Final Report

Honeypot Project: Analyzing Attacker Behavior with Varying Security Measures

Brikti Nriayo, Sai Divvela, Jonathan Lee, Anish Parepalli

Group 1G

HACS200

Summary

This final paper presents the findings of a honeypot project designed to study attacker behavior in response to different security measures. The primary research question is: *How does varying the strength of passwords affect how deep attackers traverse file systems?* We hypothesized that attackers would adjust their interaction with the file system of the system. The data collected involved monitoring attacker interactions with a honeypot that featured a corporate database, and utilized passwords of varying strengths to protect it. There were 4 different configurations, each corresponding to the strength of the passwords used to protect the corporate filesystem. These configurations were "None", "Low", "Medium" and "High" respectively. In total, we had 23,919 external connections to our 5 honeypots, as of December 1st 6pm. We collected a significant amount of data through the ACES MITM. The key metrics we collected were authentication attempts, passwords used, IP addresses that connected to the system, keystrokes and commands run, as well as time of login/logout. Using this, we were able to make some key conclusions about the data. First, we were able to determine that 46.6% of attackers were bots. We defined a bot to be any connection that lasted for less than two seconds, and entered at least 1 command. Additionally, we noticed that a lot of bots followed a very similar pattern of commands entered. We were able to find statistical differences between our honeypots with the "High" configuration and the "None" or "Weak" configurations when looking at the number of commands run. However, we also noticed that only 7.1% of all attackers entered the `cd` command, and those that entered the `cd` command did not interact with the corporate filesystem. While we did see a change in attacker behavior with respect to the honeypot configuration, because attackers did not interact with the honey in any meaningful way, we were unable to reject the null hypothesis and derive meaningful conclusions.

Background Research

Previous research has shown that attackers will opt for the path of least resistance when attacking systems, opting to go after systems that have a lower amount of security measures. (Cremonini 2006) The same paper shows that systems that have a higher level of security are incentivized to reveal this fact, as it can discourage attackers from even attempting to attack the system.

However, when not revealing this information, it is possible to trick attackers into attacking a system they might not have attempted to attack otherwise, thus giving insight into behavior when presented with different levels of security. Additional research shows that using “deception strategies” can help defenders to understand attacker behavior and “provides an edge in the OODA (Observe, Orient, Decide and Act) loop.” (Almeshekah et al. 2014) These papers suggest that using different levels of security for different systems gives insight into attacker behavior.

Based on previous studies using a high-interaction honeypot, the majority of attacks that access a system internally are done by humans (Nicomette et al. 2009). This means that we can observe the real-time behavior of attackers. While the relative skill level of these human attackers are hard to ascertain, tracking attacker behavior on honeypots still gives deeper insight into how attackers will respond to different security levels, as it will attract random attacks from different individuals (“What).

Based on previous research, it is clear that attackers will change their behavior based on the security level of the system that they are attacking. Due to this, we determined that it would be valuable to test different security models in order to examine how attackers modify their behavior in order to evaluate the effectiveness of different security measures.

Once we submitted our proposal, we realized that the majority of attackers that will connect to our honeypot are bots. This means that they will be following a predetermined script, and will not be adjusting their tactics based on security measures. Furthermore, this meant that much of the activity that would be detected would largely be the same, and that we would need more sophisticated data analyses in order to determine the impact of security measures on attacker behavior.

Experiment Design Changes

The experiment design from our initial proposal focused on testing the hypothesis that attackers would change their tactics when exposed to varying security measures. Originally, the plan was to implement a standard honeypot with multiple levels of security, including firewalls, password protections, and monitoring systems. However, after further consultation and feedback, we re-examined the project to incorporate one level of security, primarily focusing on password and password strength. This change was motivated by the inability of balancing sophisticated defenses with specialized monitoring tools to track attacker behavior precisely.

Furthermore, the implementation of a corporate database involved a lot of moving parts, which we did not anticipate when first presenting our proposal. Initially, we thought that linux file systems would have a way to natively password protect files and data. However, we soon realized that this was not the case. Our first attempt to instantiate a password protected database was to have zipped data files for attackers to look at. While working on this idea, we realized that many attackers would not interact with a zip file located on a file system.

The second approach that we had was to use a completely new file system to store our honey. However, we quickly realized that introducing a foreign and new file system would deter attackers from interacting with it, rather than encouraging them to. Finally, we settled on using poisoned commands to enforce a password when using the `cd` command into a particular file system, which would mimic a corporate database.

In order to effectively ensure that the particular file system was password protected, we had to poison the `cd` command, so that the password prompt would first appear, and then if the password matched what was required by the particular configuration, then it would allow the `cd` command to go through. We also had to ensure that attackers could not modify this logic, so we used PAM in order to ensure that all users had to go through this loop, but that only the root user would be able to modify the password requirement. Additionally, since we prevented attackers from being able to log in as root, this ensured that the password protection and `cd` command could not be tampered with.

Research Question and Hypothesis

The research question of this experiment was: *How does varying the strength of passwords affect how deep attackers traverse file systems?*

The hypothesis posits that as password strength increases, attackers will be less likely to traverse deeper into the file system. Stronger passwords would deter attackers due to the higher computational and time cost required to crack them. The **null hypothesis (H_0)** states that password strength has no effect on attacker behavior, and attackers will traverse file systems equally regardless of password protection.

Hypotheses:

- **H_0**: Password strength has no effect on how deep attackers traverse the file system.

- **H**$_1$: A weak password will affect how far an attacker traverses through the file system.
- **H**$_2$: A medium-strength password will affect how far an attacker traverses through the file system.
- **H**$_3$: A strong password will affect how far an attacker traverses through the file system.

Experiment Design

This experiment utilized Bash scripts and LXC containers to study attacker behavior across five external IPs. Each IP was managed by a background recycling script that automated the initialization, configuration, and teardown of LXC containers. The containers ran the default Ubuntu file system and included a multi-level directory structure containing fake sensitive corporate data. Access to these directories was password-protected using a poisoned cd command.

The experiment implemented four security configurations: no password, weak password, medium password, and strong password. Configurations were selected using a random number generator, ensuring diverse setups while maintaining reproducibility. A separate Bash script created the fake file system within each container based on the selected configuration. Passwords for the directories were pre-generated and verified for strength using the password evaluation tool on **security.org.**

Process Workflow

Each recycling script operated as a background process and executed the following steps:

1. **Container Initialization**:
   - Created an LXC container running the default Ubuntu file system.

- Installed and verified SSH to ensure operational connectivity.

2. **Configuration Selection**:
    - Used a random number generator to select one of the four predefined configurations (no password, weak password, medium password, strong password).
3. **File System Creation**:
    - Invoked a separate Bash script to create a multi-level directory structure containing fake sensitive financial data.
    - Poisoned the cd command to require a pre-assigned password for directory access, with the password strength corresponding to the selected configuration.
4. **MITM Server Deployment**:
    - Instantiated a Man-in-the-Middle (MITM) server using pm2 to monitor and log attacker actions.
5. **Traffic Routing**:
    - Configured IP rules to route traffic from the external IP to the MITM server and then to the LXC container.
    - Upon detecting an attacker's IP via MITM server logs, the script updated IP rules to allow only that specific IP address to connect to the container, blocking access from other sources.
6. **Recycling**:
    - Upon attacker disconnection or the expiration of a five-minute session timeout:
        - Destroyed the container.
        - Cleared all associated IP rules.

      - Terminated the MITM server process.

Continuous Deployment

To ensure uninterrupted availability, a background process prepared a new container while the active container was operational. When the active container was recycled, the pre-initialized container was immediately deployed, and another background process began preparing the next container. This ensured zero downtime across all five external IPs.

Monitoring and Data Collection

Two main logging mechanisms were employed:

1. **MITM Server Logs**:
   - Captured all attacker actions within the containers.
   - Logs were compressed periodically using a Bash script, which archived logs for every 1,000 attackers and moved them to a directory for manual upload to Google Drive.
2. **Attacker Tracking Script**:
   - A separate monitoring script ran at hourly and daily intervals to record the number of attackers for each IP and ensure the system functioned correctly.
   - The logs generated by this script allowed quick identification of potential issues, ensuring consistent operation of the experiment.

Host Machine Configuration

The host machine adhered to the Division of IT's default firewall rules, maintaining compliance with institutional security policies. We installed netdata for real-time monitoring of system performance. Additionally, it provided insights into resource utilization and overall system health. A reboot script ensured the environment remained consistent during resets by:

- Terminating all active containers, pm2 processes, and background scripts.
- Clearing any lingering IP rules.
- Reinitializing all five recycling scripts and firewall rules to restore a clean system state.

This design combined automated container recycling, dynamic IP rule management, and detailed logging to study attacker behavior effectively. The use of a random number generator for configuration selection, alongside dedicated scripts for file system creation and attacker tracking, ensured both scalability and reliability throughout the experiment.

Data Collection

The data collection process was automated through the MITM server, which generated logs capturing key attacker behaviors and interactions. The following data points were collected:

1. **Connection Details**:
    - **Time of Connection**: The timestamp when an attacker connected to the MITM server.
    - **Attacker IP**: The IP address of the attacker initiating the connection.
2. **Disconnection Details**:
    - **Time of Disconnection**: The timestamp when the attacker disconnected from the MITM server.

3. **Interaction Logs**:
   - **Commands Executed**: A record of all commands the attacker ran within the container.
   - **Keystrokes**: A detailed log of all individual keystrokes made during the session.
4. **Unsuccessful Access Attempts**:
   - **Connection Attempts**: The IP addresses and timestamps of other attackers who attempted to connect while access was restricted to a single attacker.
   - **Password Guesses**: Any passwords used by unauthorized attackers attempting to gain access.

Logs were stored in real-time on the host system and organized by session. Each log file was named based on the associated external IP and the selected security configuration. Periodically, the logs for every 1,000 attackers were compressed into archives using a Bash script and moved to a designated directory for manual upload to Google Drive. This naming convention and archival process ensured secure storage, efficient organization, and ease of access for subsequent analysis.

The experiment yielded a total of **23,919 attackers**, of which **4,459** were unique. Each attacker's session was documented with detailed logs capturing key attributes of their interactions within the honeypot environment. These attributes included security configuration, IP address, login/logout timestamps, and the nature of their activities.

To ensure the data's reliability and relevance, several steps were taken to process, clean, and prepare the collected information for analysis.

Data Processing Workflow

1. **Identification and Categorization**

   Attackers were identified using timestamps from their login/logout filenames. Attributes such as:

   - **Honeypot assignment:** (1 through 5)

     These are used to identify which attackers came from which honeypot IP.

   - **Security configuration** (None, Weak, Medium, High)

     These align with the experimental design to examine behavioral differences across security levels.

2. **Command Analysis**

   The keystrokes directory was analyzed for evidence of commands executed during the session. Commands were counted by:

   - Searching interactive and non-interactive input files.
   - Parsing commands using semicolons or other delimiters to identify discrete entries.

3. **Session Duration Calculation**

   Login and logout timestamps were processed to calculate the duration of each session. This was recorded as the **time spent in the honeypot**, a key metric for understanding attacker behavior.

4. **Data Exportation**

   Processed data, including all calculated metrics, was compiled into CSV files and uploaded to Google Drive for storage and further analysis.

Data was grouped primarily by **security level** (None, Weak, Medium, High), as this was the basis of the experimental design. Each attacker's actions, session duration, and other interactions were analyzed in the context of the assigned security configuration. Furthermore, we filtered by IP address to provide a more accurate representation of the trends that occur.

To improve data quality and prepare it for analysis we had to eliminate some of the data; the following steps were taken to remove inaccurate or incomplete entries:

1. **Duplicate Entries**

   Some login/logout mismatches, arising from a bug in the MITM system that occasionally recorded duplicate logins, were excluded. This ensured that each session was uniquely identified.

2. **Test Data**

   There were some entries which were test data from our team, so those had to be removed to perverse the experiment's integrity.

Eliminated Data Points

A total of 178 data points were removed through these processes.

The total number of valid data points is 23919.

Data Interpretation

As hinted to above, we use the following metrics to interpret our data and see if our hypotheses are rejected:

1. **Commands Run**

   Representing the extent of an attacker's engagement, this metric served as a proxy for intent or persistence.

2. **Time Spent in the Honeypot**

   This reflects the attacker's interest in the environment and their ability to overcome security obstacles.

These dependent variables were evaluated against the **security level**, the independent variable, to assess how different configurations influenced attacker behavior. We can also filter out commands that are not "cd" to refine our data even more. This focus enabled a direct investigation into the impact of security measures on engagement patterns.

Data Analysis

The data was analyzed using basic statistical methods to examine the frequency of time spent by attackers at each password strength. The key question was whether stronger password strength led to a reduction in the duration of the attacker's engagement. In terms of hypothesis testing, we performed ANOVA testing to compare the attack frequencies across the three password strengths, violin plots to visualize the distribution of attackers at each password level, and the Kruskal-Wallis test applied to assess whether there were statistically significant differences between the groups (low, medium, and high strength password) based on the duration of attacker engagement to determine if there was a significant difference in the time spent by attackers and password strength. We additionally filtered based on unique ip addresses as well, to filter out spamming by repeat attackers. In particular, we only take the first attack from each IP address, as this best enables us to understand the true behavior of an attacker. While performing these tests, we were able to reject the null hypothesis for the time spent on both overall attacks, and also while filtering on unique attacks. In particular, we had a p-value of 0.09 from running the ANOVA test on all attacks with respect to time spent and a p-value of 0.04 from Kruskal-Wallis when only considering unique IP addresses. When performing the Tukey post-hoc test on all attackers, we could not find any statistical difference between the time spent in the different configurations. However, when performing Dunn's post hoc test, we found that the time spent between the honeypot in the “High” password configuration and the “Weak” password configuration were statistically significant, as the p-value was approximately 0.028. We found this to be of particular interest as for the vast majority of the time that the honeypots were deployed, we did not have p-values this low, but now with much more data we have statistically significant differences.

In addition to tracking the time spent in the honeypot, we also tracked the number of commands run by attackers. This was important because it gave us an understanding of how attackers were interacting with our system. We performed ANOVA and Kruskal-Wallis tests for the same reasons as above. While performing these tests, we found that there was a statistically significant correlation between different configurations. In particular, we found a p-value of 0.013 from ANOVA and a p-value of 0.027 from Kruskal Wallis when accounting for all attacks. In order to analyze the correlation further, we performed a post-hoc analysis using Tukey’s post hoc test and Dunn’s post hoc test. Using these tests, we determine that there were statistically significant differences when comparing the “None” configuration to the “Medium” and “High” configurations. When solely filtering on unique attacks, we see an even stronger connection. The p-value for ANOVA and Kruskal-Wallis was 0.003. When performing the post-hoc analysis, we noticed statistically significant differences between the number of commands run on the “None” and “Weak” configurations when compared to the “High” configuration.

Finally, we performed an additional analysis to learn more about the impact that bots had on our system. We defined a bot to be an attacker that connects to our system for under 2 seconds, and runs at least one command. Out of our 23,956 connections, we found that 11,169 connections met this criteria. On average, bots entered about 1.5 commands, before quickly exiting. The most common commands entered by bots are:

- uname -s -v -n -r -m
- cd ~
- chattr -ia .ssh

The first command is uname, which is a common command for determining system information. We believe that many bots use this command to determine specific details of the system, such as what version of OS it is running. This is important for bots as certain versions of operating systems have easy to exploit vulnerabilities which can be automated, so being able to check the version quickly allows bot attackers to determine instantly whether or not to continue with the attack. The cd command was mostly used to move into the home directory. Many bot attacks used external scripts, which all require a particular directory to be in so that they can run effectively. In order to run scripts correctly, many bots used the `cd` command to change into the home directory. The final command is used to attempt to modify the attributes of a particular file. In this case, the file is the .ssh file, which controls the ssh login policy. With the chattr command, attackers attempted to make the ssh file available to be modified so that they would be able to regain access to the system with root, using privilege escalation. However, they were not successful in doing so as they did not have access to the chattr command.

By doing this analysis, we were able to obtain much deeper insight into how bots were able to interact with the honeypot. This allowed us to understand how bots impacted our honeypot, and what were the most common behaviors that we could observe from these bot attackers. However, when performing our initial statistical tests, we included these attacks as well so that we would have complete data, and be able to accurately determine the statistical differences across configurations.

However, despite having statistically significant differences in both the time spent and the number of commands run, we ultimately were unable to reject our null hypothesis. This is due to the fact that no attacker ever interacted with the corporate database, thus having to enter passwords. As a result, we were unable to observe any direct impact that the passwords had on traversing file systems. While performing additional analysis, we discovered that there was no correlation between the time of day and the number of attacks. We used the chi-squared test in order to test for statistical significance.

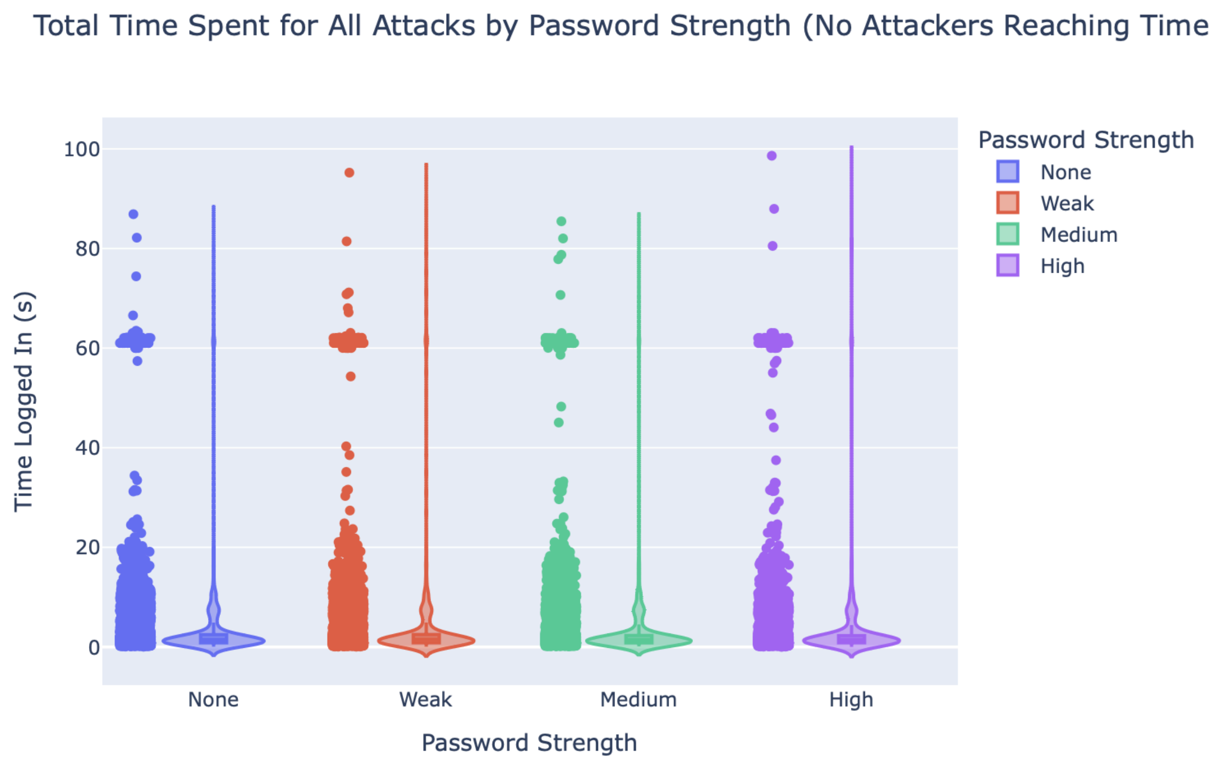
Conclusion

Our project has provided valuable insights into attacker behavior in honeypot environments. We observed that while attackers do adapt their tactics depending on the strength of the passwords, there was no interaction with the database, which was password protected. The data collected from our experiment suggests that attackers continue to probe systems at various security levels, though their strategies may shift. However, it is unclear how attackers were able to adjust their strategy, despite not interacting with the honey. Future work in this area could involve testing different honeypot configurations, adding different types of security measures which would force more attacker interaction, and analyzing behavior patterns on a larger and longer scale. While honeypots remain useful tools for cybersecurity, there is still much to explore regarding their effectiveness in predicting and preventing attacks in real-world scenarios.

Appendix A: Lessons Learned

As we worked through the project, we ran into some challenges that made us rethink our approach. One unexpected challenge was the complexity of our originally proposed research question being focused on unique attacker behavior across different security measures. The problem with this proposal was that our security measures were too complex and our approach to measuring attacker behavior was too vague of a concept making it tough to collect data effectively. Once realizing our research question was too broad and tried to accomplish too many things at once we decided to simplify our question and what we tracked so that it would be easier to implement and conduct our experiment. In the end, we focused on how attackers interact with honey, specifically in relation to password strength, which made it easier to manage and run our honeypot as we collected data. Finally, we also realized that the location of the honey made it difficult for attackers to access it, as it was located under the root directory. We thought that this would make the honey appear more legitimate, but we realized that most attackers were scouring the home directory. So, for future experiments, we will include more honey at the home level, rather than at the root level.

Appendix B: Tables and Graphs

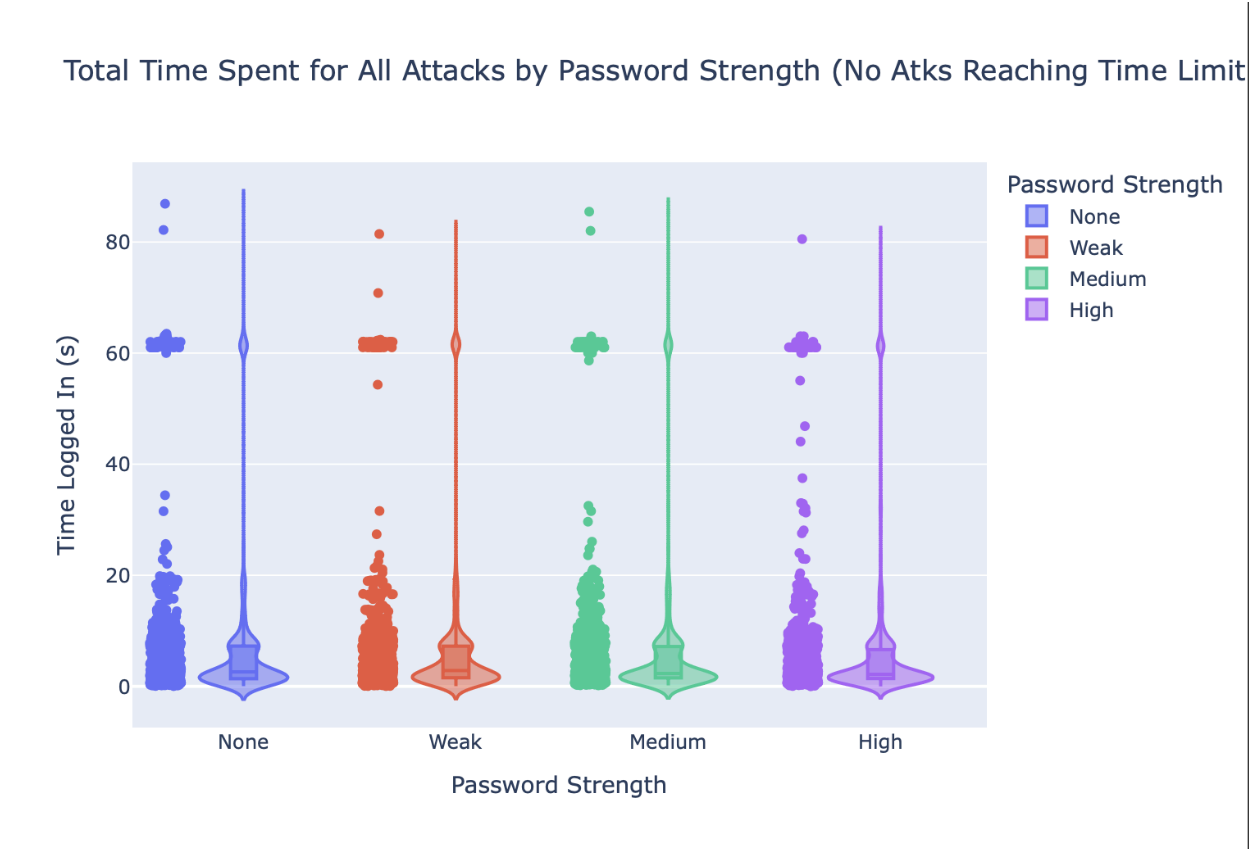



```
ANOVA Results for Time Spent (No Attackers Reaching Time Limit)
f-statistic: 2.1108
p-value: 0.0965
ANOVA Results for Time Spent (No Attackers Reaching Time Limit)
                       sum_sq       df         F    PR(>F)
C(Q("Pwd Str"))  3.789795e+02      3.0  2.110826  0.09655
Residual         1.037148e+06  17330.0       NaN      NaN

ANOVA indicates a significant difference. Running Tukey's HSD post-hoc test...
Multiple Comparison of Means - Tukey HSD, FWER=0.10
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
  High Medium  -0.0778 0.9741 -0.4967 0.3412  False
  High   None    0.197 0.7046 -0.2228 0.6167  False
  High   Weak   0.3127  0.313 -0.1038 0.7292  False
Medium   None   0.2747 0.4364 -0.1445  0.694  False
Medium   Weak   0.3905 0.1373 -0.0255 0.8064  False
  None   Weak   0.1157 0.9203 -0.3011 0.5325  False
---------------------------------------------------
```

```
Kruskal Wallis Results for Time Spent (No Attackers Reaching Time Limit)
h-statistic: 3.4739
p-value: 0.3242

Kruskal Wallis indicates no significant difference. No need for post-hoc analysis.
```

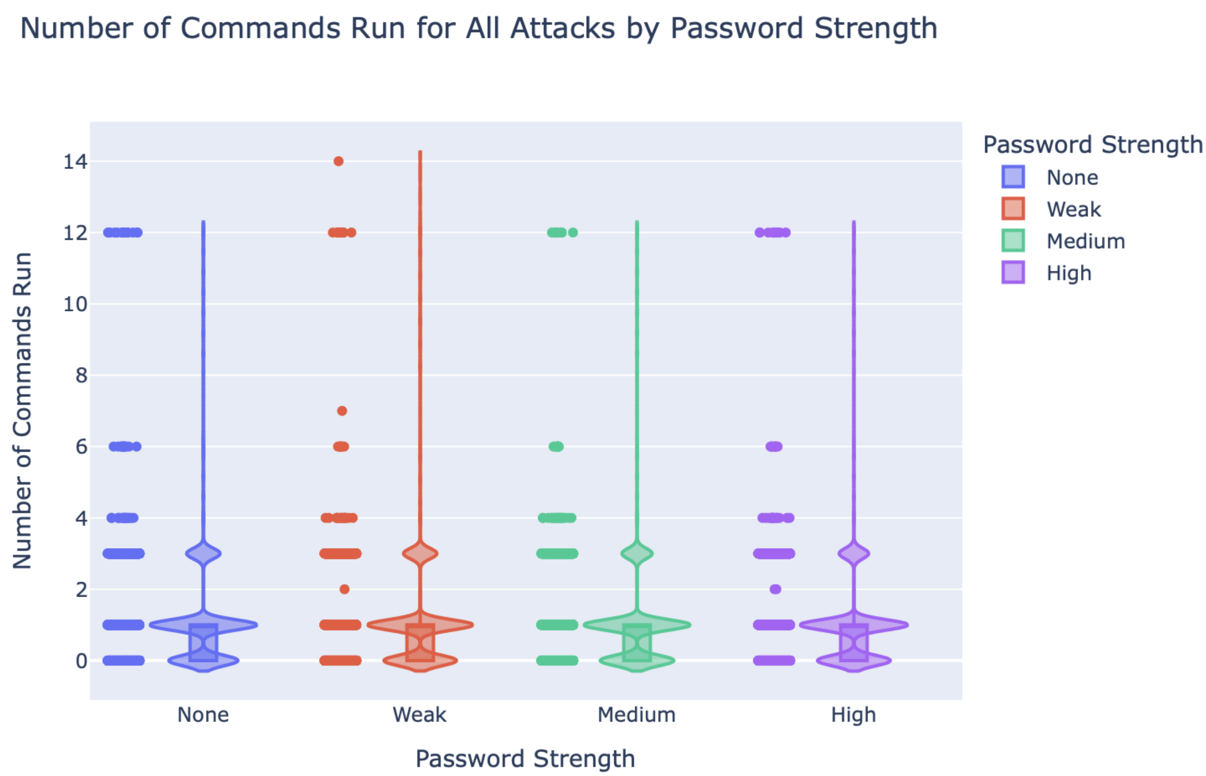



```
ANOVA Results for Time Spent (Only Unique Attackers, No Attackers Reaching Time Limit)
f-statistic: 0.7085
p-value: 0.5468
ANOVA Results for Time Spent (Only Unique Attackers, No Attackers Reaching Time Limit)
                      sum_sq      df         F    PR(>F)
C(Q("Pwd Str"))   347.876113     3.0  0.708537  0.546816
Residual       589828.469717  3604.0       NaN       NaN

ANOVA indicates no significant difference. No need for post-hoc analysis.
```

```
Kruskal Wallis Results for Time Spent (Only Unique Attackers, No Attackers Reaching Time
h-statistic: 8.0783
p-value: 0.0444

Kruskal Wallis indicates a significant difference. Running Dunn post-hoc test...
            High   Medium      None      Weak
High    1.000000  0.84272  0.605914  0.027724
Medium  0.842720  1.00000  1.000000  1.000000
None    0.605914  1.00000  1.000000  1.000000
Weak    0.027724  1.00000  1.000000  1.000000
```

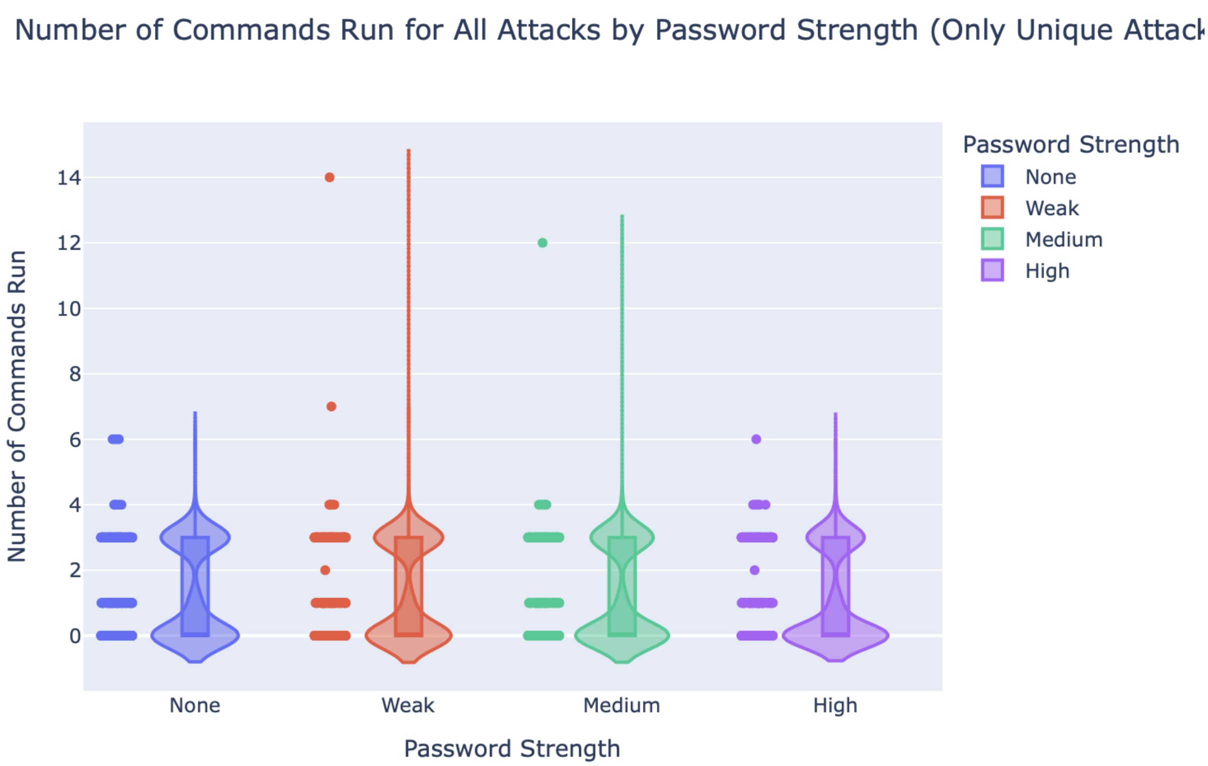



```
ANOVA Results for Commands Run
f-statistic: 3.7582
p-value: 0.0103
ANOVA Results for Commands Run
                      sum_sq       df         F    PR(>F)
C(Q("Pwd Str"))     15.731611      3.0  3.758212  0.010346
Residual         24193.280094  17339.0       NaN       NaN

ANOVA indicates a significant difference. Running Tukey's HSD post-hoc test...
Multiple Comparison of Means - Tukey HSD, FWER=0.10
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
  High Medium  -0.0011    1.0 -0.0594 0.0572  False
  High   None   0.0696 0.0319  0.0112  0.128   True
  High   Weak   0.0449 0.2857 -0.0131 0.1028  False
Medium   None   0.0707  0.028  0.0124 0.1291   True
Medium   Weak    0.046 0.2639 -0.0119 0.1038  False
  None   Weak  -0.0248 0.7615 -0.0828 0.0332  False
---------------------------------------------------
```

```
Kruskal Wallis Results for Commands Run
h-statistic: 9.1745
p-value: 0.0271

Kruskal Wallis indicates a significant difference. Running Dunn post-hoc test...
            High    Medium      None  Weak
High    1.000000  1.000000  0.064679   1.0
Medium  1.000000  1.000000  0.056085   1.0
None    0.064679  0.056085  1.000000   1.0
Weak    1.000000  1.000000  1.000000   1.0
```



```
ANOVA Results for Commands Run (Only Unique Attackers)
f-statistic: 6.3399
p-value: 0.0003
ANOVA Results for Commands Run
                      sum_sq      df         F    PR(>F)
C(Q("Pwd Str"))    41.207054     3.0  6.339935  0.000277
Residual         7810.354597  3605.0       NaN       NaN

ANOVA indicates a significant difference. Running Tukey's HSD post-hoc test...
Multiple Comparison of Means - Tukey HSD, FWER=0.10
====================================================
group1 group2 meandiff p-adj   lower  upper  reject
----------------------------------------------------
  High Medium   0.1405 0.1851 -0.0199 0.3008  False
  High   None   0.2483 0.0021  0.0884 0.4083   True
  High   Weak   0.2691 0.0005  0.1116 0.4266   True
Medium   None   0.1078 0.4127 -0.0525 0.2682  False
Medium   Weak   0.1286 0.2423 -0.0293 0.2865  False
  None   Weak   0.0208 0.9904 -0.1367 0.1783  False
----------------------------------------------------
```

```
Kruskal Wallis Results for Commands Run
h-statistic: 18.5807
p-value: 0.0003

Kruskal Wallis indicates a significant difference. Running Dunn post-hoc test...
            High    Medium      None      Weak
High    1.000000  0.401367  0.001781  0.001080
Medium  0.401367  1.000000  0.454010  0.364936
None    0.001781  0.454010  1.000000  1.000000
Weak    0.001080  0.364936  1.000000  1.000000
```

```
Chi-squared statistic: 52028.99999999999
P-value: 0.4954653915603123
Degrees of freedom: 52026
There is no statistically significant association between time of day and number of logins
```

Appendix C: Scripts

Daily Attacker Count:

```
#!/bin/bash



# Define the log file path

LOG_FILE="/home/lxc_list_log.txt"
```

```
# Print the current date and time as a header for each entry
echo "----- $(date +'%Y-%m-%d %H:%M:%S') -----" >> "$LOG_FILE"


# Run 'sudo lxc-ls' and append the output to the log file
sudo lxc-ls >> "$LOG_FILE"
sudo bash reboot.sh
# Add a newline for readability
echo "" >> "$LOG_FILE"
```

Hourly Attacker Count

```
#!/bin/bash


# Define the log file path
LOG_FILE="/home/lxc_list_log_hour.txt"


# Print the current date and time as a header for each entry
echo "----- $(date +'%Y-%m-%d %H:%M:%S') -----" >> "$LOG_FILE"


# Run 'sudo lxc-ls' and append the output to the log file
sudo lxc-ls >> "$LOG_FILE"


# Add a newline for readability
echo "" >> "$LOG_FILE"
```

Collect Zip Files:

```
#!/bin/bash

# Set the base directory to collect zip files by IP
base_dir="/home"  # Directory where ExternalIP_x folders are located
all_zips_dir="${base_dir}/all_zips"

# Create a base directory to store all ZIP files
sudo mkdir -p "$all_zips_dir"

# Function to collect zip files from a specific ExternalIP directory
collect_zips() {
    local ip_dir="$1"
    local ip_number="$2"
    local target_dir="${all_zips_dir}/ExternalIP_${ip_number}"

    echo "Collecting zip files from $ip_dir..."

    # Check if the source directory exists
    if [ -d "$ip_dir" ]; then
        # Create target directory if it doesn't exist
        sudo mkdir -p "$target_dir"
```

```
        # Copy all zip files from the source directory to the target directory
        sudo cp "$ip_dir"/*.zip "$target_dir/"
        echo "Zip files collected to $target_dir"
    else
        echo "Directory $ip_dir does not exist. Skipping..."
    fi
}

# Loop through each ExternalIP directory and collect zip files
for i in {1..5}; do
    ip_dir="${base_dir}/ExternalIP_${i}/MITM/data_zips"
    collect_zips "$ip_dir" "$i"
done

echo "All zip files have been collected into $all_zips_dir"
```

Data Collection:

```
#!/bin/bash

# Set the base logs directory and Ext2 directory
logs_dir="logs"
ext2_dir="Ext2"
zip_dir="data_zips"
```

```
timestamp=$(date +"%Y%m%d_%H%M%S")
zip_filename="logs_backup_$timestamp.zip"

# Create the data_zips directory if it doesn't exist
mkdir -p "$zip_dir"

# Create a zip file from all the logs in the specified subdirectories, including Ext2 if it exists
create_zip() {
    echo "Creating a zip file of all logs and Ext2 if present..."

    # Navigate to the logs directory
    cd "$logs_dir" || { echo "Failed to navigate to $logs_dir"; exit 1; }

    # Include Ext2 if it exists
    if [ -d "../$ext2_dir" ]; then
        zip -r "../$zip_filename" authentication_attempts/ keystrokes/ logins/ logouts/
session_streams/ "../$ext2_dir"
    else
        zip -r "../$zip_filename" authentication_attempts/ keystrokes/ logins/ logouts/
session_streams/
    fi

    # Check if the zip command succeeded
    if [ $? -eq 0 ]; then
```

```
        echo "Zip file created successfully: $zip_filename"
    else
        echo "Failed to create the zip file."
        exit 1
    fi

    # Navigate back to the original directory
    cd ..

    # Move the zip file to the data_zips directory
    mv "$zip_filename" "$zip_dir/"
    echo "Zip file moved to $zip_dir/$zip_filename"
}

# Delete all files inside the subdirectories
clean_directories() {
    echo "Deleting all files inside the subdirectories..."

    # Delete all files in the specified subdirectories
    rm -rf "$logs_dir"/authentication_attempts/* \
           "$logs_dir"/keystrokes/* \
           "$logs_dir"/logins/* \
           "$logs_dir"/logouts/* \
```

```
        "$logs_dir"/session_streams/*

    echo "All files inside the subdirectories have been deleted."
}

# Main function
main() {
    # Create the zip file
    create_zip

    # Clean up by deleting all files in the subdirectories
    clean_directories
}

# Run the main function
main
```

Recycling:

```
#!/bin/bash

# Function to print usage information
print_usage() {
    echo "Usage: $0 <base_container_name> <external_ip> <mitm_port>"
```

```
}


# Check if correct number of arguments are provided
if [ $# -ne 3 ]; then
    print_usage
    exit 1
fi


# Path to the log backup and cleanup script
directory="/FakeCompany"
log_script_path="./dataCollection2.sh"  # Adjust this path if needed


counter_file="IP_list.txt"  # File to store the current counter value# Load the counter value from the file if it exists
if [ -f "$counter_file" ]; then
    counter=$(<"$counter_file")  # Load the last saved counter value
else
    counter=1  # Start with counter at 1024 if no file exists
fi


cleanup() {
    echo "Cleaning up..."
    # Kill the background process
    if [[ -n "$preload_pid" ]]; then
```

```
        kill "$preload_pid" 2>/dev/null
    fi
    jobs -p | xargs -r kill  # Kill any other background jobs
    remove_existing
    sudo bash "$log_script_path"
    exit 0
}

trap cleanup SIGINT SIGTERM

# Assign arguments to variables
base_container_name=$1
external_ip=$2
mitm_port=$3
attacker_ip=""
curr_container_name=""
next_container_name=""

# Function to remove the container, networking rules, and MITM process
remove_existing() {

    echo "Removing existing container and rules for $curr_container_name"
```

```
    local current_container_ip=$(sudo lxc-info -n $curr_container_name -iH 2>/dev/null)

    # Clean up any existing rules for the attacker IP
    if [ -n "$attacker_ip" ]; then
        echo "Cleaning up iptables rules for attacker IP: $attacker_ip"
        sudo iptables -w --table nat --delete PREROUTING --source "$attacker_ip" --destination
"$external_ip" --jump DNAT --to-destination "$current_container_ip"
        sudo iptables -w --table nat --delete PREROUTING --source "$attacker_ip" --destination
"$external_ip" --protocol tcp --dport 22 --jump DNAT --to-destination "10.0.3.1:$mitm_port"
    fi

    if [ -n "$current_container_ip" ]; then
        sudo iptables -w -t nat -D PREROUTING -s 0.0.0.0/0 -d $external_ip -j DNAT
--to-destination $current_container_ip 2>/dev/null
        sudo iptables -w -t nat -D POSTROUTING -s $current_container_ip -d 0.0.0.0/0 -j SNAT
--to-source $external_ip 2>/dev/null
    fi

    sudo iptables -w -t nat -D PREROUTING -s 0.0.0.0/0 -d $external_ip -p tcp --dport 22 -j
DNAT --to-destination "10.0.3.1:$mitm_port" 2>/dev/null
    sudo ip addr del $external_ip/24 dev eth3 2>/dev/null

    if sudo pm2 list | grep -q "$curr_container_name"; then
        echo "Stopping MITM process for $curr_container_name"
        sudo pm2 stop "$curr_container_name"
```

```
        sudo pm2 delete "$curr_container_name"
    fi

    sudo lxc-stop -n $curr_container_name -k 2>/dev/null
    sudo lxc-destroy -n $curr_container_name 2>/dev/null

    echo "Existing container, rules, and MITM removed."

    # Update the counter value in IP_list.txt
    echo "$counter" > "$counter_file"

    echo "Counter value $counter saved to $counter_file."

}

start_mitm() {

    echo "Starting MITM for $curr_container_name"

    local current_container_ip=$(sudo lxc-info -n $curr_container_name -iH 2>/dev/null)

    if sudo pm2 start mitm.js --name "$curr_container_name" -- -n "$curr_container_name" -i
"$current_container_ip" -p $mitm_port --mitm-ip 10.0.3.1 --auto-access --auto-access-fixed 1
--debug; then
```

```
        echo "MITM server started successfully"
    else
        echo "Failed to start MITM server"
        exit 1
    fi
}

setup_networking() {

    local current_container_ip=$(sudo lxc-info -n $curr_container_name -iH 2>/dev/null)

    sudo ip link set eth3 up
    sudo ip addr add $external_ip/24 brd + dev eth3
    sudo iptables -w --table nat --insert PREROUTING --source 0.0.0.0/0 --destination $external_ip --jump DNAT --to-destination $current_container_ip
    sudo iptables -w --table nat --insert POSTROUTING --source $current_container_ip --destination 0.0.0.0/0 --jump SNAT --to-source $external_ip
    sudo iptables -w --table nat --insert PREROUTING --source 0.0.0.0/0 --destination $external_ip --protocol tcp --dport 22 --jump DNAT --to-destination "10.0.3.1:$mitm_port"
    sudo sysctl -w net.ipv4.ip_forward=1
}

# Function to start a recycling timer and repeat every 5 minutes
check_log_and_start_recycling_timer() {
```

```
log_file="logs/authentication_attempts/${curr_container_name}.log"
local current_container_ip=$(sudo lxc-info -n $curr_container_name -iH 2>/dev/null)

while true; do
  sleep 1

  # Break if a termination signal has been received
  if [[ ! -z "$terminate_flag" ]]; then
    echo "Termination signal received. Exiting log check."
    break
  fi

  if [ -f "$log_file" ]; then
    if [ -s "$log_file" ]; then
      echo "Log file $log_file exists and is non-empty. Starting recycling timer..."

      # Extract the attacker's IP from the second field in the log file
      attacker_ip=$(awk -F';' '{print $2}' "$log_file" | head -1)

      if [ -n "$attacker_ip" ]; then
        echo "Detected attacker IP: $attacker_ip"

        # Allow traffic from the attacker IP
```

```
            sudo iptables -w --table nat --insert PREROUTING --source $attacker_ip
--destination $external_ip --jump DNAT --to-destination $current_container_ip


            # Allow only the attacker to SSH on the MITM port

            sudo iptables -w --table nat --insert PREROUTING --source $attacker_ip
--destination $external_ip --protocol tcp --dport 22 --jump DNAT --to-destination
"10.0.3.1:$mitm_port"


            # Remove the rule that allows all connections (if it exists)

            sudo iptables -w --table nat --delete PREROUTING --destination $external_ip
--jump DNAT --to-destination $current_container_ip 2>/dev/null

            sudo iptables -w --table nat --delete PREROUTING --destination $external_ip
--protocol tcp --dport 22 --jump DNAT --to-destination "10.0.3.1:$mitm_port" 2>/dev/null


            echo "Recycling timer started, allowing only the attacker IP: $attacker_ip"

            start_recycling_timer

            break

          else

            echo "Could not detect attacker IP from the log file."

          fi

        else

          echo "Log file $log_file is empty. Continuing to check..."

        fi

      else

        echo "Log file $log_file does not exist yet. Continuing to check..."
```

```
        fi
    done
}

create_container() {

    local container_name=$1

    echo "Creating new container: $container_name"
    sudo lxc-create -t download -n "$container_name" -- -d ubuntu -r focal -a amd64
    sudo lxc-start -n "$container_name"
    sleep 10  # Wait for container to start
}

setup_firewall() {

    local container_name=$1

    echo "Setting up firewall for $container_name"
    sudo lxc-attach -n $container_name -- bash -c "apt update && apt install -y ufw"
    sudo lxc-attach -n $container_name -- bash -c "ufw allow 22/tcp && ufw --force enable"
}
```

```
setup_ssh() {

    local container_name=$1

    echo "Setting up SSH server $container_name"
    sudo lxc-attach -n $container_name -- bash -c "
        if ! dpkg -s openssh-server >/dev/null 2>&1; then
            apt-get update
            apt-get install -y openssh-server
        fi
    "
}

verify_ssh_setup() {

    local container_name=$1

    echo "Verifying SSH setup for $container_name"
    sudo lxc-attach -n $container_name -- bash -c "
        if dpkg -s openssh-server >/dev/null 2>&1; then
            echo 'SSH server is installed'
            if systemctl is-active --quiet ssh; then
                echo 'SSH service is running'
```

```
            else
                echo 'SSH service is not running'
                systemctl status ssh
            fi
        else
            echo 'SSH server is not installed'
        fi
    "
}

# Function to install Netdata inside the container
install_netdata() {

    local container_name=$1

    echo "Installing Netdata inside $container_name"
    #sudo lxc-attach -n $container_name -- bash -c "
        #wget -O /tmp/netdata-kickstart.sh https://get.netdata.cloud/kickstart.sh
        #yes | bash /tmp/netdata-kickstart.sh --stable-channel --claim-token
Xaxob7L5aS9Lml5L18KOGQoU0CSURRXNui_8vVWik6-QckMjI9wNj0izlvyVTg-7Bhb_hTb
yIsp9pGuQduJLnJztbcZDVfADH1Mi-OERljl09L42NafCGFn7iLQcoa3jACG2UDI
--claim-rooms 1bafb91a-89af-4b1c-a194-93ee725492aa --claim-url https://app.netdata.cloud
    #"
}
```

```
# Function to setup honey
setup_honey() {

  local scenario=$1
  local container_name=$2

  case $scenario in
  Weak)
    cat setup_honey_weak.sh | sudo lxc-attach -n $container_name -- bash
    echo "Weak scenario selected and setup complete."
    ;;
  Medium)
    cat setup_honey_med.sh | sudo lxc-attach -n $container_name -- bash
    echo "Medium scenario selected and setup complete."
    ;;
  High)
    cat setup_honey_high.sh | sudo lxc-attach -n $container_name -- bash
    echo "High scenario selected and setup complete."
    ;;
  *)
    cat setup_honey_none.sh | sudo lxc-attach -n $container_name -- bash
    echo "No specific scenario selected (None)."
```

```
      ;;
    esac
}



# Countdown function
countdown() {
    secs=$1
    log_file="logs/logouts/${curr_container_name}.log"

    while [ $secs -gt 0 ]; do
        if [ -f "$log_file" ]; then
            if [ -s "$log_file" ]; then
                echo "Attacker has logged out. Exiting countdown and recycling..."
                break
            fi
        fi

        echo -ne "Recycling in $secs seconds...\033[0K\r"
        sleep 1
        : $((secs--))
    done
}
```

```
dataCounter=1

increment_counter() {
    counter=$((counter + 1))  # Increment the counter
    dataCounter=$((dataCounter + 1))  # Increment dataCounter

    # Check if counter has reached 10
    if [ "$dataCounter" -eq 10003 ]; then
        echo "Counter reached 10. Saving logs, clearing directories, and resetting counter."
        sudo bash "$log_script_path"
        dataCounter=1  # Reset the counter
    fi

    curr_scenario=$(select_random_scenario)
}


# Function to randomly select a scenario
select_random_scenario() {
    scenarios=("None" "Weak" "Medium" "High")
    selected_scenario=${scenarios[$RANDOM % ${#scenarios[@]}]}
    echo $selected_scenario
```

```
}

# Preload the next container in the background while the current one is running
preload_next_container() {

    # Increment the counter to generate the next container name
    increment_counter
    next_container_name="${base_container_name}_${curr_scenario}_$counter"

    echo "Preloading the next container: $next_container_name"

    (
        create_container "$next_container_name"
        setup_firewall "$next_container_name"
        setup_ssh "$next_container_name"
        verify_ssh_setup "$next_container_name"
        install_netdata "$next_container_name"
        setup_honey "$curr_scenario" "$next_container_name"
    ) &  # Run preloading in the background
    preload_pid=$!
}

# Function to check if the "FakeCompany" directory exists
```

```
check_fake_company_directory () {

    while ! sudo lxc-attach -n "$next_container_name" -- test -d "$directory"; do
        echo "Directory $directory not found in $next_container_name, checking again..."
        sleep 5  # Wait for 5 seconds before checking again
    done
    echo "Directory $directory found in $next_container_name!"
}

# Function to start recycling the current container
start_recycling_timer() {

    echo "Starting the recycling process for $curr_container_name."
    countdown 300

    # Remove the current container and rules
    remove_existing

    # Check for "FakeCompany" directory in the current container
    check_fake_company_directory
    curr_container_name="$next_container_name"

    # Start MITM and networking after the directory is confirmed
```

```
    start_mitm
    setup_networking

    # Preload the next container for future use
    preload_next_container

    # Call the log checking function to continue monitoring
    check_log_and_start_recycling_timer
}

# Start main execution
curr_scenario=$(select_random_scenario)
curr_container_name="${base_container_name}_${curr_scenario}_$counter"
remove_existing
create_container "$curr_container_name"
setup_firewall "$curr_container_name"
setup_ssh "$curr_container_name"
verify_ssh_setup "$curr_container_name"
install_netdata "$curr_container_name"
setup_honey "$curr_scenario" "$curr_container_name"
start_mitm
setup_networking
```

```
echo "$curr_container_name is fully set up and waiting for an attack."


# Preload the next container in the background
preload_next_container


# Start monitoring logs and recycling containers based on activity
Check_log_and_start_recycling_timer
```

Honey None Setup:

```
#!/bin/bash


# Variables
BASE_DIR="/FakeCompany"  # Top-level directory name for the fake company
BASH_PROFILE_ENTRY="/etc/profile.d/protect_dir.sh"


# Define a date format for the file names
DATE=$(date +"%Y%m%d")


# Array of department names for directory levels
DEPARTMENTS=("HR" "Finance" "Marketing" "IT" "Sales" "Admin")


# Function to create nested directories and corporate files
create_nested_directories() {
   local current_dir="${BASE_DIR}"
```

```
    # Create the base directory
    mkdir -p "${BASE_DIR}"

    # Loop through the levels to create nested directories
    for department in "${DEPARTMENTS[@]}"; do
        current_dir="${current_dir}/${department}"  # Create subfolder for each department
        mkdir -p "${current_dir}"  # Create the current level directory
        chmod 755 "${current_dir}"  # Set permissions

        # Create files with corporate naming convention in the current level
        for j in {1..3}; do  # Create 3 files per level as an example
            # Corporate file naming convention:
DEPARTMENT_YYYYMMDD_DESCRIPTION.txt
            file_name="${department}_${DATE}_file_${j}.txt"
            touch "${current_dir}/${file_name}"  # Create the file
            echo "File created: ${current_dir}/${file_name}"
        done
    done
}

# Create the nested directory structure and files
echo "Creating nested directory structure and corporate files..."
create_nested_directories  # This will use the defined department structure
```

```
# Function to set up bash profile for directory protection
setup_bash_profile() {
    cat <<EOF > ${BASH_PROFILE_ENTRY}
# Function to change to the protected directory with a password prompt
cd_protected() {
    local target_dir="\$1"
    local input_password
    local correct_password_found=false  # Variable to track if any password matches

    # Define an associative array for passwords
    declare -A PASSWORDS
    PASSWORDS["${BASE_DIR}/HR"]="first"
    PASSWORDS["${BASE_DIR}/HR/Finance"]="second"
    PASSWORDS["${BASE_DIR}/HR/Finance/Marketing"]="third"
    PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT"]="fourth"
    PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT/Sales"]="fifth"
    PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT/Sales/Admin"]="sixth"

    # Convert relative paths to absolute paths using realpath
    target_dir=\$(realpath "\$target_dir" 2>/dev/null)

    # Allow access to the base directory without a password
```

```
if [[ "\$target_dir" == "${BASE_DIR}" ]]; then
    builtin cd "\$@"
    return
fi

# Check if the target directory is a substring of any protected directory using grep
for protected_dir in "\${!PASSWORDS[@]}"; do
    if echo "\$target_dir" | grep -q "\$protected_dir"; then
        echo "Enter password to access the directory:"
        read -s input_password

        # Loop through all passwords in the PASSWORDS array
        for dir_password in "\${PASSWORDS[@]}"; do
            if [[ "\$input_password" == "\$dir_password" ]]; then
                correct_password_found=true
                break
            fi
        done

        # If a correct password is found, change directory
        if \$correct_password_found; then
            builtin cd "\$@"
            return
```

```
            else
                # If the password is incorrect, deny access
                echo "Access denied."
                return 1
            fi
        fi
    done

    # Allow normal cd for non-protected directories
    builtin cd "\$@"
}

# Override the cd command to use the custom function
alias cd='cd_protected'
EOF

    # Make sure the script is sourced in each login shell
    chmod +x ${BASH_PROFILE_ENTRY}
}

# Set up the bash profile for directory protection

echo "Setup complete. A password is now required to access nested directories under
${BASE_DIR}, while the top-level directory is accessible without a password."
```

Honey Weak Setup:

```
#!/bin/bash

# Variables
BASE_DIR="/FakeCompany"  # Top-level directory name for the fake company
BASH_PROFILE_ENTRY="/etc/profile.d/protect_dir.sh"

# Define a date format for the file names
DATE=$(date +"%Y%m%d")

# Array of department names for directory levels
DEPARTMENTS=("HR" "Finance" "Marketing" "IT" "Sales" "Admin")

# Function to create nested directories and corporate files
create_nested_directories() {
    local current_dir="${BASE_DIR}"

    # Create the base directory
    mkdir -p "${BASE_DIR}"

    # Loop through the levels to create nested directories
    for department in "${DEPARTMENTS[@]}"; do
        current_dir="${current_dir}/${department}"  # Create subfolder for each department
```

```
        mkdir -p "${current_dir}"  # Create the current level directory
        chmod 755 "${current_dir}"  # Set permissions

        # Create files with corporate naming convention in the current level
        for j in {1..3}; do  # Create 3 files per level as an example
            # Corporate file naming convention:
DEPARTMENT_YYYYMMDD_DESCRIPTION.txt
            file_name="${department}_${DATE}_file_${j}.txt"
            touch "${current_dir}/${file_name}"  # Create the file
            echo "File created: ${current_dir}/${file_name}"
        done
    done
}

# Create the nested directory structure and files
echo "Creating nested directory structure and corporate files..."
create_nested_directories  # This will use the defined department structure

# Function to set up bash profile for directory protection
setup_bash_profile() {
    cat <<EOF > ${BASH_PROFILE_ENTRY}
# Function to change to the protected directory with a password prompt
cd_protected() {
    local target_dir="\$1"
```

```
local input_password
local correct_password_found=false  # Variable to track if any password matches

# Define an associative array for passwords
declare -A PASSWORDS
PASSWORDS["${BASE_DIR}/HR"]="abcd"
PASSWORDS["${BASE_DIR}/HR/Finance"]="1234"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing"]="password"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT"]="default"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT/Sales"]="admin"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT/Sales/Admin"]="test"

# Convert relative paths to absolute paths using realpath
target_dir=\$(realpath "\$target_dir" 2>/dev/null)

# Allow access to the base directory without a password
if [[ "\$target_dir" == "${BASE_DIR}" ]]; then
   builtin cd "\$@"
   return
fi

# Check if the target directory is a substring of any protected directory using grep
for protected_dir in "\${!PASSWORDS[@]}"; do
```

```
  if echo "\$target_dir" | grep -q "\$protected_dir"; then
    echo "Enter password to access the directory:"
    read -s input_password

    # Loop through all passwords in the PASSWORDS array
    for dir_password in "\${PASSWORDS[@]}"; do
      if [[ "\$input_password" == "\$dir_password" ]]; then
        correct_password_found=true
        break
      fi
    done

    # If a correct password is found, change directory
    if \$correct_password_found; then
      builtin cd "\$@"
      return
    else
      # If the password is incorrect, deny access
      echo "Access denied."
      return 1
    fi
  fi
done
```

```
    # Allow normal cd for non-protected directories
    builtin cd "\$@"
}

# Override the cd command to use the custom function
alias cd='cd_protected'
EOF

    # Make sure the script is sourced in each login shell
    chmod +x ${BASH_PROFILE_ENTRY}
}

# Set up the bash profile for directory protection
echo "Setting up bash profile for directory protection..."
setup_bash_profile

echo "Setup complete. A password is now required to access nested directories under ${BASE_DIR}, while the top-level directory is accessible without a password."
```

Honey Medium Setup:

```
#!/bin/bash

# Variables
```

```
BASE_DIR="/FakeCompany"  # Top-level directory name for the fake company
BASH_PROFILE_ENTRY="/etc/profile.d/protect_dir.sh"

# Define a date format for the file names
DATE=$(date +"%Y%m%d")

# Array of department names for directory levels
DEPARTMENTS=("HR" "Finance" "Marketing" "IT" "Sales" "Admin")

# Function to create nested directories and corporate files
create_nested_directories() {
    local current_dir="${BASE_DIR}"

    # Create the base directory
    mkdir -p "${BASE_DIR}"

    # Loop through the levels to create nested directories
    for department in "${DEPARTMENTS[@]}"; do
        current_dir="${current_dir}/${department}"  # Create subfolder for each department
        mkdir -p "${current_dir}"  # Create the current level directory
        chmod 755 "${current_dir}"  # Set permissions

        # Create files with corporate naming convention in the current level
```

```
        for j in {1..3}; do  # Create 3 files per level as an example
            # Corporate file naming convention:
DEPARTMENT_YYYYMMDD_DESCRIPTION.txt
            file_name="${department}_${DATE}_file_${j}.txt"
            touch "${current_dir}/${file_name}"  # Create the file
            echo "File created: ${current_dir}/${file_name}"
        done
    done
}

# Create the nested directory structure and files
echo "Creating nested directory structure and corporate files..."
create_nested_directories  # This will use the defined department structure

# Function to set up bash profile for directory protection
setup_bash_profile() {
    cat <<EOF > ${BASH_PROFILE_ENTRY}
# Function to change to the protected directory with a password prompt
cd_protected() {
    local target_dir="\$1"
    local input_password
    local correct_password_found=false  # Variable to track if any password matches

    # Define an associative array for passwords
```

```
declare -A PASSWORDS
PASSWORDS["${BASE_DIR}/HR"]="pass123"
PASSWORDS["${BASE_DIR}/HR/Finance"]="jan2724"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing"]="abc123"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT"]="xyz_abc"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT/Sales"]="doremi"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT/Sales/Admin"]="difam246"

# Convert relative paths to absolute paths using realpath
target_dir=\$(realpath "\$target_dir" 2>/dev/null)

# Allow access to the base directory without a password
if [[ "\$target_dir" == "${BASE_DIR}" ]]; then
    builtin cd "\$@"
    return
fi

# Check if the target directory is a substring of any protected directory using grep
for protected_dir in "\${!PASSWORDS[@]}"; do
    if echo "\$target_dir" | grep -q "\$protected_dir"; then
        echo "Enter password to access the directory:"
        read -s input_password
```

```
        # Loop through all passwords in the PASSWORDS array
        for dir_password in "\${PASSWORDS[@]}"; do
          if [[ "\$input_password" == "\$dir_password" ]]; then
            correct_password_found=true
            break
          fi
        done

        # If a correct password is found, change directory
        if \$correct_password_found; then
          builtin cd "\$@"
          return
        else
          # If the password is incorrect, deny access
          echo "Access denied."
          return 1
        fi
      fi
    done

    # Allow normal cd for non-protected directories
    builtin cd "\$@"
}
```

```
# Override the cd command to use the custom function
alias cd='cd_protected'
EOF

    # Make sure the script is sourced in each login shell
    chmod +x ${BASH_PROFILE_ENTRY}
}

# Set up the bash profile for directory protection
echo "Setting up bash profile for directory protection..."
setup_bash_profile

echo "Setup complete. A password is now required to access nested directories under ${BASE_DIR}, while the top-level directory is accessible without a password."
```

Honey Strong Setup:

```
#!/bin/bash

# Variables
BASE_DIR="/FakeCompany"  # Top-level directory name for the fake company
BASH_PROFILE_ENTRY="/etc/profile.d/protect_dir.sh"

# Define a date format for the file names
DATE=$(date +"%Y%m%d")
```

```
# Array of department names for directory levels
DEPARTMENTS=("HR" "Finance" "Marketing" "IT" "Sales" "Admin")

# Function to create nested directories and corporate files
create_nested_directories() {
  local current_dir="${BASE_DIR}"

  # Create the base directory
  mkdir -p "${BASE_DIR}"

  # Loop through the levels to create nested directories
  for department in "${DEPARTMENTS[@]}"; do
    current_dir="${current_dir}/${department}"  # Create subfolder for each department
    mkdir -p "${current_dir}"  # Create the current level directory
    chmod 755 "${current_dir}"  # Set permissions

    # Create files with corporate naming convention in the current level
    for j in {1..3}; do  # Create 3 files per level as an example
      # Corporate file naming convention:
DEPARTMENT_YYYYMMDD_DESCRIPTION.txt
      file_name="${department}_${DATE}_file_${j}.txt"
      touch "${current_dir}/${file_name}"  # Create the file
      echo "File created: ${current_dir}/${file_name}"
```

```
        done
    done
}

# Create the nested directory structure and files
echo "Creating nested directory structure and corporate files..."
create_nested_directories  # This will use the defined department structure

# Function to set up bash profile for directory protection
setup_bash_profile() {
    cat <<EOF > ${BASH_PROFILE_ENTRY}
# Function to change to the protected directory with a password prompt
cd_protected() {
    local target_dir="\$1"
    local input_password
    local correct_password_found=false  # Variable to track if any password matches

    # Define an associative array for passwords
    declare -A PASSWORDS
    PASSWORDS["${BASE_DIR}/HR"]="abcd"
    PASSWORDS["${BASE_DIR}/HR/Finance"]="1234"
    PASSWORDS["${BASE_DIR}/HR/Finance/Marketing"]="password"
    PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT"]="default"
```

```
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT/Sales"]="admin"
PASSWORDS["${BASE_DIR}/HR/Finance/Marketing/IT/Sales/Admin"]="test"

# Convert relative paths to absolute paths using realpath
target_dir=\$(realpath "\$target_dir" 2>/dev/null)

# Allow access to the base directory without a password
if [[ "\$target_dir" == "${BASE_DIR}" ]]; then
  builtin cd "\$@"
  return
fi

# Check if the target directory is a substring of any protected directory using grep
for protected_dir in "\${!PASSWORDS[@]}"; do
  if echo "\$target_dir" | grep -q "\$protected_dir"; then
    echo "Enter password to access the directory:"
    read -s input_password

    # Loop through all passwords in the PASSWORDS array
    for dir_password in "\${PASSWORDS[@]}"; do
      if [[ "\$input_password" == "\$dir_password" ]]; then
        correct_password_found=true
        break
```

```
            fi
        done

        # If a correct password is found, change directory
        if \$correct_password_found; then
            builtin cd "\$@"
            return
        else
            # If the password is incorrect, deny access
            echo "Access denied."
            return 1
        fi
    fi
  done

  # Allow normal cd for non-protected directories
  builtin cd "\$@"
}

# Override the cd command to use the custom function
alias cd='cd_protected'
EOF
```

```
    # Make sure the script is sourced in each login shell
    chmod +x ${BASH_PROFILE_ENTRY}
}

# Set up the bash profile for directory protection
echo "Setting up bash profile for directory protection..."
setup_bash_profile

echo "Setup complete. A password is now required to access nested directories under
${BASE_DIR}, while the top-level directory is accessible without a password."
```

Reboot:

```
#!/bin/bash

# Function to clear IP rules
clear_ip_rules() {
    echo "Clearing IP rules..."
    sudo bash /home/clearIPRules.sh
}
```

```
# Function to apply required firewall rules

apply_firewall_rules() {

    echo "Applying required firewall rules..."

    sudo modprobe br_netfilter

    sudo sysctl -p /etc/sysctl.conf

    sudo bash /home/required_firewall_rules.sh

}



stop_and_delete_containers() {

    # Get the list of all LXC containers (both running and stopped)

containers=$(sudo lxc-ls -f | awk 'NR>1 {print $1}')



# Loop through each container

  for container in $containers; do

      echo "Stopping container: $container"

      sudo lxc-stop -n $container 2>/dev/null  # Stop the container (ignore error if already stopped)
```

```
      echo "Deleting container: $container"

      sudo lxc-destroy -n $container  # Delete the container

  done



  echo "All containers have been deleted."

}



clear_pm2_processes() {

    # Stop all PM2 processes

    sudo pm2 stop all



    # Delete all PM2 processes from the list

    sudo pm2 delete all



    # Optionally, you can also save the PM2 list to reset the process list completely

    sudo pm2 save

}
```

```
kill_recycle_processes() {
    # Get all process IDs related to recycle2Final.sh except for the grep command itself
    pids=$(ps aux | grep 'recycleFinal.sh' | grep -v grep | awk '{print $2}')


    if [ -z "$pids" ]; then
        echo "No processes found for recycle2Final.sh"
    else
        echo "Killing the following processes related to recycle2Final.sh:"
        echo "$pids"


        # Kill all the found processes
        for pid in $pids; do
            sudo kill "$pid"
            echo "Killed process with PID: $pid"
        done
```

```
        # Verify if any processes are still running

        remaining_pids=$(ps aux | grep 'recycleFinal.sh' | grep -v grep | awk '{print $2}')


        if [ -z "$remaining_pids" ]; then

            echo "All processes related to recycle2Final.sh have been successfully terminated."

        else

            echo "Some processes could not be killed: $remaining_pids"

            echo "You may need to use kill -9 for these processes."

        fi

    fi

}


kill_recycle_processe

# You can call the function like this:

clear_pm2_processes


# Main execution starts here
```

```
stop_and_delete_containers

clear_ip_rules

apply_firewall_rules


echo "Start?"

sleep 5


# Honeypot Start

cd /home/ExternalIP_1/MITM || exit 1

nohup sudo bash recycleFinal.sh honeypot1 128.8.238.199 8001 &

disown


cd /home/ExternalIP_2/MITM || exit 1

nohup sudo bash recycleFinal.sh honeypot2 128.8.238.39 8003 &

disown


cd /home/ExternalIP_3/MITM || exit 1
```

```
nohup sudo bash recycleFinal.sh honeypot3 128.8.238.56 8002 &

disown



cd /home/ExternalIP_4/MITM || exit 1

nohup sudo bash recycleFinal.sh honeypot4 128.8.238.115 8008 &

disown



cd /home/ExternalIP_5/MITM || exit 1

nohup sudo bash recycleFinal.sh honeypot5 128.8.238.116 8005 &

disown




echo "Reboot script execution completed."
```

Required Firewall Rules:

```
#!/bin/sh -e

#

# "Super fancy Firewall"
```

```
# Division of IT

#

#

# To enable the firewall, you may need to enable the br_netfilter kernel module

# by running the following commands:

# modprobe br_netfilter

# sysctl -p /etc/sysctl.conf

#



#pve-firewall restart



##

# Reset the firewall

/sbin/iptables -F

/sbin/iptables -X

#/sbin/iptables -t nat -F

#/sbin/iptables -t nat -X
```

```
/sbin/iptables -t mangle -F

/sbin/iptables -t mangle -X

/sbin/iptables -P INPUT ACCEPT

/sbin/iptables -P FORWARD ACCEPT

/sbin/iptables -P OUTPUT ACCEPT



sysctl -w net.bridge.bridge-nf-call-iptables=1



##

# Firewall Mode

##

# Mode 1: Allow all traffic to the Honeypots

# Mode 2: Allow only the listed port (hp_tcp, hp_udp)

# Mode 3: Block the Honeypots

MODE=1
```

```
# NOTE: MITM should listen on the $CONTAINER_GATEWAY IP, other IPs will get blocked
by this firewall


##
# Container network settings
##
# Update this if your container IP address and network is different
#
CONTAINER_NETWORK="10.0.3.1/24"
CONTAINER_GATEWAY="10.0.3.1"
CONTAINER_INTERFACE="lxcbr0"


##
# Rate Limiting Logging
##
# 0: No logging (default)
```

```
# 1: All the traffic dropped because of the rate limiting rules is logged in Syslog (can generate a lot of logs!)

LOG=0



##

# MODE=2: Ports to Open on the Honeypots

hp_tcp='22'

hp_udp=''



##

# Ports to open on the Host

host_tcp='22'

host_udp=''



########### DO NOT CHANGE ###############

trusted_ip='172.30.0.0/16 10.255.0.0/16 192.168.11.0/24'

#########################################
```

```
# Default policy

/sbin/iptables -F INPUT

/sbin/iptables -P INPUT DROP

/sbin/iptables -F FORWARD

/sbin/iptables -P FORWARD DROP

/sbin/iptables -A FORWARD -s 0.0.0.0/0.0.0.0 -d 0.0.0.0/0.0.0.0 -m state --state INVALID -j DROP -m comment --comment "Drop invalid connections"

/sbin/iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT -m comment --comment "Allow existing connections"

/sbin/iptables -F OUTPUT

/sbin/iptables -P OUTPUT ACCEPT



####################

## Host ##

####################
```

```
# Allow lxc-net service

/sbin/iptables -A INPUT -i $CONTAINER_INTERFACE -p tcp -m tcp --dport 53 -j ACCEPT
-m comment --comment "Container Network DNS TCP"

/sbin/iptables -A INPUT -i $CONTAINER_INTERFACE -p udp -m udp --dport 53 -j ACCEPT
-m comment --comment "Container Network DNS UDP"

/sbin/iptables -A INPUT -i $CONTAINER_INTERFACE -p tcp -m tcp --dport 67 -j ACCEPT
-m comment --comment "Container Network DHCP TCP"

/sbin/iptables -A INPUT -i $CONTAINER_INTERFACE -p udp -m udp --dport 67 -j ACCEPT
-m comment --comment "Container Network DHCP UDP"

/sbin/iptables -A INPUT -i lo -j ACCEPT -m comment --comment "Allow local loopback"


# Allow TCP port listed in host_tcp

for i in $host_tcp;

do

    for ip in $trusted_ip;

    do

        /sbin/iptables -A INPUT -s "$ip" -p tcp --dport $i -m state --state NEW -j ACCEPT
```

```
    done
done

# Allow UDP port listed in host_udp
for i in $host_udp;
do
    for ip in $trusted_ip;
    do
        /sbin/iptables -A INPUT -s "$ip" -p udp -m udp --dport $i -j ACCEPT
    done
done

# Allow connections to the host on the private ip $CONTAINER_GATEWAY (for the MITM)
/sbin/iptables -A INPUT -d $CONTAINER_GATEWAY -p tcp ! --dport 22 -j ACCEPT -m
comment --comment "Allow connections to the host for MITM"
/sbin/iptables -A INPUT -d 127.0.0.1 -p tcp ! --dport 22 -j ACCEPT -m comment --comment
"Allow connections to localhost for MITM"
```

```
# Allow related/established connections

/sbin/iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT -m comment
--comment "Allow related/established connections"


###############

## Honeypots ##

###############



####

## Honeypot Incoming Traffic

####



############### HERE is a good place to block incoming/outgoing traffic
###################
```

```
#                                                                #
# To block some traffic for one honeypot, use the -d <Honeypot Private IP> parameter      #
# To block some Internet IP, use the -s <Attacker Public IP> parameter                   #
#                                                                #
# for example:                                                         #
# /sbin/iptables -A FORWARD -i lxcbr0 -d 172.20.0.2 -p tcp --dport 22 -j DROP             #
#    will block SSH traffic to 172.20.0.2)                                        #
#                                                                #
# /sbin/iptables -A FORWARD -i lxcbr0 -s 8.8.8.8 -d 172.20.0.2 -p tcp --dport 22 -j DROP   #
#    will block SSH traffic to 172.20.0.2 coming from 8.8.8.8 only)                     #
##############################################################################################


# Block container to container communication
/sbin/iptables -A FORWARD -i $CONTAINER_INTERFACE -o $CONTAINER_INTERFACE
-s $CONTAINER_GATEWAY -d $CONTAINER_NETWORK -j ACCEPT -m comment
--comment "Accept connection from host to honeypots"
```

```
/sbin/iptables -A FORWARD -i $CONTAINER_INTERFACE -o $CONTAINER_INTERFACE
-s $CONTAINER_NETWORK -d $CONTAINER_GATEWAY -j ACCEPT -m comment
--comment "Accept connection from honeypots to host"

/sbin/iptables -A FORWARD -i $CONTAINER_INTERFACE -o $CONTAINER_INTERFACE
-s $CONTAINER_NETWORK -d $CONTAINER_NETWORK -j DROP -m comment
--comment "Drop connection between honeypots"



# Forward container traffic for lxc-net

/sbin/iptables -A FORWARD -o $CONTAINER_INTERFACE -j ACCEPT -m comment
--comment "Forward Container traffic"

/sbin/iptables -A FORWARD -i $CONTAINER_INTERFACE -j ACCEPT -m comment
--comment "Forward Container traffic"



# MODE 1: Allow everything on $CONTAINER_INTERFACE (to the Honeypots Containers)

if [ "$MODE" -eq 1 ]; then

echo "DEBUG: Firewall MODE 1"

/sbin/iptables -A FORWARD -d $CONTAINER_NETWORK -j ACCEPT -m comment
--comment "Allow connections to the honeypots"

fi
```

```
# MODE 2: Allow only certain ports
if [ "$MODE" -eq 2 ]; then
echo "DEBUG: Firewall MODE 2"
for i in $hp_tcp;
do
	/sbin/iptables -A FORWARD -d $CONTAINER_NETWORK -p tcp --dport $i -m state
--state NEW -j ACCEPT
done

for i in $hp_udp;
do
	/sbin/iptables -A FORWARD -p udp -d $CONTAINER_NETWORK -m udp --dport $i -j
ACCEPT
done

fi
```

```
if [ "$MODE" -eq 3 ]; then

    echo "DEBUG: Firewall MODE 3"

    # Default policy drops all @ FORWARD

    exit 0

fi



# Allow Ping

/sbin/iptables -A FORWARD -p icmp -m icmp --icmp-type any -j ACCEPT -m comment
--comment "Allow ICMP (Ping)"



####

## Rate Limiting

####



# Create a Table udp_flood in iptables (table of actions)

/sbin/iptables -N udp_flood
```

```
/sbin/iptables -A udp_flood -m hashlimit --hashlimit-name UDP_FLOOD --hashlimit-mode srcip
--hashlimit-srcmask 32 --hashlimit-upto 60/minute --hashlimit-burst 10 -j RETURN

if [ "$LOG" -eq 1 ]; then
    /sbin/iptables -A udp_flood -j LOG --log-level info --log-prefix "[FW] Rate Limit Reached: "
fi

/sbin/iptables -A udp_flood -j DROP

# Create a Table syn_flood in iptables (table of actions)
/sbin/iptables -N tcp_flood
/sbin/iptables -A tcp_flood -m hashlimit --hashlimit-name TCP_FLOOD --hashlimit-mode srcip
--hashlimit-srcmask 32 --hashlimit-upto 60/minute --hashlimit-burst 10 -m state --state NEW -j
RETURN # Cannot have more than 60 new connections per minute, burst is 10
# A container is not allowed to have more than 512kbytes/second of bandwidth
#/sbin/iptables -A tcp_flood -m hashlimit --hashlimit-name TCP_BANDWIDTH
--hashlimit-mode srcip --hashlimit-srcmask 32 --hashlimit-above 8/sec --hashlimit-burst 8 -j
DROP
```

```
if [ "$LOG" -eq 1 ]; then
    /sbin/iptables -A tcp_flood -j LOG --log-level info --log-prefix "[FW] Rate Limit Reached: "
fi

/sbin/iptables -A tcp_flood -j DROP

# Traffic matching UDP/TCP flood goes to the table
/sbin/iptables -I FORWARD 3 -s $CONTAINER_NETWORK -p udp -j udp_flood
/sbin/iptables -I FORWARD 3 -s $CONTAINER_NETWORK -p tcp -j tcp_flood

####
## Outgoing Traffic
###

# Allow all other HP outgoing traffic
```

```
/sbin/iptables -A FORWARD -s $CONTAINER_NETWORK -j ACCEPT -m comment
--comment "Allow all other honeypot outgoing"


exit 0
```

Works Cited / References

Almeshekah, Mohammed H., and Eugene H. Spafford. “Planning and integrating deception into computer security defenses.” *Proceedings of the 2014 New Security Paradigms Workshop*, 15 Sept. 2014, https://doi.org/10.1145/2683467.2683482.

Cremonini, Marco, and Dmitri Nizovtsev. "Understanding and influencing attackers’ decisions: Implications for security investment strategies." School of Bussiness: working paper 68 (2006).

Nicomette, V., Kaâniche, M., Alata, E. et al. Set-up and deployment of a high-interaction honeypot: experiment and lessons learned. J Comput Virol 7, 143–157 (2011). https://doi.org/10.1007/s11416-010-0144-2

“What Is a Honeypot? Meaning, Types, Benefits, and More.” *Fortinet*, www.fortinet.com/resources/cyberglossary/what-is-honeypot. Accessed 16 Sept. 2024.