An Analysis of the Effects of Port Scanning on Attack Durations and Quantity of Commands Run in High-interaction Honeypots

Anthony Naritsin, Kaleb Schmucki, Alexander Smedley, Yichi "Rocky" Zhang

HACS 200 - Group 2C- Section 0201

Executive Summary:

The goal of our project was to explore the effects that disparate quantities of open ports can have on the vulnerability of a machine. Specifically, we addressed two questions: first, does the amount of open ports on a container affect the frequency of attacks it receives, and second, does the amount of open ports on a container affect the type of attacks it receives. For the first research question, we hypothesized that there would be a positive correlation between the number of open ports and the frequency of attacks. For the second research question, we hypothesized that the number of open ports would produce an identifiable difference in type of attacks. In both cases, our final data contradicted our hypotheses, mostly by a means of lack of support for either of them. Over the course of our project, we collected a variety of data including the number of attacks each machine received, the number of ports open during each attack, the start and end time for each attack, the type of attack, and the number of commands run while the attacker was inside the machine. The end result was data that did not support our hypothesis, though not without patterns entirely. The data showed a negative correlation between the number of open ports on the machine and the frequency of non-empty attacks. The goal for the second research question was to specify a ratio of human to bot attacks associated with each number of open ports that we ran on our machines, however, our data included exclusively bot attacks, and because of this, we were unable to identify any pattern of attack type based on the number of open ports on a machine. Overall, our project successfully addressed our research questions with a distinct set of rules for our machines and data collection and showed that at the scale for which we had the resources to test, we could not identify any remarkable patterns to support our hypothesis.

Background Research:

_____This project shows similar themes to research done in an experiment by Mathew, Tabassum, and Lu Ai Siok in 2014. In their experiment, the Swinburne University of Technology professors focused on identifying the various vulnerabilities correlated with open ports on a common user computer. The findings of the Swinburne University experiment study show that a greater number of open ports exhibits a distinct increase in network vulnerability. Logically, it follows that a network with an increased number of open ports becomes more appealing to attackers as a result of greater vulnerability, and often a greater variety of vulnerabilities, as most machines will host an array of different port types.

Another set of research that provides a similar main idea to our own is the work of Lee, which concentrated on the different classifications of port scans that attackers use to probe servers. In her work, Lee classifies three distinct manners of port scanning: horizontal scans, vertical scans, and block scans. A vertical scan is designed to ping multiple ports on a single host, a horizontal scan is used to ping the same port on multiple hosts, and a block scan uses a combination of the two. Similarly to the study done by the Swinburne team, these different port scans display a clear vulnerability in open ports, and the desire for an attacker to target a specific type of port. One important note that Yin addresses in relation to detecting port scans is that it "can provide an early warning and detect some new attacks." This shows the benefits of being aware of the ports you have open and keeping track of who is attempting to access them.

An article from Malwarebytes written by Pieter Arntz, Arntz discusses the increased vulnerability shown to coincide with a greater quantity of open RDP (Remote Desktop Protocol) ports. Arntz begins by defining a set of brute force attacks and their distinct purpose and technique. This list includes reverse brute force attacks, which is when an attacker uses a specific

list of usernames with a set of random passwords; credential stuffing, when an attacker possesses a set of valid usernames and passwords and tests them on different systems; hybrid brute force attack, when an attacker attempts the most likely to succeed combinations - often through the use of a dictionary attack; and finally rainbow table attacks, when an attacker has some semblance of the user password and uses a hash function to find the password in a database. Arntz discusses the specific dangers of RDP ports in relation to brute force attacks and subsequently explains that brute force attacks are simply a numbers game, and the more ports that are open, the more brute force attacks will occur. Because of this, Arntz's biggest recommendations to increase security against brute force attacks are to "limit the number of open ports, restrict the access to those that need it, [and] enhance the security of the port and the protocol." He explains that the COVID-19 quarantine has caused a distinct increase in open RDP ports worldwide, and a marked increase in brute force attacks to go along with it.

From these sources, we are able to draw the conclusion that an increased number of open ports at the very least creates a noticeable increase in system vulnerability, and if that increased vulnerability is identified by the wrong people, could lead to an increase in attacks.

Experiment Design Changes:

From the beginning of this semester, we knew that our research topic was going to focus on how variations of accessible network ports affect intrusion activity. Originally, our plan for collecting data included splitting our honey pot deployment schedule in half, with each half testing a different research question. The first half of the deployment phase would have three honeypots ready to be attacked, where each time they recycled, they would have a random number of ports open. They would have echo servers at each port so as to not have data breaches (which could result from exploitable services). This would allow us to collect data and see if

intrusion activity differed with the number of ports open on a container. The second half of the deployment phase would then have single, specific ports open that would change on each recycle. This would allow us to collect data and see if intrusion activity differed based on which ports were open. Our honey was planned to be fake people's names, email addresses, passwords, and their respective hashes, IP addresses, and a fake EULA as well. We would generate almost all of this from Faker. We would use Man in the Middle (MITM) as our main data collection scheme and then write our own scripts to parse the data from that. We would have a recycling script to automatically recycle our honeypots whenever an attacker left, or after 5 minutes of activity. Additionally, if no one entered the honeypot after 30 minutes, we would recycle then to ensure we would get enough data. In the end of our data analysis, we would use ANOVA testing to draw statistical conclusions from our data. This was the general overview of our experiment before we started performing the setup.

Once we started moving on a plan, we made a couple of changes along the way to our experiment design. These changes did not come until about halfway through the development phase though, because we did not know they would cause issues until we ran into them. The first major change we made was on October 15th, where we started to discuss the scope of our project. We decided to narrow the scope to just focus on examining the effects of having varying numbers of ports open rather than also looking at specific ports. With this change, we essentially are just focusing on two of our original three research questions, disregarding the question, "Which types of ports are susceptible to the highest frequency of attacks?" Additionally, it was at this time we were advised to rotate or randomize our IP addresses to eliminate any IP bias that may occur out of our hands. We did this manually up until November 5th where we changed our plan to automatically eliminate IP bias on recycle instead of a weekly manual plan. Instead of

assigning a random IP address, we assigned a random number of ports open on the container such that each IP address would see an equal number of random accessible ports. In the weeks just before this, on October 24th, we also set up and configured the fourth container to allow for more data collection and spread of independent variables in our experiment whereas previously we were just using three containers. On November 18th, we started discussing using other kinds of statistical tests besides ANOVA since our data was lacking (only for about a week's time period) due to technical issues. We decided that if it came down to it, and if we did not have enough data points, we would run a chi-square test instead of ANOVA. In the end, though, we had more than enough data points and ran Kruskal-Wallis in addition to ANOVA tests because we had enough data to do so and draw even more statistically relevant conclusions. Also, at this time, we had been running into major technical issues that lots of other teams were running into as well, and with how close we were to the end of deployment, we decided that integrating SNORT! would no longer be necessary and would make the research question too broad. Due to these same technical issues, we changed our data parsing scripts to start reading and collecting data directly from the raw man in the middle data instead of the generated tar files, since these kept corrupting. This brought us into Thanksgiving break and the beginning of December, where we ended data collection without any other major changes.

Research Question and Hypothesis:

Through our research, we answer two questions discussing how the amount of ports open on a machine affects cyber attacks. Our first research question is: "Does the number of ports open affect the frequency of attacks on a network?" Our second question is: "Does the number of ports open affect the type of attacks that occur?" We measure the type of attacks by a number of

parameters: whether the attack is human, bot, or empty, the commands run, the number of commands run, and the duration of the attack.

Our hypothesis for the first research question was that a greater number of ports open will increase the frequency of attacks. Attackers often precede an attack with a port scan (Background section). If an attacker detects many open ports, they obtain a wider attack surface and are more likely to target the host for its increased vulnerability. Our second hypothesis was that the number of ports open will change the types of attacks that occur. Specifically, we believe that machines with more ports open will experience longer, varied, and more intense attacks. We also believe that they will experience a greater proportion of human attackers. For one, certain ports host services with specific vulnerabilities that attackers may seek to exploit. Moreover, humans are less likely to attack indiscriminately compared to bots and are thus likely to seek machines with vulnerabilities. In general, attackers are more likely to spend time in a machine if it contains specific or greater vulnerabilities.

For the first research question, our null hypothesis is **H_0: The number of ports open does not affect the frequency of attacks**. For the second research question, our null hypothesis is **H_0: All quantities of open ports have no measurable pattern of attack.**

Experiment Design:

The finalized experiment design, meaning the non-technical parts of our experiment, is as follows. Our goal was to answer these two research questions, “Does the number of ports open affect the frequency of attacks on a network?” and, “Does the number of ports open affect the type of attacks that occur?” In short, we are looking to see if the number of open ports affects intrusion activity. We started with our host machine, which held four containers, called honeypots, that we would use to entice attackers to come into. We had fake data inside each

honeypot and set up monitoring systems to collect all sorts of data, such as the number of commands run, which commands they ran, how long they were inside the container, and more. These containers had scripts in place to eliminate potential bias and ensure the safety of the network that we were running on. After approximately one and a half months of data collection, we would then run ANOVA and Kruskal-Wallis tests on the data between number of commands run, intrusion duration, and the number of ports that were open on the container that was breached. We operationalized all of our dependent variables to ensure we could categorize and qualify each action into meaningful data.

The finalized architecture, meaning how the experiment was set up technically is as follows. Our host VM is the default one that was supplied for us, however, we have configured it so that MITM runs on it. We have on our host VM a /logs directory which holds our error, plain texts, and MITM logs for each of our containers. It runs our four containers whose purpose is listed below. Our containers each have a recycling process that consists of three scripts, wait, monitor, and recycle. Each one calls another in the cycle, so it will continue indefinitely until a member takes them down. We will start from when an attacker finishes their attack. Immediately after this (either a five-minute wait or they disconnect from the session), the old compromised container will be deleted, another one will be cloned from our template and the number of ports on that container was randomly assigned (to eliminate IP bias). Each time a container is initialized, firewall rules will be applied to ensure that the correct ports are open. Then, the recycling script calls the wait script, and it waits until either an attacker intrudes, or 30 minutes is up. If no attacker enters, it recycles the container again back to this point, however, if an attacker does enter, then the wait script is called. The wait script logs all the data mentioned earlier in the paper that we want to collect until the attacker leaves, then it calls the recycle script. The scripts

also block all incoming intruders from entering the container while it is occupied, followed by blocking the intruder when they leave so as to not get attacked by the same IP twice.

For safety measures, we were originally apprehensive about denial of service attacks and others like it, however with our implementation of a rate-limiting to 512 KB/s, we figured this would no longer be an issue. Additional safety measures include the fact that our honeypots are unable to access the Proxmox Workstation VM since it is located on the Safe Private Network, which a visual of can be seen in the appendix (Figure 7.1).

Data Collection:

To answer our research questions we directly collected the start time of each attack, the end time of each attack, the attacker's IP address, the container ID, the number of ports open on the host during that session, and the commands run.

We collected data by running the MITM software on our honeypots. MITM logs various timestamped data, such as connections from IP addresses, key presses, and commands run. We directly parsed the MITM logs with Regex. In particular, we developed a script to read each container's log, identify each unique session via IP, identify starting and ending timestamps, and extract necessary lines--namely commands executed and the number of ports open. Since MITM does not log the number of ports open by default, we injected this value for each session into the log in our wait.sh script. For our final analysis, we utilized data points from November 17th to December 7th. Due to changes in methodology, data collected before November 17th was omitted from our final analysis.

In total, we obtained 1966 attacks. For 0 open ports, we collected 424 bot attacks and 77 empty attacks for a total of 501. For 25 open ports, we obtained 443 bot attacks and 72 empty attacks for a total of 515. For 50 open ports, we obtained 392 bot attacks and 83 bot attacks for a

total of 475. Finally, for 100 ports, we obtained 394 bot attacks and 81 empty attacks for a total of 475. No human attacks were detected within the data collection period.

While reading the MITM logs, our script simultaneously processed the data. For one, the relevant data is cut from the grabbed lines and stored. Data points not directly available from MITM logs, such as duration and the number of commands run, are calculated and stored. To prevent duration from being the entire waiting period, we calculate it as the difference between the time the attacker enters the honeypot and the time the attacker runs their final command. To calculate the number of commands run, we count each component separated by a pipe or semicolon. When counting commands, we do not count options as separate functions. Our parsing script also placed attacks into three categories: human, bot, and empty attacks. Human attacks were identified through interactive commands and the presence of keystrokes, which indicate the presence of someone actually typing on a computer. Bot attacks were identified through a lack of keystrokes, and the presence of instantaneous, non-interactive commands. Finally, empty attacks are classified as just any attack with no command run. As our script stored data, if any information for a specific section was missing, such as a start time, an invalid, dummy value would be placed instead to allow for easy identification of invalid sessions. Once data is fully read it is segmented and inputted row by row into four spreadsheets distinguished by container IP.

To further process the data, we manually copied the four spreadsheets into a single sheet. We then manually removed invalid sessions by identifying cells with invalid values and deleting them. Deleted data points had unknown values, such as nonexistent start times, and were subsequently deleted for our analysis. Then, we sorted data by port number and finally by the

type of attack (bot/empty). This was done to allow the easy omission of empty attacks as their values (0 duration and commands run) could skew results.

Finally, we further processed each non-empty attack to measure its maliciousness. To do so, each attack's commands were put into a bash script on Rocky's computer and were scanned with Windows Defender to determine if they would be detected. Window distinguishes attacks by their severity, the levels being low, medium, high, and severe. A value of 0 was assigned to undetected attacks, 1 was assigned to low threat attacks, 2 was assigned to medium, 3 was assigned to high, and 4 was assigned to severe threats.

Data Analysis:

The first analyses conducted were in relation to the first research question and the relationship between the frequency of attacks and the number of ports. Figure 1.1 illustrates this relationship for non-empty attacks, empty attacks, and attacks ignoring a certain annoying attack (more on that later). The strongest correlation of the three was that between the frequency of non-empty attacks and the number of ports. Recall that non-empty means that no commands were run. This correlation had $R^2 = 0.505$, meaning 50.5% of the variance in the data can be explained by the LSRL, but since this meant $R = -0.71$, it was determined that more work had to be done to show this weak negative correlation was significant. After running a chi-squared test with the result of $p = 0.22$ (Table 1.11), the null hypothesis could not be rejected, so it could not be concluded that there was a significant difference between groups, despite a moderate negative correlation between the number of non-empty attacks and the number of ports. No further statistical tests were run with regards for this research question because the number of attacks is a statistic that could be derived from low-interaction honeypots, and there is no way to analyze the difference between different groups' distributions because whether or not an attack occurred

is a binary, not a measurement. At this point, it should be noted that only 6 of the attacks were flagged by Windows Defender, which is not enough data to do any meaningful analysis. It is hypothesized that this is a combination of Windows Defender being a subpar antivirus software, and the fact that many of the attacks ran benign commands (e.g. proc-cpu).

Analyzing the data for the second research question proved to be tricky since the distributions of the duration and number of commands ended up having enormous spikes the majority of the time. This was a direct result of having the majority of our attacks originating from a single bot that would run the command `cat proc/cpuinfo | grep name | wc -l`. In our case, 1566 of our 1966 attacks ran this exact command. From this point forward, we will refer to those attacks as “proc-cpu” attacks. When analyzing the distribution of the duration of the attacks, it was observed that the first and second most frequent durations were 86396 ms and 86398 ms (Figure 3.1), having 393 attacks and 353 attacks, respectively. After excluding “proc-cpu” attacks from the data, 86396 ms and 86398 ms only have 9 attacks combined (Figure 3.2). For reference, the third most frequent duration in the raw data had 233 attacks, and excluding “proc-cpu” attacks, that duration became the most frequent duration with 76 attacks. Thus, it was suspected that proc-cpu attacks were a culprit for our spikey data.

In order to run ANOVA, certain assumptions need to be made about the sample and population. Unfortunately, just about every assumption needed to run ANOVA, normality, absence of outliers, random/independent samples, and equal variances, were violated by the unfiltered data. When running ANOVA tests, SigmaXL, the Excel plugin used, would generate a report about the extent to how much each assumption was violated.

To test for normality, Anderson-Darling tests were run for each group of ports (0, 25, 50, 100). (Since there are 4 groups per test and 8 ANOVA tests, Anderson-Darling was run 32 times;

more on this later.) Of all 32 times Anderson-Darling was performed, the null hypothesis was rejected each time, leading to the conclusion that none of the data was sampled from a normal distribution (Table 5.21-5.28). Intuitively, this made sense because the attacks seem to be dominated by a select few number of bots. Graphically, it was obvious that the distribution of the raw data for both parameters skewed and even U-shaped (Figure 3.1, 4.3). It is evident from Figure 3.1 that our testing methodology also resulted in a significant spike that contributed to a U-shaped distribution; namely, empty attacks were recorded as having a duration of 0 ms. (Since empty attackers would often stay the full 5 minutes in the container and run zero commands, it was decided that 0 ms was a more appropriate measure of that duration than 5 minutes.) It almost goes without saying that U-shaped distributions are not normal distributions. A similar story can be told regarding the distribution of the number of commands run because empty attacks also made a spike at 0 commands, and all proc-cpu attacks caused a spike at 3 commands.

To make ANOVA even less viable, another complication arising U-shaped distributions is the extremely large variances. At one point, Prof. Cukier even asked us why the variances were so high. When a lot of the data is concentrated at the ends of the distribution, the variance maximized because most of the data is about half the range from the mean. It should be noted, however, that ANOVA also requires equal variances, which was tested for using the aptly named Levene's test of equal variances. Without having to exclude any data, the two Levene's tests run on our raw data did not provide significantly low p-values, meaning it could not be concluded that the variances were unequal. The issue with using ANOVA is the number of outliers in the data. SigmaXL, the software used for running statistical tests, frequently indicated that the number of outliers would be problematic when empty attacks and proc-cpu attacks were included. The software distinguished between possible outliers (3*IQR), likely outliers

(2.2*IQR), and extreme outliers (1.5*IQR). In the unfiltered data, it was found that there was an average of 88.4 outliers per group (Tables 5.21, 5.23). (Four groups per test for 0, 25, 50, and 100 ports.) For these reasons and more, it was decided that ANOVA was not an appropriate statistical test for our use.

To resolve this, the Kruskal-Wallis test was used instead, which is more or less a nonparametric version of ANOVA and believed to be more resilient of a statistical test than ANOVA for this data. Another measure thought to potentially be helpful was analyzing the data without the inclusion of empty attacks, (which are somewhat uninteresting anyway,) and/or without proc-cpu attacks. The goal in excluding that data is to try to limit the number of spikes and outliers enough to potentially make ANOVA more viable. For example, when proc-cpu attacks and empty attacks were excluded, the average number of extreme outliers per group dropped from ~42.9 to ~8.4 (Table 5.21, 5.23, 5.26, 5.28). While it was recognized that either or both of these measures would significantly lessen the sample size, it was agreed upon that it could also make the data more practical because empty attacks and proc-cpu attacks are not necessarily harmful. (It is also unknown how frequent empty/proc-cpu attacks are in the real world. If both types of attacks happened to be uncommon, then excluding them may also make the results more realistic.) As such, analyses were run with regards to the two parameters, duration and number of commands, with and without empty attacks, with and without proc-cpu attacks, and using ANOVA and Kruskal Wallis. In total, we ran 16 statistical tests (Table 5.1).

To easily digest the breakdown of all 16 statistical tests, refer to Table 5.1. The glaring observation is that 7/8 of the tests related to the duration of attacks provided p-values that could reject the null hypothesis. (The only exception to this was the ANOVA test on the raw durations, without the exclusions of empty cpu-proc attacks. We hypothesize that this is a result of

ANOVA's fragility and the fact that the measure of milliseconds leads to the large variances in addition to the other issues of ANOVA mentioned previously.) This allows us to confidently conclude that groups that have different numbers of ports open do experience attacks of different durations. However, the opposite story is true for the 8 tests for the number of commands: 1/8 of the tests related to the number of commands provided p-values that could reject the null hypothesis. Looking more closely at the relationship between the number of ports and the duration of attacks, the graphs were made that broke the data down by the number of ports (Figures 3.1, 3.2). No clear correlation was derived from these two graphs. The last effort to find some correlation between the number of ports and the duration of attacks was graphing the total sum of duration for each number of ports (Figure 3.3). There was a moderate negative correlation with $R^2 = 0.505$ and $R = -0.71$, so a chi-squared test was run to determine if these differences were statistically significant. (Note: we could not use the 7/8 ANOVA or Kruskal-Wallis tests because they were not solely concerned with the sum of all attacks for a particular group.) The chi-squared test resulted in $p = 0.096$, so if $\propto = 0.1$, the null hypothesis could be barely rejected, but for the sake of consistency and having used $\propto = 0.05$ for every other test, it should be concluded that the null hypothesis cannot be rejected. Thus, the moderate negative correlation between number of ports and total duration was concluded to be statistically insignificant. Replicating this graph and the chi-squared test for attacks that are non-empty and not proc-cpu, a statistically significant moderately negative correlation can be found ($R^2 = 0.521$, $R = 0.721$, $p = 0.026$) as per Figure 3.4 and Table 3.4. As a result, it can conclude that there is a moderate negative correlation between the number of ports and the duration of the attacks for non-empty attacks that are not proc-cpu attacks.

As somewhat of a post-script, there are lots of interesting other graphs mostly in Section 1 of Appendix B for one's statistically significant viewing pleasure.

Conclusion:

The most confident conclusion that can be drawn from the data is that honeypots with different numbers of ports open will be subjected to attacks of different durations. This claim can be made confidently because it was verified by 7/8 ANOVA and Kruskal-Wallis tests that tested 4 variations of the dataset each because empty attacks and proc-cpu attacks were included/excluded depending on the variation (Table 5.1). This conclusion was investigated further by graphing the total time (duration) spent in each group (of 0, 25, 50, or 100 ports) in Figure 3.3. While this revealed a moderate negative correlation ($R = -0.71$), a chi-squared test revealed that the difference between groups turned out to be statistically insignificant ($p = 0.096$). (However, since the p-value was relatively close to 0.05, allowing for a more lenient $\propto$ ($= 0.10$), so some may assert that the data points towards a significant negative correlation between the number of ports and duration of attacks.) It turns out that a statistically significant negative correlation can be established between the number of ports and the duration of an attack, but only for non-empty attacks that are not proc-cpu attacks ($R^2 = 0.521$, $R = 0.721$, $p = 0.026$) as per Figure 3.4 and Table 3.4. This conclusion could be applied in cases where one has reason to believe that there will be no empty attackers or proc-cpu attacks, but it is unknown whether the attacks we received were representative of all attacks through SSH. It is suspected that this was a representative sample because the containers were placed on a public network that could be accessed by anyone worldwide. A potential hypothesis for why the correlation would be negative is that it was a result of the testing methodology. It is plausible that it may take some time on the order of a few hundred milliseconds to scan an available machine for ports before

entering it, and in that time, a bot that does not scan ports can enter, leading to attacks with longer (aggregate) durations for honeypots with fewer ports. Verifying this would require revision of the honeypot setup as well as further research. Although configuring Snort! to detect port scans was out of the scope of this project, the inclusion of Snort! would be key in determining which bots are potentially port scanning. Moreover, while it is intuitively obvious that scanning a machine with more ports could take longer, this would have to be empirically verified as well, perhaps with various types of scans, some faster than others. One could also just time how long scanning a variable number of open ports takes.

Appendix A:

Throughout the project, we learned that when producing a complex system with various components, such as a lengthy script with various functions, you must test each component thoroughly before testing the entire system. This project has a lot of moving parts, and there were parts where we had spent a lot of time debugging and not knowing where our problem manifested. Testing each component individually allows you to isolate the problem component, and makes fixing issues easier. Another lesson we learned was using other groups as a resource and collaborating with them. Since we were using a lot of the same software and resources, many groups shared similar issues, such as tar file corruption. By discussing with other groups, we were able to gain ideas and avoid pitfalls. Despite our planning, many things in this project were unexpected and required us to adapt. We soon realized that the questions we sought to answer and the technologies we wanted to use were infeasible. For instance, corruption of tar files forced us to directly parse MITM logs instead. Had we not been prepared to change course, we would not have obtained valid data.

In terms of feedback, we feel that the class could have been paced and organized a little better. Regarding pacing, we spent the first few weeks discussing topics and performing small homework assignments, and then were tasked with writing a proposal worth a huge portion of our grade. While it was on the syllabus, to begin with, we did feel unprepared for the shift in workload. We also feel that lectures could have been ordered better. Early in the semester, we received a lot of valuable information, but it was not relevant at the point we were in the project. If lectures were placed relative to where we were in the project (i.e. a tutorial on setting up MITM while most groups were doing MITM), it would have made the class more helpful. Additionally, we felt that we were not prepared for the number of bot attacks we would experience. Many groups were expecting human attacks, like us, and received none. In general,

though, the class taught us a lot about project management and immersed us within several technical tasks. We appreciate the extensive help of instructors and the understanding of the issues we faced.

We were definitely surprised by the quantity and content of the attacks. For one, we were not expecting the huge prevalence of bot attacks, especially bots executing the same commands consistently. Approximately 75% of our attacks were a single bot, which was strange considering we blocked IPs that had already entered our honeypot. The pattern of attacks was primarily brute force attacks with bots testing every password possible (typically with username and password being the same), then running a single command and leaving. This was definitely distinct from our expectation that attackers would make knowledgeable decisions.

While we had several pitfalls and failures, we will just name a few. For one, we never could figure out how to use Snort to detect port scans. While we attempted to use online resources, the lack of official guidance via the HACS 200 class and our lack of experience meant implementing Snort was infeasible and was thus dropped. Another major pitfall was the corruption of MITM session files despite implementing various fixes. Due to this issue, we had to change our methodology to directly parse MITM logs and this cost us much of the data before we had implemented that solution. Finally, for some reason, MITM would hang connections after a while which we could only fix by rebooting the system. We never realized a fix for this, and if the problem went undetected for a period of time we could lose a considerable amount of data. One additional pitfall was the prevalence of a particular bot which clobbered our results. Since one bot dominated, we had no human attacks and our analysis of various bot behaviors was limited.

Appendix B:

Section 1 (general data that could be collect for a low-interaction honeypot):
Figure 1.1:

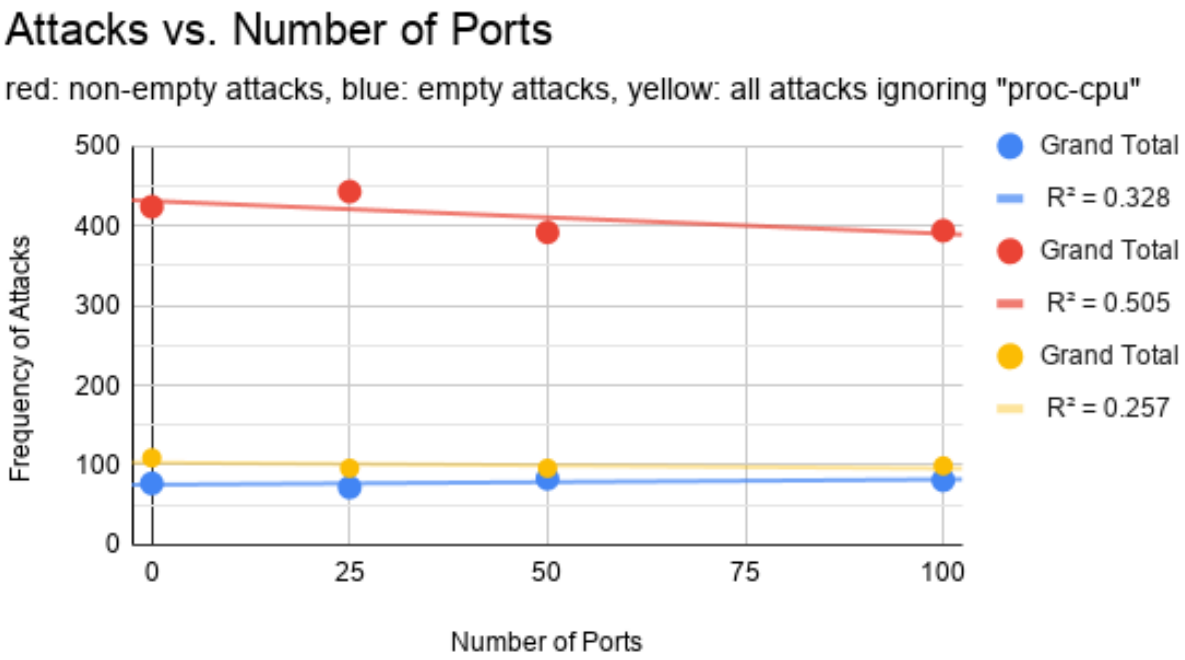


Table 1.11:

	Number of Non-Empty Attacks	
Number of Ports:	Observed:	Expected:
0	424	413.25
25	443	413.25
50	392	413.25
100	394	413.25
Chi-Squared P-Value = 0.22		

Table 1.12:

Non-Empty Attacks	*Number of Ports*			
Container ID	0	25	50	100
101	84	91	68	78
102	116	108	99	109
103	86	88	89	82
104	138	156	136	125

Table 1.13:

Empty Bot Attacks	*Number of Ports*			
Container ID	0	25	50	100
101	6	8	9	8
102	16	20	17	20
103	34	25	31	30
104	21	19	26	23

Table 1.14:

(Non-Empty) Bot Attacks	*Number of Ports*			
Container ID	0	25	50	100
101	6	8	9	8
102	16	20	17	20
103	34	25	31	30
104	21	19	26	23

Figure 1.2:

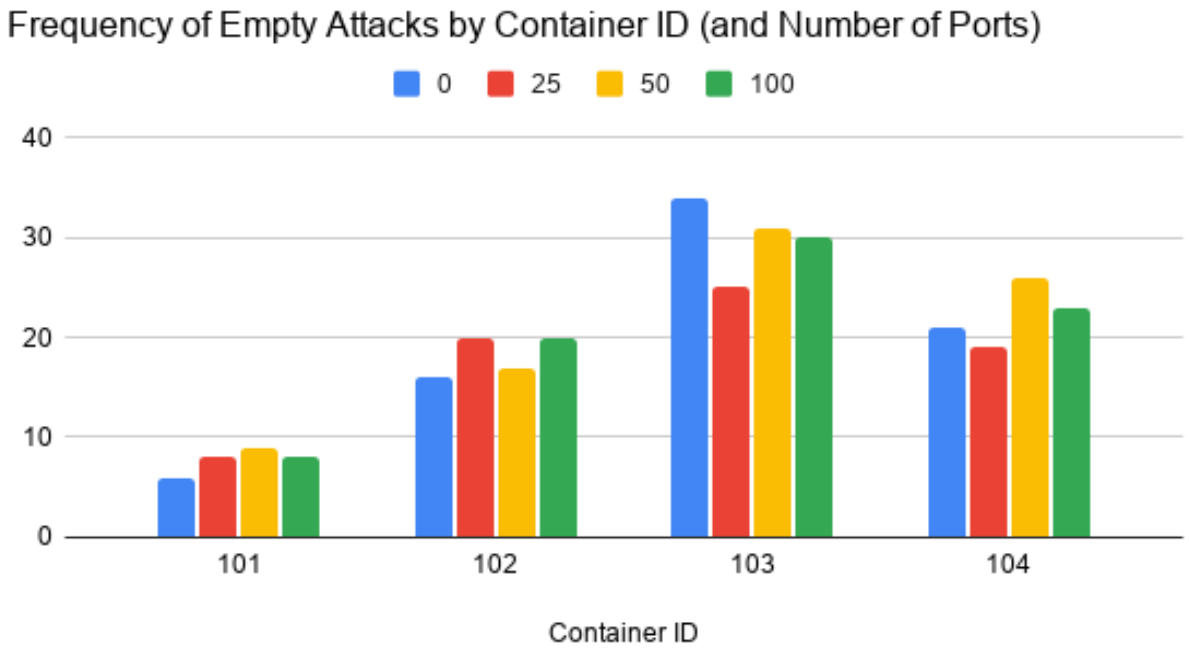
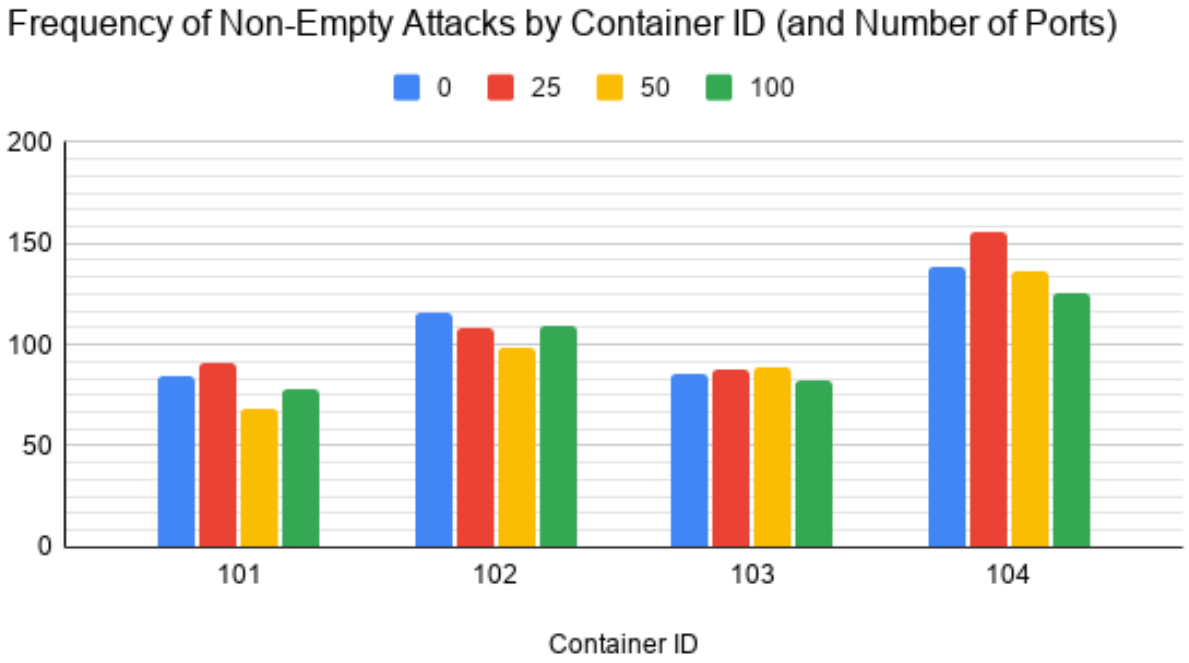


Figure 1.3:



Section 2 (pertaining to the distribution of attacks with respect to time):
Figure 2.1:

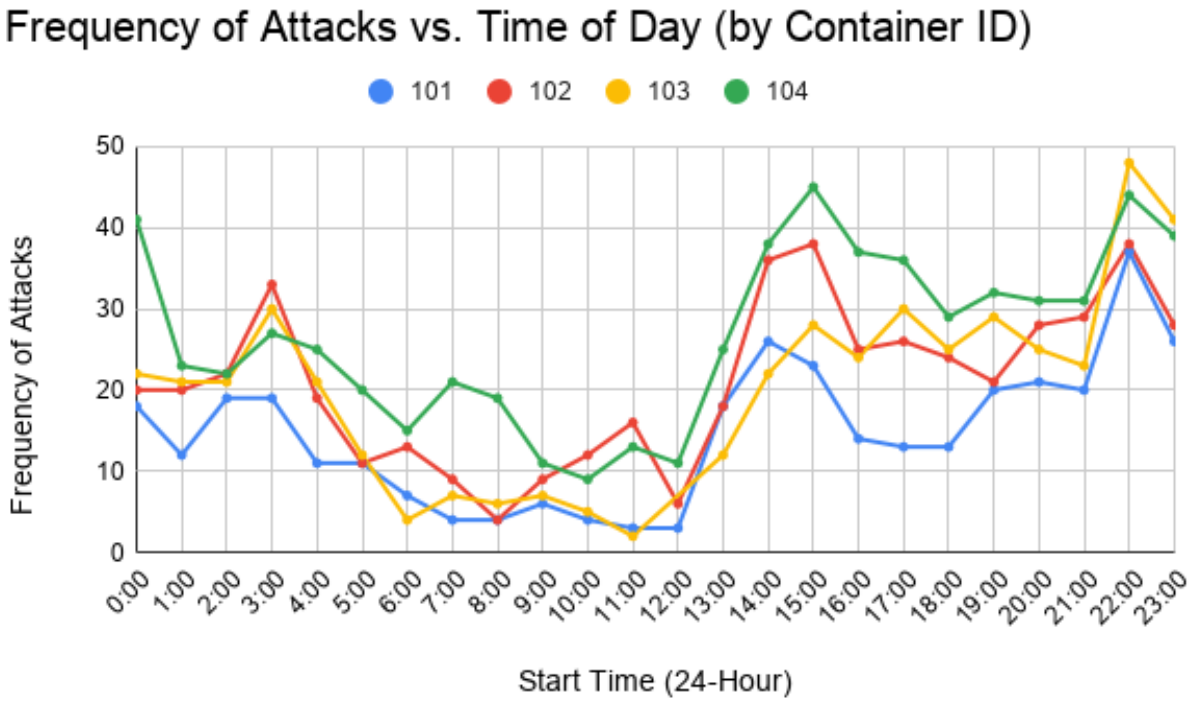


Figure 2.2:

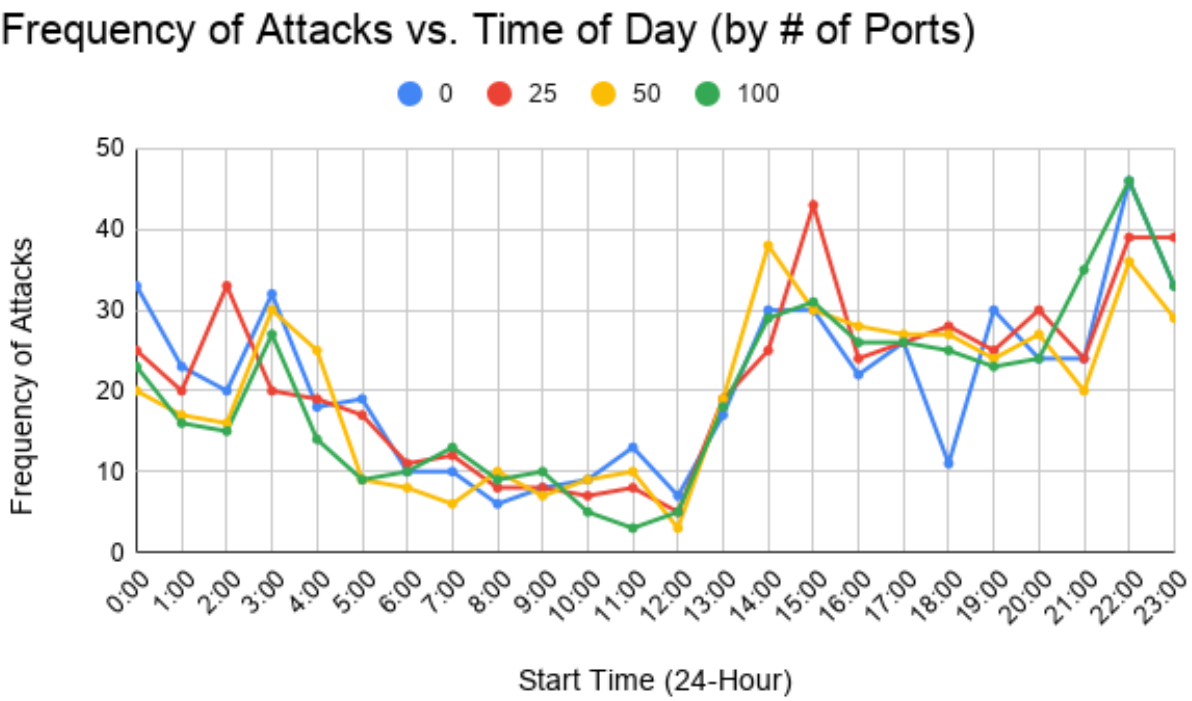


Figure 2.3:

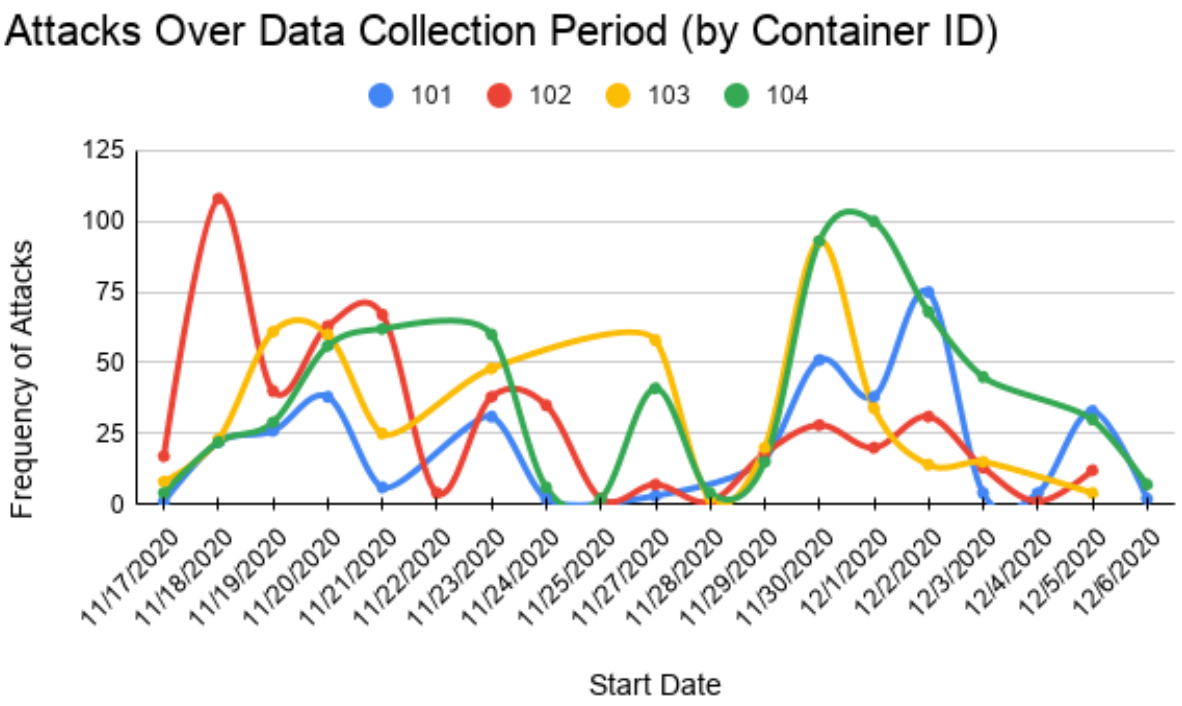
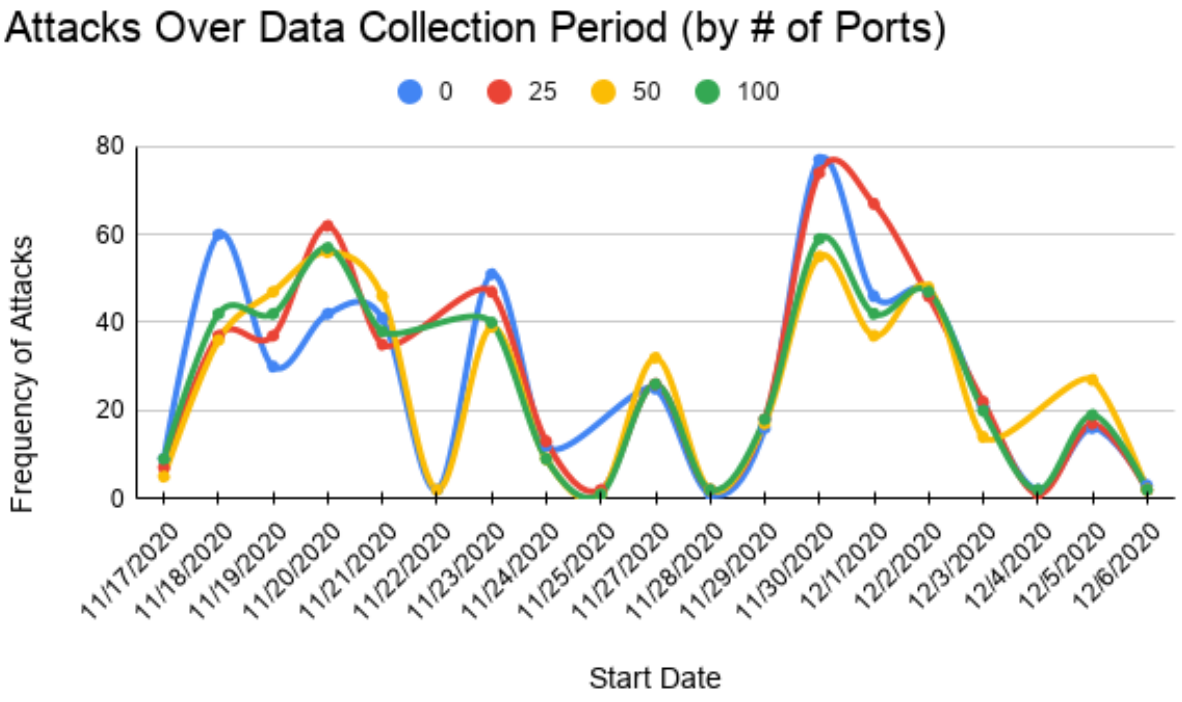


Figure 2.4:



Section 3 (pertaining to the duration of attacks):
Figure 3.1:

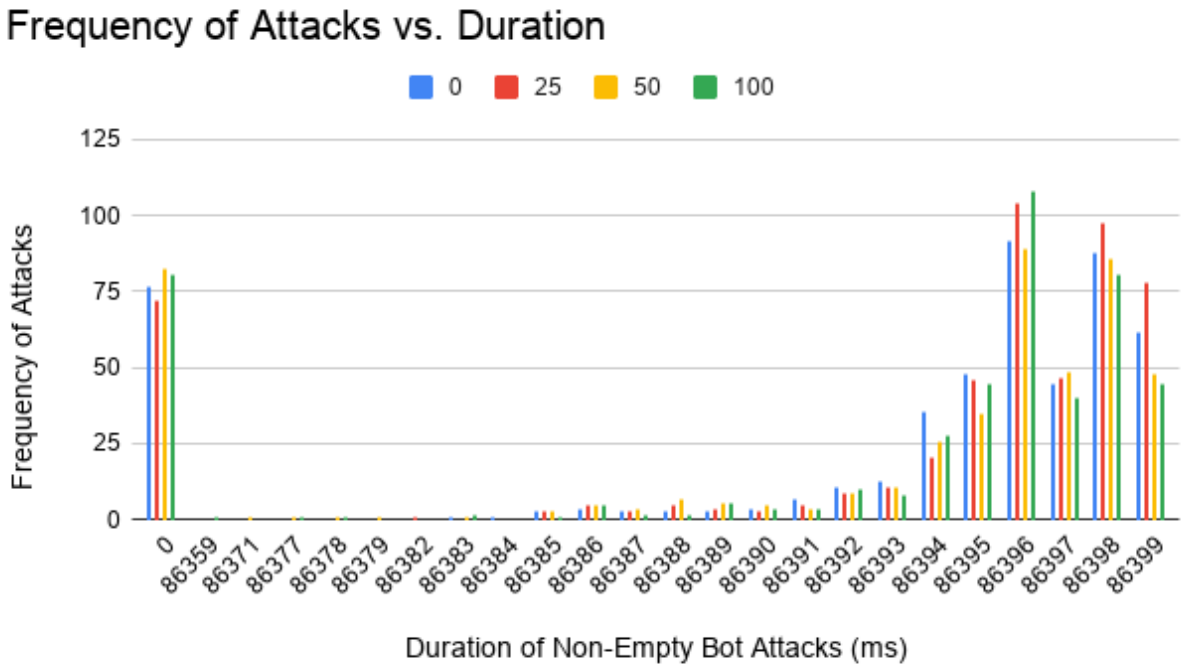


Figure 3.2:

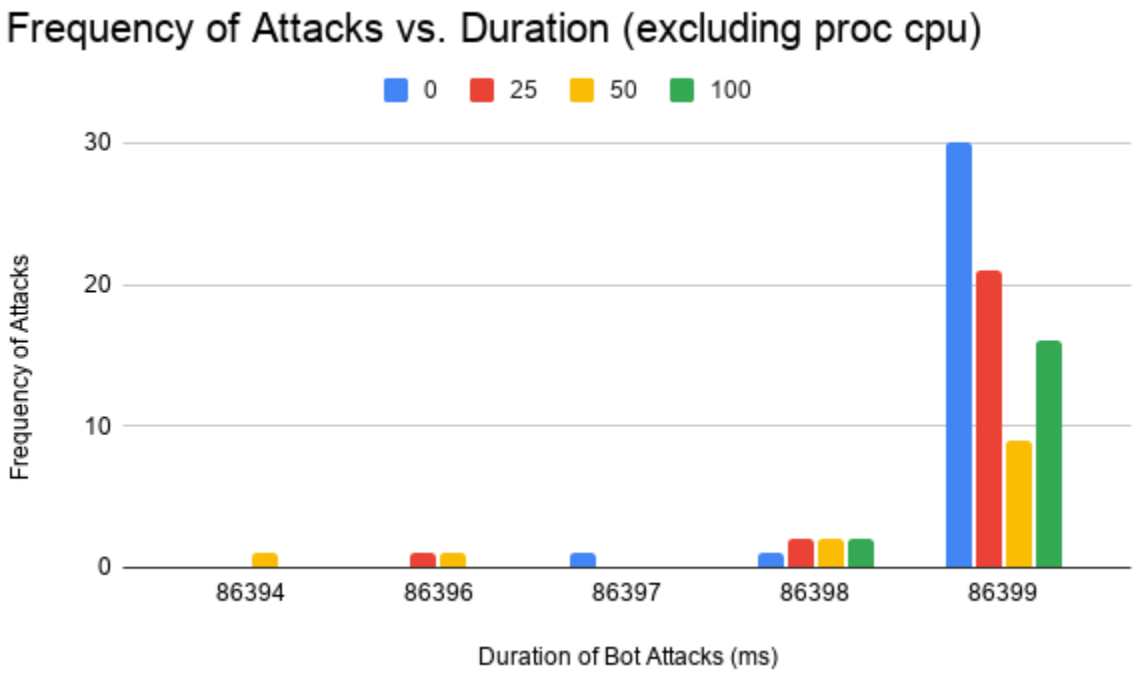


Figure 3.3:

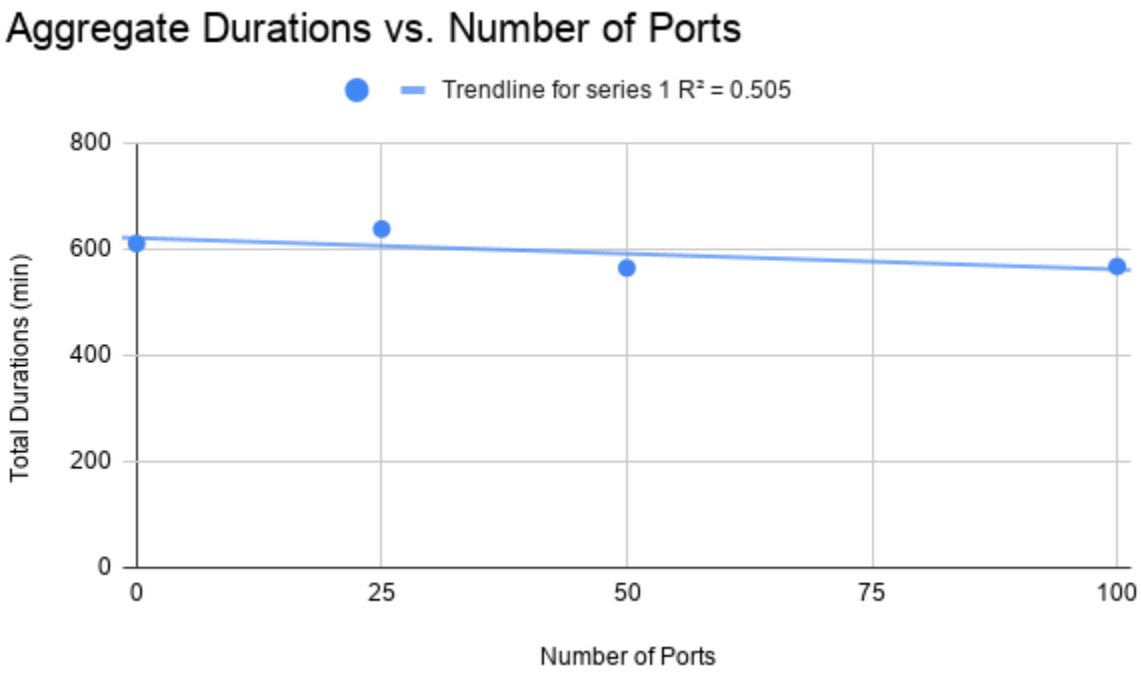


Table 3.3:

	Aggregate Duration (min):	
Number of Ports:	Observed:	Expected:
0	610.53	595.05
25	637.89	595.05
50	564.45	595.05
100	567.33	595.05
Chi-Squared P-Value = 0.096		

Figure 3.4:

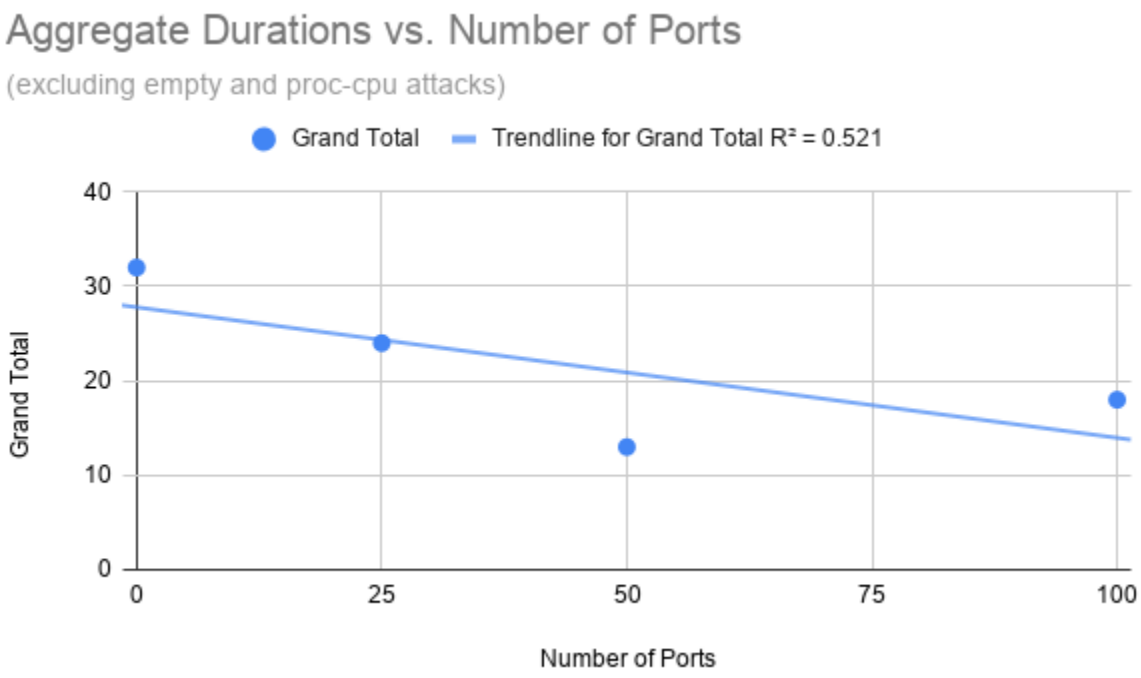


Table 3.4:

	Aggregate Duration (min):	
Number of Ports:	Observed:	Expected:
0	32	21.75
25	24	21.75
50	13	21.75
100	18	21.75
Chi-Squared P-Value = 0.026		

Section 4 (pertaining to the number of commands):
Figure 4.1:

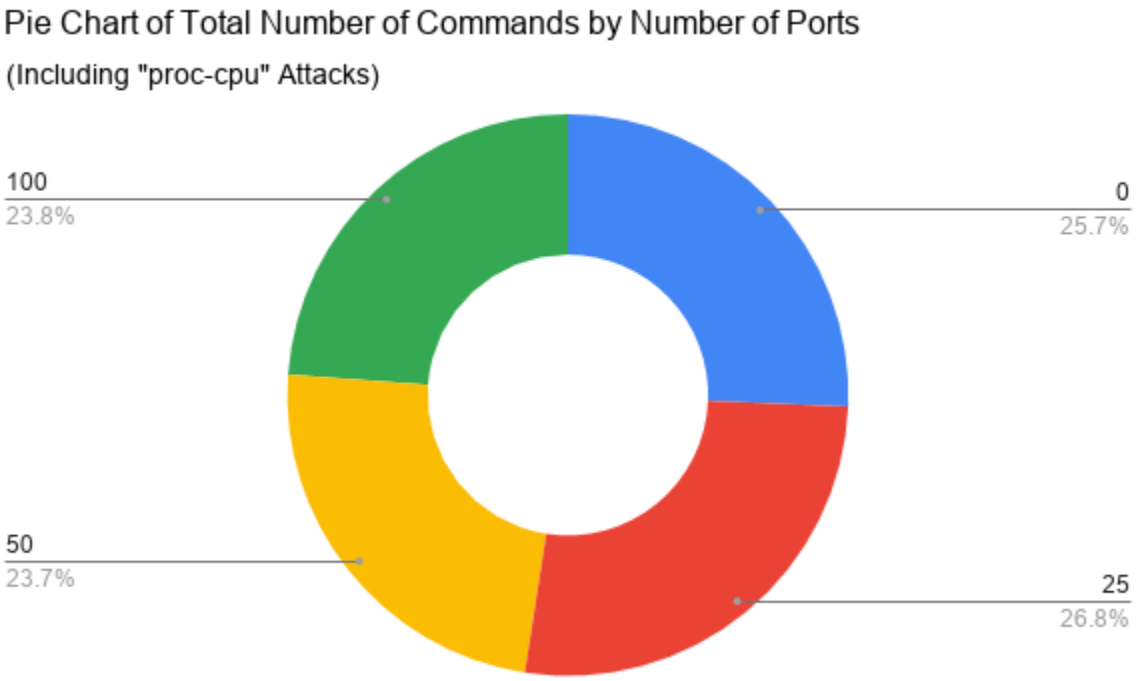


Figure 4.2:

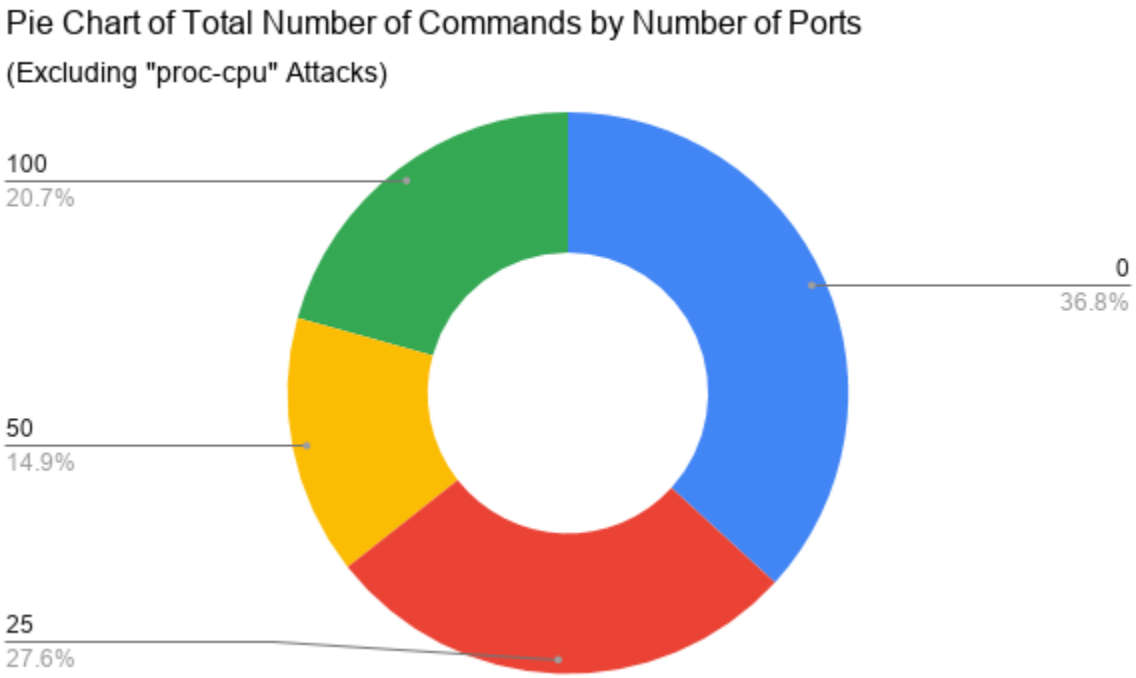


Table 4.1:

Attacks including proc-cpu	
Number of Ports:	*Total Number of Commands:*
0	501
25	515
50	475
100	475
Total	1966

Table 4.2:

Attacks excluding proc-cpu	
Number of Ports:	*Total Number of Commands:*
0	109
25	96
50	96
100	99
Total	400

Figure 4.3:

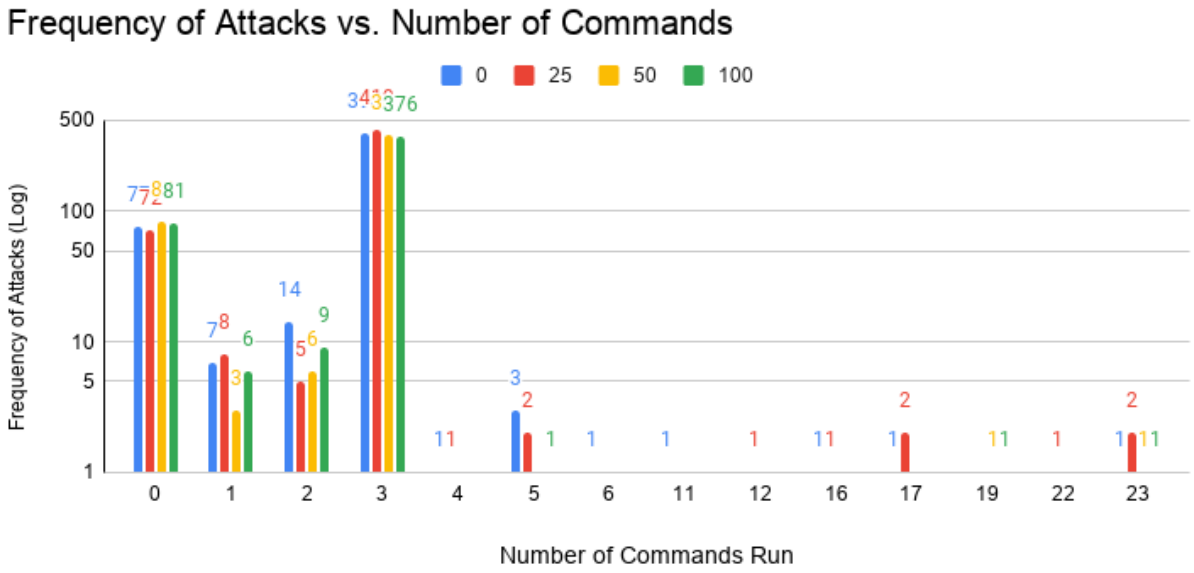


Figure 4.2:

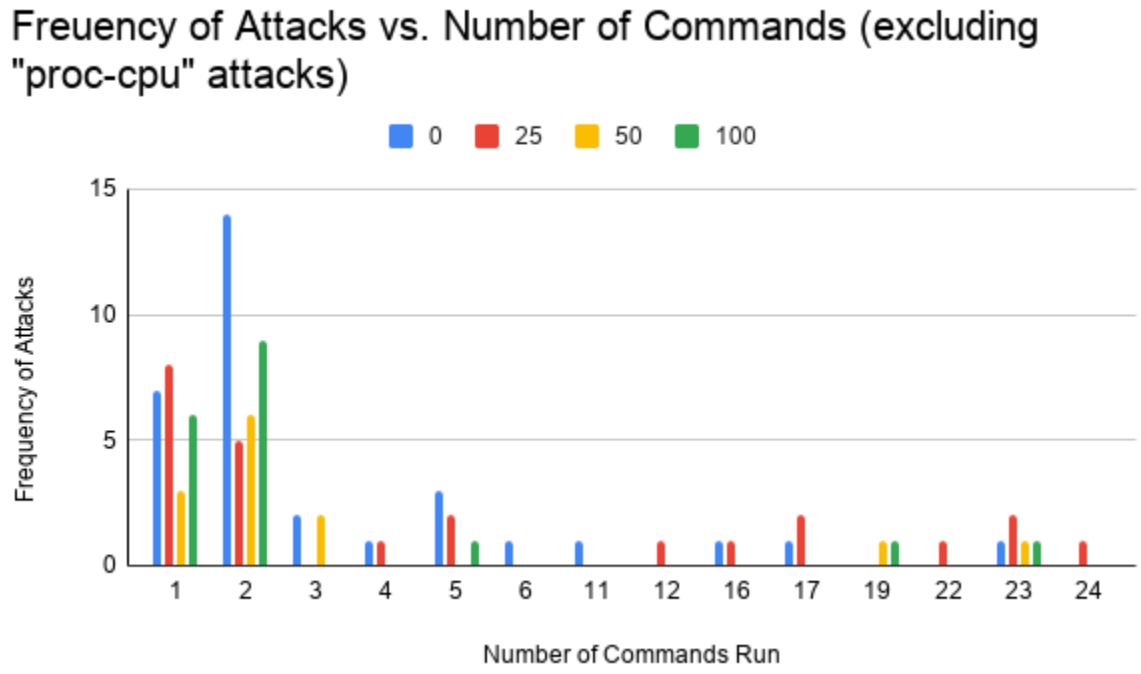
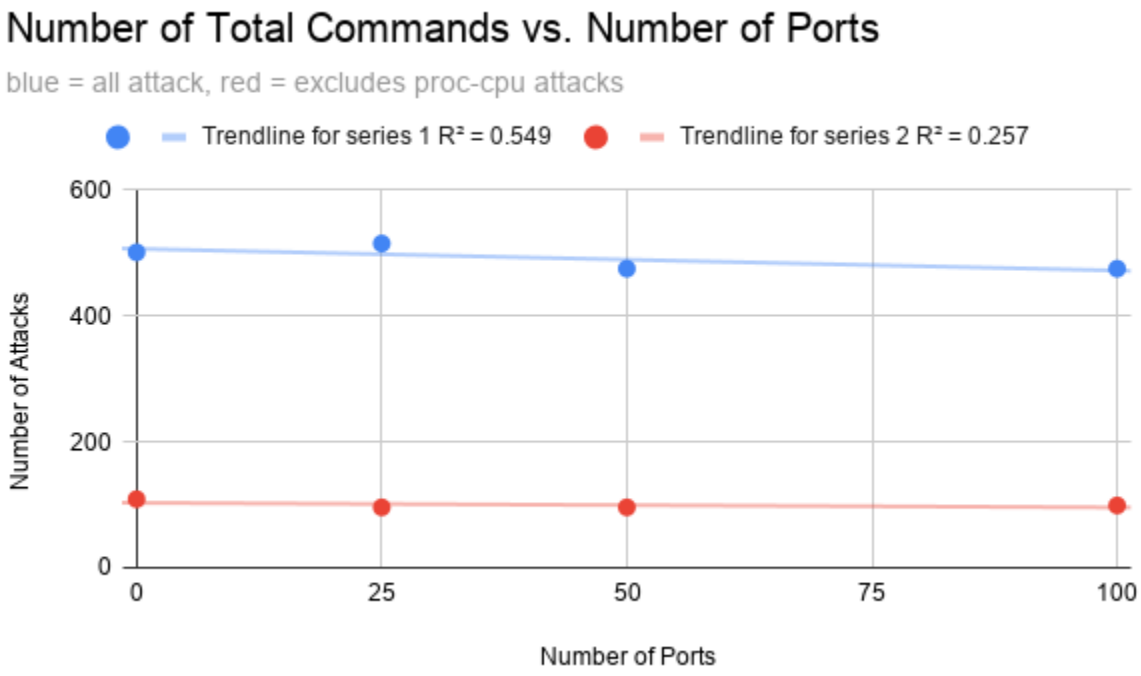


Figure 4.3:


Number of Total Commands vs. Number of Ports
blue = all attack, red = excludes proc-cpu attacks
Trendline for series 1 R² = 0.549
Trendline for series 2 R² = 0.257
600
400
200
0
Number of Attacks
0
25
50
75
100
Number of Ports

Section 5 (statistics tests):

Table 5.1:

Parameter:	Included CPU Proc Atks:	Included Empty Attacks:	ANOVA:	Kruskal-Wallis:
Duration (ms):			p = 0.851	p = 0.046
			p = 0.019	p = 0.031
Number of Commands:			p = 0.053	p = 0.438
			p = 0.048	p = 0.195
Duration (ms):			p = 0.031	p = 0.022
			p = 0.041	p = 0.027
Number of Commands:			p = 0.078	p = 0.218
			p = 0.204	p = 0.619

Table 5.21:

Duration (including empty attacks): One-Way ANOVA Assumptions Report				
Normality (Anderson Darling P-Value): H_0 = data is sampled from a normal distribution	p = 0.000	p = 0.000	p = 0.000	p = 0.000
Robustness:	Minimum sample size for a robust ANOVA test = 12. Each sample size is greater than this.			
Outlier Counts: Extreme (3*IQR), Likely (2.2*IQR), Potential (1.5*IQR)	Extreme = 77 Potential = 5	Extreme = 72 Likely = 4 Potential = 8	Extreme = 83 Likely = 1 Potential = 3	Extreme = 82 Likely = 2
Randomness (Non-parametric Runs Test P-Value): H_0 = data is random	p = 0.000	p = 0.032	p = 0.160	p = 0.287
Equal Variance (Levene's test for Equal Variances P-Value): H_0 = variances are equal	p = 0.412			

Table 5.22:

Duration (excluding empty attacks): One-Way ANOVA Assumptions Report				
Normality (Anderson Darling P-Value): H_0 = data is sampled from a normal distribution	p = 0.000	p = 0.000	p = 0.000	p = 0.000
Robustness:	Minimum sample size for a robust ANOVA test = 27. Each sample size is greater than this.			
Outlier Counts: Extreme (3*IQR), Likely (2.2*IQR), Potential (1.5*IQR)	Extreme = 5 Likely = 10 Potential = 7	Extreme = 4 Likely = 13 Potential = 7	Extreme = 6 Likely = 9 Potential = 10	Extreme = 6 Likely = 9 Potential = 10
Randomness (Non-parametric Runs Test P-Value): H_0 = data is random	p = 0.000	p = 0.000	p =0.000	p = 0.000
Equal Variance (Levene's test for Equal Variances P-Value): H_0 = variances are equal	p = 0.322			

Table 5.23:

Number of Commands (including empty attacks): One-Way ANOVA Assumptions Report				
Normality (Anderson Darling P-Value): H_0 = data is sampled from a normal distribution	p = 0.000	p = 0.000	p = 0.000	p = 0.000
Robustness:	Minimum sample size for a robust ANOVA test = 74. Each sample size is greater than this.			
Outlier Counts: Extreme (3*IQR), Likely (2.2*IQR), Potential (1.5*IQR)	Extreme = 107	Extreme = 96	Extreme = 94	Extreme = 96
Randomness (Non-parametric Runs Test P-Value): H_0 = data is random	p = 1.000	p = 1.000	p = 1.000	p = 1.000
Equal Variance (Levene's test for Equal Variances P-Value): H_0 = variances are equal	p = 0.829			

Table 5.24:

Number of Commands (excluding empty attacks): One-Way ANOVA Assumptions Report				
Normality (Anderson Darling P-Value): H_0 = data is sampled from a normal distribution	p = 0.000	p = 0.000	p = 0.000	p = 0.000
Robustness:	Minimum sample size for a robust ANOVA test = 50. Each sample size is not greater than this.			
Outlier Counts: Extreme (3*IQR), Likely (2.2*IQR), Potential (1.5*IQR)	Extreme = 30	Extreme = 24	Extreme = 11	Extreme = 18
Randomness (Non-parametric Runs Test P-Value): H_0 = data is random	p = 0.017	p = 0.019	p =0.005	p = 0.007
Equal Variance (Levene's test for Equal Variances P-Value): H_0 = variances are equal	p = 0.176			

Table 5.25:

Duration (excluding proc-cpu attacks & including empty attacks): One-Way ANOVA Assumptions Report				
Normality (Anderson Darling P-Value): H_0 = data is sampled from a normal distribution	p = 0.000	p = 0.000	p = 0.000	p = 0.000
Robustness:	Minimum sample size for a robust ANOVA test = 13. Each sample size is greater than this.			
Outlier Counts: Extreme (3*IQR), Likely (2.2*IQR), Potential (1.5*IQR)	None	None	Extreme = 13	Extreme = 18
Randomness (Non-parametric Runs Test P-Value): H_0 = data is random	p = 0.000	p = 0.000	p = 0.021	p = -0.004
Equal Variance (Levene's test for Equal Variances P-Value): H_0 = variances are equal	p = 0.031			

Table 5.26:

Duration (excluding proc-cpu attacks & empty attacks): One-Way ANOVA Assumptions Report				
Normality (Anderson Darling P-Value): H_0 = data is sampled from a normal distribution	p = 0.000	p = 0.000	p = 0.000	p = 0.000
Robustness:	Minimum sample size for a robust ANOVA test = 52. Each sample size is not greater than this.			
Outlier Counts: Extreme (3*IQR), Likely (2.2*IQR), Potential (1.5*IQR)	Extreme = 2	Extreme = 3	Extreme = 1	Extreme = 2
Randomness (Non-parametric Runs Test P-Value): H_0 = data is random	p = 0.123	p = 0.812	p =0.726	p = 0.020
Equal Variance (Levene's test for Equal Variances P-Value): H_0 = variances are equal	p = 0.041			

Table 5.27:

Number of Commands (excluding proc-cpu attacks & including empty attacks): One-Way ANOVA Assumptions				
Normality (Anderson Darling P-Value): H_0 = data is sampled from a normal distribution	p = 0.000	p = 0.000	p = 0.000	p = 0.000
Robustness:	Minimum sample size for a robust ANOVA test = 112. Each sample size is greater than this.			
Outlier Counts: Extreme (3*IQR), Likely (2.2*IQR), Potential (1.5*IQR)	Extreme = 3	Extreme = 16	Extreme = 13	Extreme = 18
Randomness (Non-parametric Runs Test P-Value): H_0 = data is random	p = -0.000	p = 0.000	p = -0.021	p = -0.004
Equal Variance (Levene's test for Equal Variances P-Value): H_0 = variances are equal	p = 0.078			

Table 5.28:

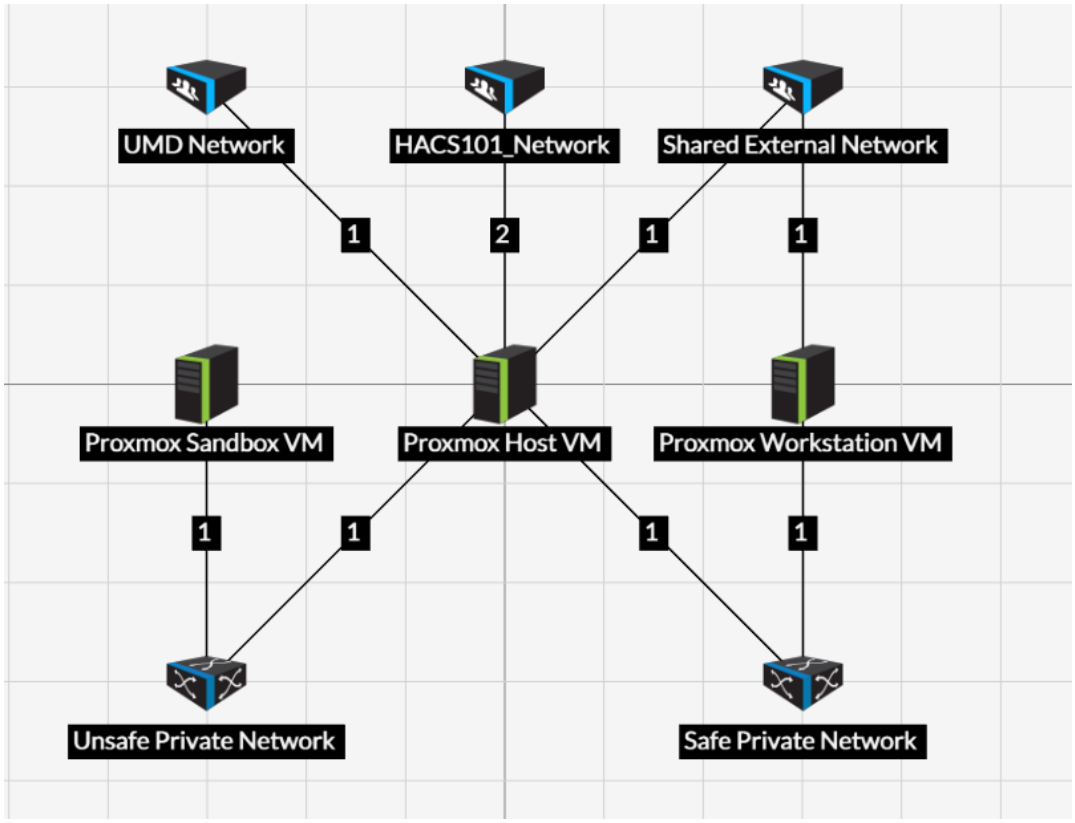
Number of Commands (excluding proc-cpu attacks & empty attacks): One-Way ANOVA Assumptions Report				
Normality (Anderson Darling P-Value): H_0 = data is sampled from a normal distribution	p = 0.000	p = 0.000	p = 0.000	p = 0.000
Robustness:	Minimum sample size for a robust ANOVA test = 19. Each sample size is not greater than this.			
Outlier Counts: Extreme (3*IQR), Likely (2.2*IQR), Potential (1.5*IQR)	Extreme = 30	None	Extreme = 11	Extreme = 18
Randomness (Non-parametric Runs Test P-Value): H_0 = data is random	p = -0.01	p = 0.25	p =0.0045	p = 0.777
Equal Variance (Levene's test for Equal Variances P-Value): H_0 = variances are equal	p = 0.142			

Section 6:

Figure 6.1:

Section 7:

Figure 7.1:



Appendix C:

monitor.sh

```
#!/bin/bash

ct_num=$1

if [ $ct_num -eq "103" ]; then
        ct_ip="172.20.0.4"
        d_port="10002"
fi

if [ $ct_num -eq "104" ]; then
        ct_ip="172.20.0.3"
        d_port="10001"
fi

if [ $ct_num -eq "101" ]; then
        ct_ip="172.20.0.2"
        d_port="10000"
fi

if [ $ct_num -eq "102" ]; then
        ct_ip="172.20.0.5"
        d_port="10003"
fi

while read a
do

    # Monitor for login event
    accepted_line=$(echo "$a" | grep "sshd" | grep "Accepted password for")
    if [ ! -z "$accepted_line" ]; then
    sleep 1
    # Extract IP address
    ip_addr=$(tail -n1 /root/MITM_data/logins/$ct_num.txt | cut -d";" -f2)

    date >> ~/logs/plain$ct_num.log
    echo "New connection from $ip_addr" >> ~/logs/plain$ct_num.log

    echo "$(date --rfc-3339=ns | cut -c -23) - [Ports] $(cat
~/recycling_scripts/port$ct_num.txt)" >> ~/logs/mitm$ct_num.log

    # Allow only the attacker in the container for the duration of attack
    iptables --table filter --insert INPUT 1 --protocol tcp --source $ip_addr --destination
172.20.0.1 --destination-port $d_port --jump ACCEPT
    iptables --table filter --insert INPUT 2 --protocol tcp --source 0.0.0.0/0 --destination
172.20.0.1 --destination-port $d_port --jump DROP

    /root/recycling_scripts/wait.sh $ct_num $ip_addr &
    exit
    fi
done
```

wait.sh

```
#!/bin/bash

ct_num=$1
ip_addr=$2

# Wait time
WAIT_TIME=300

if [ $ct_num -eq "103" ]; then
    ct_ip="172.20.0.4"
    d_port="10002"
fi

if [ $ct_num -eq "104" ]; then
    ct_ip="172.20.0.3"
    d_port="10001"
fi

if [ $ct_num -eq "101" ]; then
    ct_ip="172.20.0.2"
    d_port="10000"
fi

if [ $ct_num -eq "102" ]; then
    ct_ip="172.20.0.5"
    d_port="10003"
fi

# Wait for specified amount of time
echo "Started waiting for timeout on container $ct_num"
sleep $WAIT_TIME

# Check to see if malware.sh was created; if it was, block the user
# ct_files=$(ls -1 /var/lib/lxc/$ct_num/rootfs/root/)
# if echo $ct_files | grep "malware.txt"; then
    # Permanently block attacker
pct exec $ct_num -- killall sshd
iptables --table filter --insert INPUT 1 --protocol tcp --source $ip_addr --destination
172.20.0.1 --destination-port $d_port --jump DROP
echo "Blocked attacker for ip $ip_addr" >> ~/logs/plain$ct_num.log
#fi

# Delete previously created rules
iptables --table filter --delete INPUT --protocol tcp --source $ip_addr --destination
172.20.0.1 --destination-port $d_port --jump ACCEPT
iptables --table filter --delete INPUT --protocol tcp --source 0.0.0.0/0 --destination
172.20.0.1 --destination-port $d_port --jump DROP

echo "Beginning recycle of $ct_num" >> ~/logs/plain$ct_num.log
/root/recycling_scripts/recycle.sh $ct_num &
exit
```

recycle.sh

```
#!/bin/bash

ct_num=$1

if [ $ct_num -eq "103" ]; then
    ct_ip="172.20.0.4"
    d_port="10002"
fi

if [ $ct_num -eq "104" ]; then
    ct_ip="172.20.0.3"
    d_port="10001"
fi

if [ $ct_num -eq "101" ]; then
    ct_ip="172.20.0.2"
    d_port="10000"
fi

if [ $ct_num -eq "102" ]; then
    ct_ip="172.20.0.5"
    d_port="10003"
fi

#Kill currently running tail process
kill -9 $(ps aux | grep "tail -n 0 -F /var/lib/lxc/$ct_num/rootfs/var/log/auth.log" | awk
'{print $2}')
kill -9 $(ps aux | grep "tail -n 0 -F ~/logs/mitm$ct_num.log" | awk '{print $2}')
kill  $(ps aux | grep node | grep $ct_num | awk -F' ' '{print$2}')
kill -9 $(ps aux | grep "openPorts.sh $ct_num" | awk '{print$2}')
sleep 10

echo -n $({ cat ~/recycling_scripts/port$ct_num.txt}) >> ~/logs/port_data.log
echo -n " " >> ~/logs/port_data.log
echo $(ls -lt ~/MITM_data/sessions/ | head -n2 | awk '{print $9}'|cut -b -36) >>
~/logs/port_data.log

echo -n $ct_num >> ~/logs/port_data_better.log
echo -n " " >> ~/logs/port_data_better.log
echo -n $(cat ~/recycling_scripts/port$ct_num.txt) >> ~/logs/port_data_better.log
echo -n " " >> ~/logs/port_data_better.log
echo $(tail -1 ~/MITM_data/logins/$ct_num.txt) >> ~/logs/port_data_better.log

# Unmount container
pct unmount $ct_num
# Stop container
pct unlock $ct_num
pct stop $ct_num --skiplock 1
# Unmount container
pct unmount $ct_num
```

openPorts.sh

```
#!/bin/bash

ctnum=$1
random=$2
i=0
currentPort=0;
portCount=0;
lineCount=0;

cp /dev/null port.txt
python3 genPortFile.py

if [ $random -eq 1 ]
then
        portCount=0
elif [ $random -eq 2 ]
then
        portCount=25
elif [ $random -eq 3 ]
then
        portCount=50
elif [ $random -eq 4 ]
then
        portCount=100
fi

#print to a text file
echo $portCount > ~/recycling_scripts/port$ctnum.txt

while [ $i -lt $portCount ]
do
        lineCount=$(($i+1))
        currentPort=$(sed -n "$lineCount p" ~/recycling_scripts/port.txt)
        pct exec $ctnum -- ncat -l $currentPort --keep-open --exec "/bin/cat" & >>
~/logs/err$ctnum.log
        i=$(($i+1))
done
```

makeHoney.py

```
from faker import Faker
from faker.providers import internet
from random_words import RandomWords
from random import randint, choice
import string
import hashlib

def randomCapitalize(str):
    newStr = ""
    for x in str:
        if randint(1,10) % 2 == 0:
            newStr += x.upper()
        else:
            newStr += x
    return newStr

def main():
    Faker.seed(1)

    txtFile = open("records.txt", "w")
    hashFile = open("hash.txt", "w")
    emailFile = open("CompanyEmails.txt", "w")

    fake = Faker()

    fakeIP = Faker()
    fakeIP.add_provider(internet)

    rw = RandomWords()

    txtFile.write("----------------------------------------------------------------\n")
    txtFile.write("**********Important. Top Sneaky. Do Not Disclose.**********\n")
    txtFile.write("----------------------------------------------------------------\n")

    for _ in range(1000):
        txtFile.write(fake.name() + "\n")
        password = rw.random_word()
        while len(password) < 8:
            password = rw.random_word()
        password = randomCapitalize(password) + "".join(choice(string.punctuation +
string.digits) for x in range(randint(2, 6)))
        txtFile.write(password + "\n")
        emailFile.write(fake.email() + "\n")
        txtFile.write(fake.ipv4_private() + "\n\n")

        hashFile.write(hashlib.sha256(bytes(password,"utf-8")).hexdigest())
        hashFile.write("\n")

if __name__ == "__main__":
    main()
```

health_script.sh

```
#!/bin/bash

date > "health.log"

#Host data
free --mega | grep 'Mem:' | awk '{print $4}' >> "health.log"
df -m | grep /dev/mapper/pve-root | awk {'print $4'} >> "health.log"
uptime | awk 'NF>1{print $NF}' >> "health.log"
ifconfig | grep enp4s2 -A 7 | grep "RX packets" | awk '{print int($5/1024)}' >> "health.log"
ifconfig | grep enp4s2 -A 7 | grep "TX packets" | awk '{print int($5/1024)}' >> "health.log"

#C-101
pct exec 101 -- free --mega | grep 'Mem:' | awk '{print $4}' >> "health.log"
pct exec 101 -- df -m | grep /dev/mapper/pve-vm--101--disk--0 | awk {'print $4'} >>
"health.log"
pct exec 101 -- uptime | awk 'NF>1{print $NF}' >> "health.log"
pct exec 101 -- ifconfig | grep eth0 -A 7 | grep "RX bytes" | awk '{print $2}' | cut -c7- |
awk '{print int($1/1024)}' >> "health.log"
pct exec 101 -- ifconfig | grep eth0 -A 7 | grep "TX bytes" | awk '{print $6}' | cut -c7- |
awk '{print int($1/1024)}' >> "health.log"

#C-102
pct exec 102 -- free --mega | grep 'Mem:' | awk '{print $4}' >> "health.log"
pct exec 102 -- df -m | grep /dev/mapper/pve-vm--102--disk--0 | awk {'print $4'} >>
"health.log"
pct exec 102 -- uptime | awk 'NF>1{print $NF}' >> "health.log"
pct exec 102 -- ifconfig | grep eth0 -A 7 | grep "RX bytes" | awk '{print $2}' | cut -c7- |
awk '{print int($1/1024)}' >> "health.log"
pct exec 102 -- ifconfig | grep eth0 -A 7 | grep "TX bytes" | awk '{print $6}' | cut -c7- |
awk '{print int($1/1024)}' >> "health.log"

#C-103
pct exec 103 -- free --mega | grep 'Mem:' | awk '{print $4}' >> "health.log"
pct exec 103 -- df -m | grep /dev/mapper/pve-vm--103--disk--0 | awk {'print $4'} >>
"health.log"
pct exec 103 -- uptime | awk 'NF>1{print $NF}' >> "health.log"
pct exec 103 -- ifconfig | grep eth0 -A 7 | grep "RX bytes" | awk '{print $2}' | cut -c7- |
awk '{print int($1/1024)}' >> "health.log"
pct exec 103 -- ifconfig | grep eth0 -A 7 | grep "TX bytes" | awk '{print $6}' | cut -c7- |
awk '{print int($1/1024)}' >> "health.log"

#C-104
pct exec 104 -- free --mega | grep 'Mem:' | awk '{print $4}' >> "health.log"
pct exec 104 -- df -m | grep /dev/mapper/pve-vm--104--disk--0 | awk {'print $4'} >>
"health.log"
pct exec 104 -- uptime | awk 'NF>1{print $NF}' >> "health.log"
pct exec 104 -- ifconfig | grep eth0 -A 7 | grep "RX bytes" | awk '{print $2}' | cut -c7- |
awk '{print int($1/1024)}' >> "health.log"
pct exec 104 -- ifconfig | grep eth0 -A 7 | grep "TX bytes" | awk '{print $6}' | cut -c7- |
awk '{print int($1/1024)}' >> "health.log"
```

health_script.py

```
import gspread
import os
from oauth2client.service_account import ServiceAccountCredentials

# use creds to create a client to interact with the Google Drive and Google Sheets API
scope =
["https://spreadsheets.google.com/feeds",'https://www.googleapis.com/auth/spreadsheets',"https://www.googleapis.com/auth/drive.file","https://www.googleapis.com/auth/drive"]
creds = ServiceAccountCredentials.from_json_keyfile_name('health_log_creds.json', scope)
client = gspread.authorize(creds)

# Put the name of your spreadsheet here
sheet = client.open("Health Logs").sheet1
os.system("./health_script.sh")

# Example of how to insert a row
file1 = open('./health.log', 'r')
row = file1.readlines()
index = sheet.row_count
sheet.append_row(row)
```

parse_mitm.py

```
#!/usr/bin/python

import os
import sys
import datetime
import re
import csv
from random import randint

# Date:
# 2020-11-17 20:42:55.846 -
date_re = r"(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}.\d{3}) - "

class Attack:
        typeAttack = "Unknown" #"human" "bot" "empty"
        containerID = 0
        numPorts = -1
        numCmds = -1
        attackerIP = "Unknown"
        commands = "Unknown"
        startTimeStr = "Unknown"
        endTimeStr = "Unknown"
        cmds = [] #list of strings
        raw_cmd = ""

        #datetime
        startTimeObj = datetime.datetime.strptime("1900-01-01 00:00:00.000","%Y-%m-%d %H:%M:%S.%f")
        endTimeObj = datetime.datetime.strptime("1900-01-01 00:00:00.000","%Y-%m-%d %H:%M:%S.%f")
        duration = startTimeObj - endTimeObj

#sessionID,containerID,conatinerIP,containerIP
#look .csv file
        def getDuration(self):
                if self.startTimeStr != "Unknown" and self.endTimeStr != "Unknown":
                        self.startTimeObj = datetime.datetime.strptime(self.startTimeStr, "%Y-%m-%d
%H:%M:%S.%f")
                        self.endTimeObj = datetime.datetime.strptime(self.endTimeStr, "%Y-%m-%d
%H:%M:%S.%f")
                        if (self.startTimeObj is not None) and (self.endTimeObj is not None):
                                self.duration = self.startTimeObj - self.endTimeObj

                return self.duration

def main():
        start_str = "2020-11-17 20:00:00.000000"
        end_str = "2020-12-07 23:59:00.000000"
        start_obj = datetime.datetime.strptime(start_str, "%Y-%m-%d %H:%M:%S.%f")
        end_obj = datetime.datetime.strptime(end_str,"%Y-%m-%d %H:%M:%S.%f")

        # lst = getAttacksByDay(start_obj, 8)
        # getAttacksInRange(start_obj, end_obj)

        print(len(getAttacks(start_obj, end_obj, "101")))
        print(len(getAttacks(start_obj, end_obj, "102")))
        print(len(getAttacks(start_obj, end_obj, "103")))
        print(len(getAttacks(start_obj, end_obj, "104")))
```

```
def printAttacks(atkArr):
        for atk in atkArr:
                print(atk)

def isValidDatetime(str, start_obj, end_obj):
        obj = datetime.datetime.strptime(str,"%Y-%m-%d %H:%M:%S.%f")
        return obj > start_obj and obj < end_obj

def getAttacks(start_obj, end_obj, conID):
        log_file = open("/root/logs/mitm" + conID + ".log", "r")
        atkList = []
        att = Attack()
        isCurrentlyProcessing = False
        lineNum = 0

        with open('/root/parse_scripts/attacks' + conID + '.csv', 'w', newline='') as file:
                        file.write("")
                        file.close()

        for line in log_file:
                # print(line)
                if checkConnectionLine(line) is not None and not isCurrentlyProcessing:
                        # print("valid check connect")
                        res = checkConnectionLine(line)
                        att = Attack()
                        att.containerID = int(conID)
                        att.attackerIP = res.group(2)
                        isCurrentlyProcessing = True

                if checkEnteredLine(line) is not None:
                        # print("valid check enter")
                        res = checkEnteredLine(line)
                        att.startTimeStr = res.group(1)
                        att.endTimeStr = res.group(1)
                        # print("started: " + att.startTimeStr)
                        if not isValidDatetime(att.startTimeStr, start_obj, end_obj):
                                att = Attack()
                                continue

                if checkPortLine(line) is not None:
                        # print("valid check port")
                        res = checkPortLine(line)
                        att.numPorts = int(res.group(2))

                if checkBotCmd(line) is not None:
                        # print("valid check cmd")
                        res = checkBotCmd(line)
                        att.endTimeStr = res.group(1)
                        att.raw_cmd = res.group(2)
                        lst = att.raw_cmd.split(";")
                        lst = [item.split("|") for item in lst]
                        lst = [item.strip() for sublist in lst for item in sublist]
                        att.cmds = att.cmds + lst

                if checkExitLine(line) is not None:
                        res = checkExitLine(line)
                        # print("getAtk: start time: " + att.startTimeStr)
                        # print("getAtk: end time: " + att.endTimeStr)
```

```
                        att.getDuration()
                        att.numCmds = len(att.cmds)

                        isCurrentlyProcessing = False

                        if att.numCmds == 0:
                                att.typeAttack = "empty"
                        else:
                                att.typeAttack = "bot"

                        with open('/root/parse_scripts/attacks' + conID + '.csv', 'a', newline='') as
csvfile:
                                w = csv.writer(csvfile, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)
                                w.writerow([att.startTimeStr, att.endTimeStr,
str(att.duration.seconds),\
                                                                        str(att.containerID),
att.attackerIP, att.typeAttack,\
                                                                        str(att.numPorts),
str(att.numCmds), " \\ ".join(att.cmds),
                                                                        str(att.raw_cmd)])
                                csvfile.close()
                        atkList.append(att)
                        att = Attack()
                        # print("valid check exit")

                lineNum += 1

        log_file.close()
        print("Processed " + str(lineNum) + " lines")

        return atkList

def checkConnectionLine(line):
        # Connection Line:
        # 2020-11-17 20:42:32.529 - [Debug] [Connection] Attacker connected: 154.8.226.52
        connect_re = date_re + r"\[Debug\] \[Connection\] Attacker connected:
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})"
        return re.search(connect_re, line)

def checkEnteredLine(line):
        # Entered Line:
        # 2020-11-17 20:42:55.846 - [Debug] [LXC-Auth] Attacker authenticated and is inside container
        enter_re = date_re + r"\[Debug\] \[LXC-Auth\] Attacker authenticated and is inside container"
        return re.search(enter_re, line)

def checkPortLine(line):
        # Port Num Line:
        # 2020-11-17 20:42:56.852 - [Ports] 50
        port_re = date_re + r"\[Ports\] (\d{1,3})"
        return re.search(port_re, line)

def checkBotCmd(line):
        # Bot Cmd Line:
        # 2020-11-17 20:42:35.185 - [Debug] [EXEC] Noninteractive mode attacker command: cat
/proc/cpuinfo | grep name | wc -l
        bot_cmd_re = date_re + r"\[Debug\] \[EXEC\] Noninteractive mode attacker command: (.*)"
        return re.search(bot_cmd_re, line)
```

```
def checkExitLine(line):
        # Exit Line:
        # 2020-11-18 00:44:58.088 - [Info] Exiting...
        exit_re = date_re + r"\[Info\] Exiting..."
        return re.search(exit_re, line)

if __name__ == "__main__":
    main()
```

Works Cited

Arntz, Pieter. “Brute Force Attacks Increase Due to More Open RDP Ports.” Malwarebytes Labs, Malwarebytes, 19 Oct. 2020, blog.malwarebytes.com/exploits-and-vulnerabilities/2020/10/brute-force-attacks-increasing/.

Lee, Cynthia Bailey, Chris Roedel, and Elena Silenok. "Detection and characterization of port scan attacks." *Univeristy of California, Department of Computer Science and Engineering* (2003).

Mathew, Kuruvilla, Mujahid Tabassum, and Marlene Valerie Lu Ai Siok. "A study of open ports as security vulnerabilities in common user computers." 2014 International Conference on Computational Science and Technology (ICCST). IEEE, 2014.

Yin, Chunmei, et al. “Honeypot and Scan Detection in Intrusion Detection System.” Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04CH37513), vol. 2, 5 May 2004, pp. 1107–1110., doi:10.1109/ccece.2004.1345313.