honeypots. Our web server had 731, our IoT device had 558, and our personal computer had 467. The graphics below depict attacker intersections in more detail.

| | Web Server 1552 | IoT Device 1389 | PC 1318 |
|---|---|---|---|
| Database 1337 | 511 | 526 | 539 |
| Web Server 1552 | | 532 | 536 |
| IoT Device 1389 | | | 537 |

**Pairwise Intersections:** Given any two honeypots, this chart displays how many attackers had a session in both of those honeypots.

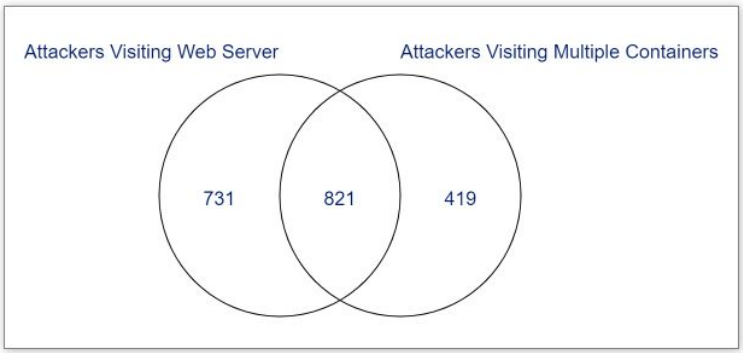


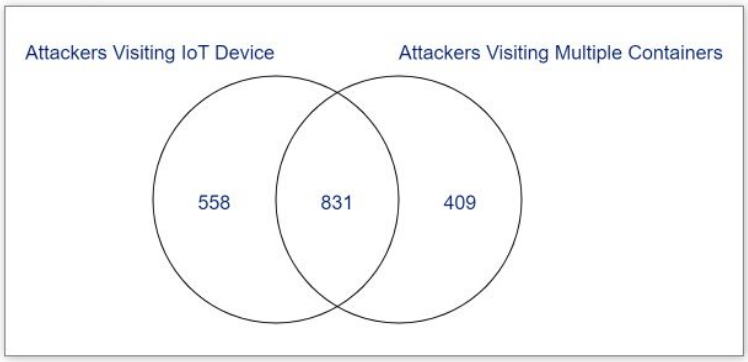

**Database Overlaps:** This venn diagram displays how many unique attackers the database honeypot received.

Out of all of the attackers that visited more than one container, 819 of them visited the database, while 421 did not. Furthermore, the database had 518 unique attackers that did not visit any other container.

**Web Server Overlaps:** This venn diagram displays how many unique attackers the web server honeypot received.

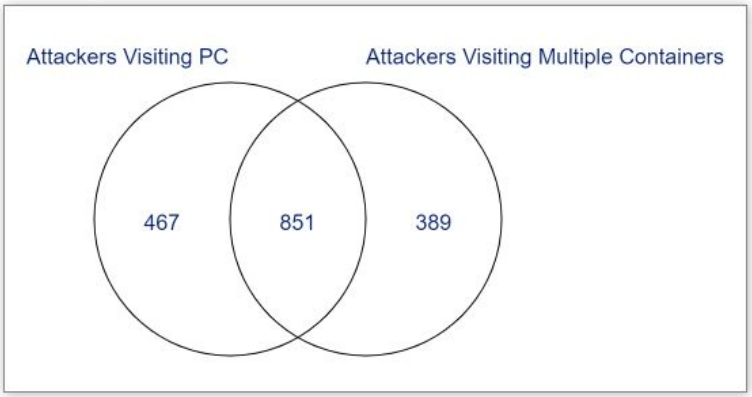
Out of all of the attackers that visited more than one container, 821 of them visited the web server, while 419 did not.




**IoT Device Overlaps:** This venn diagram displays how many unique attackers the IoT honeypot received.

Out of all of the attackers that visited more than one container, 831 of them visited the IoT device, while 409 did not. Furthermore, the database had 558 unique attackers that did not visit any other container.




**Personal Computer Overlaps:** This venn diagram displays how many unique attackers the PC honeypot received.

Out of all of the attackers that visited more than one container, 851 of them visited the personal computer, while 389 did not. Furthermore, the database had 467 unique attackers that did not visit any other container.

## References

Abomhara, M., & Koien, G. M. (2015). Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks. *Journal of Cyber Security and Mobility*, *4*(1), 65–88. doi: 10.13052/jcsm2245-1439.414

Čenys, A., Rainys, D., Radvilavičius, L., & Bielko, A. (2004). Development of Honeypot System Emulating Functions of Database Server. NATO Research and Technology Organisation.

Dowling, S., Schukat, M., & Melvin, H. (2017). A ZigBee Honeypot to Assess IoT Cyberattack Behaviour. *2017 28th Irish Signals and Systems Conference (ISSC),* 1-6. Retrieved from https://ieeexplore.ieee.org/abstract/document/7983603

Hewlett-Packard. (2014, July 29). HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack. Retrieved from https://www8.hp.com/us/en/hp-news/press-release.html?id=1744676

Luo, T., Xu, Z., Jin, X., Jia, Y., & Ouyang, X. (2017). IoTCandyJar : Towards an Intelligent-Interaction Honeypot for IoT Devices. Retrieved from https://paper.seebug.org/papers/Security%20Conf/Blackhat/2017_us/us-17-Luo-Iotcandyjar-Towards-An-Intelligent-Interaction-Honeypot-For-IoT-Devices-wp.pdf

McAfee Labs. (2019, August). Threats Report August 2019. Retrieved from https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf

Minn Pa Pa, Y., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., Rossow, C. (2016). IoTPOT: A Novel Honeypot for Revealing Current IoT Threats. *Journal of Information*

Processing, 23(3), 522–533. Retrieved from https://www.jstage.jst.go.jp/article/ipsjjip/24/3/24_522/_article

O'Gorman, B., Wueest, C., O'Brien, D., Cleary, G., Lau, H., Power, J., ... Wallace, S. (2019, February). ISTR Internet Security Threat Report. Retrieved from https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf

Rahmatullah, S. D. K., Nasution, S. M., Azmi, F. (2016). Implementation of Low Interaction Web Server Honeypot Using Cubieboard. *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, 127-131. Retrieved from https://ieeexplore.ieee.org/abstract/document/7814970

Shulman, A. (2006). Top Ten Database Security Threats: How to Mitigate the Most Significant Database Vulnerabilities , 1–14.

Van Beveren, J. (2001). A Conceptual Model of Hacker Development and Motivations . *Journal of E-Business, 1*(2).

Walsh, R. (2010, July). Folk Models of Home Computer Security. *SOUP's '10 Proceedings on the Sixth Symposium on Usable Privacy and Security*. Retrieved from https://www.rickwash.com/papers/rwash-homesec-soups10-final.pdf

Kirk, R. E. (1988). *Experimental design procedures for the behavioral sciences*. Pacific Grove, CA: Brooks/Cole.

Tiku, M. L. (1971). "Power Function of the F-Test Under Non-Normal Situations". *Journal of the American Statistical Association*. 66 (336): 913–916. doi:10.1080/01621459.1971.10482371.