The Effect of Web Page Content on SSH Attacker Behavior

Group 1C: FBI (Friendly Besties Incorporated)

Damian Figueroa, Annika Meng, Svetlana Semenova, Alina Sharma, Annabel Yang

HACS200 — 0101

1. Executive Summary

This project aimed to explore the connection between attacker behavior on an SSH server and the content of a web page hosted on the same system. We investigated how the presence of a web server and differing static web page content may influence the number of commands per attacker and invasiveness of commands run during these attacks. We hypothesized that the presence of a web server would increase the rate of SSH attacks and inspire different commands, and the rate of attacks and the types of commands run would vary based on the web page content.

Our experiment design consisted of containers with varying web page designs, which included the following four: a blank page, an informational page, a password change page, and an IOT device page. We also had a control experiment with no web server on it. Using a Man-in-the-Middle (MITM) server, we intercepted SSH traffic on each container and logged attacker behavior in honeypots. We also logged HTTP requests to filter SSH attackers to only those that had sent a GET request to their respective container's website before entering through SSH, which helped us determine whether the web page design affected the rate and invasiveness of attacker commands.

We collected data from eleven honeypots (three controls with no web server and two repeats of each web page design) in order to evaluate whether (1) a web server affects rate of commands per attacker IP (i.e. attack rate), (2) a web server affects command invasiveness, (3) if differing web page content affects attack rate, and (4) if differing web page content affects command invasiveness. Overall, we found that there exists a statistically significant relationship between the presence of a web server and command frequency and invasiveness, as well as differing web page content and frequency of attack rate/command invasiveness. However, these significant results may also be influenced by some amount of IP bias and randomized spam attacks. In future endeavors, we would attempt to account for both IP bias and spam attacks to form a more definitive conclusion.

2. Background Research

Web applications are made with an enormous variety whose technical designs are unique to the services they provide, so the attacks on them tend to be quite varied as well. Thus, research exploring how web applications influence attacker behavior has been prevalent throughout the years. For example, Vasek et. al. showed that web servers like Apache and Nginx are more likely to be attacked.[1] Other studies observe attacker behavior after purposefully introducing known vulnerabilities, such as Canali & Balzarotti[2] and Dandy et al.[3]

Another attractive target is a different point of entry into a computer system: SSH. Studies have shown that automated attacks from bots first tend to scan to find open ports, and then brute-force username and password combinations to get into an SSH server.[4] Then, once access to the SSH server is gained, human attackers or more sophisticated bots tend to enter the system and execute commands (e.g. running informational commands, edit/delete files, and download malicious programs via `wget`)1.[5] [6]

However, web page attacks and SSH attacks are not isolated from one another. For example, attackers have been shown to scan for both SSH servers and web servers.[7] Furthermore, web pages and their content can inspire SSH attacks: in 2017, Wordfence identified a mass-scanning of web pages' content to search for private SSH keys that were unintentionally exposed on those web pages, followed by breaking into SSH servers using those private keys.[8]

To our knowledge, the only experiment that has looked at this specific connection between web

---

[1] Marie Vasek, John Wadleigh, and Tyler Moore, "Hacking Is Not Random: A Case-Control Study of Webserver-Compromise Risk," *IEEE Transactions on Dependable and Secure Computing* 13, no. 2 (January 2016): pp. 206-219, https://doi.org/10.1109/tdsc.2015.2427847.

[2] Davide Canali and Davide Balzarotti, "Behind the Scenes of Online Attacks: an Analysis of Exploitation Behaviors on the Web," *20th Annual Network & Distributed System Security Symposium (NDSS 2013)*, February 2013, https://doi.org/hal-00799082.

[3] Dandy Kalma Rahmatullah, Suiya Michrandi Nasution, and Fairuz Azmi, "Implementation of Low Interaction Web Server Honeypot Using Cubieboard," 2016 *International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, January 16, 2016, https://doi.org/10.1109/iccerec.2016.7814970.

[4] Vincent Nicomette et al., "Set-up and Deployment of a High-Interaction Honeypot: Experiment and Lessons Learned," *Journal in Computer Virology* 7, no. 2 (2010): pp. 143-157, https://doi.org/10.1007/s11416-010-0144-2.

[5] Ibid.

[6] Sayyed Mehdi Amin et al., "An Experimental Study of SSH Attacks by Using Honeypot Decoys," *Indian Journal of Science and Technology* 6, no. 12 (2013): pp. 1-12, https://doi.org/10.17485/ijst/2013/v6i12.8.

[7] Solomon Z Melese and P.S. Avadhani, "Honeypot System for Attacks on SSH Protocol," *International Journal of Computer Network and Information Security* 8, no. 9 (August 2016): pp. 19-26, https://doi.org/10.5815/ijcnis.2016.09.03.

[8] Mark Maunder, "New Attacker Scanning for SSH Private Keys on Websites," Wordfence, October 18, 2017, https://www.wordfence.com/blog/2017/10/ssh-key-website-scans/.

page content and SSH attacks is a small, limited study done by SSH following the 2017 attacks.[9] We have also been unable to find any research which studies how actual web page content shapes attacker behavior — only how web servers shape attacker behavior. Thus, our study attempts to analyze this unexplored niche to see if static web page content influences attacker behavior on an open SSH server.

3. Experiment Design Changes

Our initial design had three honeypots with Apache web servers and a control honeypot without a web server. The honeypots with web servers had three different web pages: an informational site, a password changing site, and an IOT management site, all with branding from a fictitious "Department of Information Security at UMD." These would have been hosted on ports 80 and 443 for HTTP and HTTPS connectivity. We planned to use Apache's native logging system and the ACES MITM server for collecting SSH activity. We would consider an attacker's behavior as potentially influenced by the website design if they had sent a GET request prior to their SSH session, as that means that they would have seen the web page before compromising the honeypot. The containers would be recycled as soon as an attacker finished their SSH session.

Due to having one more IP than expected at the start of the project, we implemented the rest of the project along with a new honeypot that had a blank web page. This served as a basis for comparison (a web page with no content) so that we could easily analyze whether the web page content in other honeypots were having an effect on attack rate/command invasiveness. Around the same time, we also decided to only serve our websites on port 80. Since port 443 is dedicated for the HTTPS protocol, using this port means that our web servers would be expected to handle HTTPS connections. To do so would have required paid certificates, and this was not feasible given the resources and timeframe of our project.

Since our project aimed to analyze attacker behavior after they looked at a honeypot's web page, we decided not to randomize IPs. This allowed attackers who looked at a container's web page to come later and SSH into the associated honeypot. Because we did not randomize IPs, we sought to reduce any

[9] "Hackers Are Now Scanning for SSH Keys to Exploit," PAM solutions, Key Management Systems, Secure File Transfers (SSH Communications Security, March 25, 2022), https://www.ssh.com/blog/ssh-key-scan-attack-honeypot.

IP bias by requesting four extra IPs to duplicate all of our website experiments a few days prior to deployment. Additionally, near the beginning of November, we received two more IPs to duplicate our control honeypot to reduce IP bias in our control group.

In the first week after deployment, we noticed that some attackers were idling in our honeypots, which stalled the experiment for other valid attackers, since auto access would not be re-enabled until after the attacker disconnected. Thus, we shortened the session time limit so that the server would disconnect an attacker after an hour. We decided this would not create a significant impact on our experiment, since our SSH servers already had timeout features. During this time, we also realized that even if auto access had already been enabled in a session, multiple attackers could simultaneously connect to the container if they used the same credentials. Therefore, we modified the MITM code to reject any additional SSH sessions while an attacker was within the container.

A few weeks later, we discovered that attackers regularly utilized non-interactive SSH sessions to poll information about the system. Because we initially recycled after every attacker session, this caused the host's CPU utilization to continuously be around 100%. To deal with this, we first requested more CPU and memory resources to make the recycling process faster. Secondly, we decided to stop recycling containers after what we determined to be "clean" sessions, of which consisted solely of non-interactive invocations of the `uname` command. Since the output of `uname` was consistent throughout all of our containers (see Section 5), we modified the MITM to instead give the attacker output from a manually-curated "command cache file" rather than executing the command on the container. In this way, it would not be necessary to recycle the container since it was never modified; thus, we could greatly reduce the load on our system by only restarting the MITM server to reset auto-access privileges. Lastly, we implemented a random delay after container deletion to stagger the recreations of containers. These changes were all done within the first two weeks of October.

However, due to the frequent and time-consuming recycling of the containers, the effective uptime of our honeypot's websites was quite short. To maintain our websites while containers recycled, we created "shadow Apache servers" that would temporarily serve an offline container's website on its

public IP address until it came back online. We decided this would not cause any issues with the experiment — the websites were static and the servers sent out identical responses between shadow and container websites; therefore, an attacker would not perceive any change in the website's behavior.

The rest of the design stayed the same. Implementation details are documented in ***Section 5***.

4. Research Question and Hypothesis

Our research question is the following: Does the presence of a web server and content of a static web page influence the frequency and invasiveness of commands run during an attack? We investigated this question through two measures of attacker interactions: the number of commands ran by each IP address, as well as how invasive the commands run were.

To justify our alternative hypotheses, refer to ***Section 2***, in which we noted that web page content and SSH content have been shown to be linked before. Thus, it is reasonable to hypothesize that the presence of a web server could make attackers think that vulnerable information is on the web and SSH server, influencing attacker commands and attack rate. Similarly, it is logical to hypothesize that a web page's content may influence attacker commands and attack rate on a SSH server because an attacker may believe that the web page has sensitive information (for instance, information that exposes vulnerabilities in the SSH server, as in the 2017 SSH key attack[10]).

Thus, we present four alternate hypotheses: H1: the presence of a web server will promote more commands per attack (i.e. "attack rate"), H2: the presence of a web server will promote different levels of command invasiveness, H3: varying web page content will promote different rates of commands per attack, and H4: varying web page content will promote different levels of command invasiveness.

Given our four alternative sub-hypotheses, our null hypothesis is that differing web page content and the presence of a web server has no statistically significant relationship with attack rates and the invasiveness of attacker commands.

Within the context of our research question, we define "command invasiveness" as the degree by

[10] Mark Maunder, "New Attacker Scanning for SSH Private Keys on Websites," Wordfence, October 18, 2017, https://www.wordfence.com/blog/2017/10/ssh-key-website-scans/.

which the command run impacts the honeypot; for instance, a command such as `uname` would not change the honeypot at all and is therefore a low-level invasive command. For more details about command invasiveness and how it was categorized, see ***Figure B7*** in which commands are categorized into levels of invasiveness based on keywords. We also define "attacker" as any unauthorized user — identified by IP — who attempts to log into, run commands on, and interact with the SSH server, and "web page content" as what a user would see if they connected to the web page using our IPs.

5. Experiment Design

Our host virtual machine's main specifications included a total of 4 CPU cores, 32 GB of memory, and 130 GB of storage to host our containers and experiment logs.

Our firewall rules closely followed the course-provided rules, which we modified to only allow TCP port 80 for HTTP traffic and TCP port 22 for SSH traffic. The external IPs assigned to us were added onto our public facing network interface and were not removed for the duration of the experiment.

We created five container templates. All had the same honey in the form of a PostgreSQL database containing fake employee credentials (name, username, email, date of birth, phone number, plain-text password, and an employment start date) for the fictitious Department of Internet Security at UMD. All containers had their `wget` and `curl` poisoned such that copies of the downloaded files were stored in the host VM. The control container had no Apache web server installed on it. The other containers all received the same Apache version (2.4.41). The five websites were mirrored from existing sites using HTTrack[11] and modified manually to have no sensitive information nor links to legitimate UMD web pages (informational site, password change site) while the rest were created manually (swipe access management site, blank site). This creation process was done by a script that was run once at the beginning of our experiment.

Each of the eleven experiments was controlled by a `systemctl` service. Each service had its own `journalctl` log that we could use for monitoring. We also created a `systemctl` target to start or stop all

[11] Roche, Xavier and Yann Philippot. *HTTrack: Website Copier*. V. 3.49-2. PC. 2017

of these container services simultaneously. That target was configured to start upon boot, ensuring that our experiment could survive an unexpected reboot of the host.

Each `systemctl` service ran a bash script that had input as the template container name, one of the eleven IPs assigned to that specific experiment, and an experiment number. When this script received an interrupt signal, it would save any remaining session data and then exit.

The recycling process was implemented as follows. First, the container template would be copied with the experiment number appended as an identifier. Once the container fully started and a container IP was generated, the hostname of the container was changed to "`is-admin`" (which was consistent for every container), NAT rules were added, and then the MITM control flow began (described in the next paragraph). The NAT rules redirected packets headed towards the external IP into the container except for SSH packets, which were instead redirected into the MITM application. Once an attacker finished their SSH session, the NAT rules were deleted and relevant files and logs were extracted from the container (Apache logs and MITM logs were zipped together and stored on the host VM, and any downloads were stored on the sandbox VM). We also removed any read-only flags from the container's file system to prevent our script from being unable to fully destroy the container. After a random delay, the recycling process started again. See ***diagram 1*** for a visual representation of the recycling flow.

Our control flow between recycles was as follows. When the attacker connected, we checked two things: (1) if there was already an attacker present and (2) if the attacker included dangerous characters (those that could be used to do command injection) in their credentials. If either of those conditions were met, the attacker's connection was rejected. Otherwise, the attacker connection was accepted with a session time limit of one hour. If the attacker executed a non-interactive command within our cache file, the MITM would give back the cached response and would exit with a non-dirty exit code, which would cause our bash script to only restart the MITM server. In all other cases, to be on the side of caution, the MITM would exit with a dirty exit code which would cause our bash script to perform a full recycle of the container. See ***diagram 1*** for a visual representation of this process.

For additional data, we captured all packets going to port 80 using `tcpdump` captures on each

external IP. These captures were stored in 24-hour chunk files on the host VM. The bash script that controlled all eleven `tcpdump` processes was wrapped in a `systemctl` service that was configured to run continuously and restart at reboot.

As mentioned in ***Section 3***, we created a shadow Apache server. To do this, we created a container with Apache installed that had all four websites on it. We then configured Apache to listen to eight ports, one for each external IP with a website, which routed to eight virtual host ports. The logs were continuously created and stored on the host VM. On the host VM, we added NAT rules to redirect HTTP traffic to and from the shadow container only when the containers were down for recycling. When honeypot containers were up, web pages were served from the honeypot container's web server. See ***Figure B12*** for a visual representation of the network configuration.

Finally, we installed NetData onto our honeypot host so that we could monitor the resource usage of our machine from the NetData website. All of our code was uploaded to GitHub as a backup, and processed data was saved in the Google Drive folder provided to us.

From this point on, we refer to the different experiments as follows: the control as Control; the blank page experiment as Blank; the information site experiment as IT; the password change site experiment as PWD; and the swipe access management site as Swipe.

6. Data Collection

We collected data from the ACES MITM server's session logs, each honeypot container's Apache server's HTTP logs, and the shadow Apache server's HTTP logs. From the MITM session logs we collected the attacker's IP, username, password, entry timestamp, container name, non-interactive or interactive, and commands run. From the Apache HTTP logs we collected the IP, timestamp, container name, request type, and user agent. The container name was indicative of the container's website type due to the naming conventions used (e.g. blank-t-2 refers to the second blank honeypot container).

The data points were extracted from their respective data sources and then formatted into a comma-separated values (CSV) file with pipe delimiters, using bash scripts and standard command line

tools such as `grep`, `awk`, and `sed`. Since we were interested in the types of commands run, we extracted all commands from the sessions. If one line had multiple commands concatenated by `&&`, `||`, or `;;`, the split commands were analyzed separately. Due to our storage method, we also split commands based on an individual pipe character, denoted `|`.

This resulted in three separate datasets: MITM attacker session data (not containing commands), attacker command data (one attacker's session could produce many commands), and Apache request data. The script for extracting this data can be seen in ***Script 1C*** in ***Appendix C***.

In total, we collected 589,585 MITM sessions, 525,267 commands, and 131,602 Apache requests. Note that the number of commands is significantly less than the number of MITM sessions. With some manual inspection of the raw MITM session data, we were able to attribute this to the fact that many attackers started a session but exited immediately without performing any action that could be observed through the MITM server.

Once the raw data was cleaned and collected into a CSV file, we used Jupyter Notebook to create a Python workflow for processing the cleaned data. This was particularly useful since Python has many libraries that make processing CSV files and getting basic information about a dataset easy and straightforward (such as Pandas, Numpy, etc.). For the process described below, the data was also imported into a SQLite3 database, where each dataset became a table in the database.

Since our research observes attacker behavior after viewing/processing web page content, we filtered MITM attacker session data to only include sessions where (1) the MITM attacker's IP matched to the GET request's IP, (2) the MITM compromised container matched to the GET request's container, and (3) the MITM attacker's entry time occurred after the GET request's timestamp. This was done using an SQL inner join query, which was run through SQLite3 and Python, whose output was saved into a CSV file. Note that this matching criteria allows for multiple MITM attacker's sessions to match to a singular Apache GET request, as we assume that the attacker has already 'seen' the website in any later sessions. Additionally, since the control honeypots did not have Apache servers, all control data was manually included in the final MITM attacker session data and final command data. This was achieved by

appending all of the control data (i.e. rows of data with a 'control' container type) to the final CSV file. The Jupyter notebook which demonstrates this process may be found in ***Script C2*** in ***Appendix C***. All in all, this filtering process resulted in 364,126 valid MITM sessions and 293,547 valid commands for analysis.

We also collected packets via `tcpdump` aimed towards port 80 and saved files that attackers downloaded onto the honeypot, with the intention of using it for secondary analysis if necessary. However, given that valid data generated by Apache matching was sufficient for our primary analysis and main research question, we chose to not include this data in our analysis. Any other elimination of data or grouping of data was for analysis purposes, and so that reasoning is described in ***Section 7***.

7. Data Analysis

All statistical tests described below were performed using the processed and filtered data found as described in ***Section 6*** unless otherwise stated.

In order to begin analyzing our data, we first had to determine if IP bias played a role in the number of attacks each honeypot received. It is important to note that the analysis done in this paragraph does not directly answer our research question; it is done to determine if there is IP bias and if we can to reduce it by combining data sets. Our experimental design involved two identical honeypots (they had the same web page contents) with different IPs to reduce the effect of IP bias. We compared each set of non-control honeypots using a Mann-Whitney U test and our three Control honeypots through a Kruskal-Wallis test, as our data did not follow a normal distribution (See ***Figure B15***). Through the *p*-values (See ***Figures B1*** and ***B2***), we determined that there exists a statistically significant difference in the number of attacks per attacker IP between the honeypots at a 95% level of confidence. This means that there was likely IP bias within our data, and thus we combined the datasets of each type of honeypot for all future analysis to reduce (not eliminate) the effects of IP bias (e.g. the data from both honeypots that had blank web pages were combined into one dataset). The limitations of combining this data are discussed more in ***Section 8.3***.

With these combined datasets, we began analyzing our data to answer each of our four sub-hypotheses (see ***Section 4***). To evaluate H1 and H3, we used Kruskal-Wallis tests to determine whether the difference between the number of attacks per IP received by each honeypot was statistically significant. These tests were run with the data from all honeypots to answer H1, and all data from honeypots with web servers to evaluate H3 (see ***Figure B4***). We decided to use this test because the distribution of attacks was not normal (see ***Figure B3***) and the data had varying sample sizes.

For both H1 and H3, the Kruskal-Wallis tests did not offer insight on the difference between individual website designs and Control, we conducted post-hoc analysis through Mann-Whitney tests. To answer H1, we ran a Mann-Whitney test between data from Control and data from all honeypots with web servers. The same process was repeated to evaluate H3, in which data between all honeypots with website designs were compared in pairs (see ***Figure B5*** and ***B6***).

We also used the combined datasets to answer H2 and H4. For this section of analysis, we used an iPython Google Colab notebook shown in ***Appendix C***, Script C3. We first manually categorized the commands we received during each session into low, medium, and high levels of invasiveness based on keywords (see ***Figure B7*** for command keywords and **B8** for the final table of categorized commands per honeypot). There were also a small number of unknown commands, which were ultimately not included as data points as their effect was unknown. To evaluate H2, we used chi-square tests of independence comparing Control (which has no web server) to each of the other honeypots (all of which had web servers). See ***Figure B9*** for all *p*-values. To evaluate H4, we once again ran chi-square tests of independence, this time comparing the Blank honeypot (which contained a blank web page) to the Swipe, IT, and PWD honeypots (which contained differing types of web page content; see ***Section 5*** for more details). See ***Figures B9*** for the calculated *p*-values. For these two evaluations, we used chi-square tests of independence because it was the best statistical procedure to examine the differences between our categorical command data.

Given that all but one of the *p*-values for each of our sub-hypotheses in this section were less than 0.05, we are able to reject the null hypothesis: there is evidence to suspect that differing web page content

and the presence of a web server has a statistically significant relationship with attack command rates and the invasiveness of attacker commands.

8. Conclusion

8.1 - Interpretations of Results

To evaluate H1, we compared the number of commands ran by each attacker IP in Control (no web server) to honeypots with web servers (Blank, Swipe, IT, and PWD). The Kruskal-Wallis test, which resulted in a *p*-value less than 0.05, indicated that there was a significant difference in the number of commands per attacker IP among the honeypots. Each Mann-Whitney test comparing Control honeypot to honeypots with web servers yielded statistically significant results, with Control having a lower median number of commands per attacker IP compared to all other honeypots (***Figure B14***). We therefore have evidence that the presence of a web server could lead to a higher number of commands run by each attacker IP.

To evaluate H2, we compared each honeypot with a web server (Blank, Swipe, IT, and PWD) to Control (which has no web server), we found that every pair's test returned *p*-values far lower than 0.05 (see ***Figure B9***), indicating that each honeypot's command invasiveness distribution is significantly different from Control. Thus, we have evidence to suspect that the presence of a web server could lead to differing levels of command invasiveness.

To evaluate H3, we compared the honeypots with web servers in a Kruskal-Wallis test, which yielded a *p*-value of less than 0.05. This result indicates that there is a statistically significant difference in the number of commands run by each attacker IP when changing the website design. The Mann-Whitney test was ran between all pairs of honeypots with website designs. While the Blank-Swipe and Blank-IT pairs yielded *p*-values under 0.05, indicating that there is a statistically significant difference in attacker rates when changing the web design, the Blank-PWD pair resulted in a *p*-value of 0.0804 wherein the null hypothesis cannot be rejected.

To evaluate H4, we compared the honeypots with varying types of web page content (Swipe, IT,

and PWD) to Blank, and found that each pair's test returned a *p*-value far below 0.05 (see ***Figure B9***), indicating that the Swipe, IT, and PWD honeypots' command invasiveness distributions differed significantly from that of Blank. Thus, we have evidence to support that the content of a web page could result in differing levels of command invasiveness.

Thus, when considering our research question (see ***Section 4***), we are able to say that our *p*-values for each sub-hypotheses do indicate that the presence of a web server and varying web page content impacts both attack command frequency and command invasiveness during attacks.

8.2 - Why Are Our Values Significant?

The differences in attack rate and command invasiveness between Control and the others (Blank, Swipe, IT, and PWD) may be because the presence of the web server made attackers believe that the honeypot contained more valuable information, such as personally identifiable information. It could also lead the attacker to believe that the honeypot in question had more resources because it was hosting a web server. However, it is also possible that the random spam attacks (a large amount of attacks in a time period from the same attacker during which only the same command, such as `uname`, is run) that some containers received may have impacted the number of commands in various command categories. For instance, ***Figure B11*** depicts a spam attack during which the command `uname -a` was spammed many times. This made Control's low invasiveness command count higher because `uname -a` is categorized as a low-invasive command. Comparing Blank to any other honeypot would now return a more significant *p*-value because of Blank's unusual command category distribution.

The spam attacks and IP bias may have also similarly impacted the differences in command invasiveness and attack rate between Blank and Swipe, IT, and PWD. However, the differences for this test could also be explained by the differing web page content between Blank, Swipe, IT, and PWD. For instance, the IT honeypot's web page was merely an informational site, whereas the PWD honeypot web page simulated a password login. This might make the attacker believe that the PWD honeypot might contain sensitive information, such as user information or passwords, which could have made them use different commands than they would have normally. To further support this theory, reference ***Appendix A***

part 5, where we perform further analysis on the Swipe honeypot and why its data differs from the other honeypots with web pages. Although all of our honeypots had more medium-level invasive commands (see ***Figure B10***) than other categories, Swipe was the only honeypot whose medium-level category occupied 80% of its total commands. Because Swipe is the only honeypot that contains IoT information, it may have attracted different commands than the other honeypots. This furthers our theory that web page content may have impacted command invasiveness in this experiment.

It should be noted that one of our *p*-values for H3 (determining if web page content had an effect on attack rate) was *not* significant (see ***Figure B5***). Given that the rest of our values for this section were significant, we believe this *p*-value may have occurred because PWD simply received similar levels of attacks to the Blank honeypot due to IP bias. Additionally, we noted that when comparing the median values of commands run per attacker IP for H3 (see ***Figure B14***), Blank had the highest median value compared to the rest of the honeypots. This was interesting because Blank had no web page content, and thus we expected that to be less of interest to attackers than other web pages, such as Swipe or PWD. However, this was not the case: we suspect that Blank may have received more commands per attacker IP because attackers may find an empty web page interesting because it is abnormal, and may indicate vulnerabilities or other hidden valuable information. Of course, Blank may have simply also received more attacks due to IP bias or spam attacks.

As stated above, the difference in command invasiveness may also be due to IP bias, because some honeypots could have randomly gotten more spam attacks. Because of these issues, it is possible that if we continued the experiment for a longer time and accounted properly for IP bias (see ***Section 8.3***), the differences between our honeypots' command invasiveness would reduce, and we would not be able to reject the null hypothesis.

8.3 - Limitations & Future Work If Continued

Our project had many limitations, some of which we would address if we had the chance to continue the experiment. For instance, we only used four differing types of web pages to test how the presence of a web server and differing web page content affects attack rate and invasiveness, which is

limited in scope. To address this, we would include many other types of web page content, such as non-university web pages or web pages in different languages. Furthermore, we did not quantify which websites were necessarily more of interest (i.e. exploitable) than others. For example, we cannot definitively state that Swipe's web page is more of interest to attackers than PWD's web page, which limits our understanding of whether attacks would respond more strongly to web pages that have more perceived exploitable data. If we could redo this project or continue it, we would carefully select websites that have obvious differences in perceived exploitable data so that we can further analyze how web page content affects attack rate and invasiveness of commands.

Additionally, our experiment used GET requests to determine if attackers looked at the web page at all. However, looking at the web page does not equate interacting with or even processing the content from the web page, which may mean that web page data may not influence the attackers at all. If given the chance, we would determine the degree to which attackers interact with the web page, and use that to further refine our dataset (e.g. adding a form or button on the website that attackers could click to measure interactivity). We would also like to further examine what kinds of files the attackers may be downloading on our honeypot using commands like `wget`, as prior literature has noted that attackers can download both malware and large, harmless files to test compromised systems.[12] The first is highly invasive while the latter is less invasive. Currently, our experiment does not consider this context and counts all downloads as highly invasive.

Finally, we would also address the issue of IP bias in our experiment more thoroughly. Although we combined our datasets from honeypots with the same web page design to help reduce this bias (see ***Section 7***), we understand that IP bias still impacts our data. If given the chance, our group would employ more than two honeypots with the same web page but also with rotating IPs within same-web page container groups in order to get more data from various IPs and further reduce bias.

---

[12] Alata, E., V. Nicomette, M. Kaâniche, M. Dacier, and M Herrbeeexplore.ieee.org. "Lessons Learned from the Deployment of a High-Interaction Honeypot." eeexplore.ieee.org, December 11, 2006. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4020829.

Appendix A

1. Lessons Learned

We have a few lessons that we wanted to highlight, which fell into the categories of project management and technical lessons.

Out of the project management lessons, the one lesson we most strongly learned is the importance of giving every idea and criticism, no matter how mundane or extreme, floating time during meetings. Without us making an effort every single time to give all ideas a fighting chance, we would not have as cohesive a project as we have now. In terms of a coding project management lesson, we saw first hand the helpfulness of having a very organized code base from the beginning. By having a good code structure right from the start, it became significantly easier to make

There were a couple of technical lessons that we learned. First, we learned about how wonderful `systemctl` services are, and about how they work in general.

2. Feedback On Project

Our overall feedback is that we believe there should have been more instructor-provided support for this project, especially in the technical aspects. Although we personally did not have too many issues with the technical aspects of the project, we know that other groups had a lot of difficulty doing so. Therefore, if there had been more materials provided to us at the start, we believe the project as a whole could have gone much smoother for everyone.

One of the areas we believe improvements could be made is adding features to the MITM server to make it compliant with the experiment design. For instance, it is recommended to us that we only allow one attacker within a container at a time to prevent multiple attackers within the same container at the same time interfering with each other. We believe that many groups (including us) naïvely assumed that the MITM server would prevent multiple connecting attackers given that it already orchestrates the entire SSH connection. The recommended solution was to modify NAT rules to disallow any new connections to the MITM server, but we believe that this functionality should be built into MITM directly because this would reduce the chance of unexpected failure due to “residue” rules. For instance, let’s say a group has

an attacker-limiting script that inserts and removes a NAT rule to prevent a new SSH connection, but it encounters some error and fails to remove its rule preventing incoming SSH connections into a container. This could prevent that container from accepting new SSH connections until an administrator manually intervenes to remove the rule. Moreover, if this group has other scripts to manage NAT rules, they have to be diligent to ensure that the different scripts are properly appending/inserting the rules in order. A failure to do so could cause the firewall to erroneously deny all incoming SSH connections to a container or permit multiple attackers to connect simultaneously. Therefore, we opted to instead implement an attacker limiter directly within the MITM via the parameter "`--attacker-limit <number>`". This allows the MITM server to directly determine whether to accept or deny a new attacker based on the number of SSH sessions it is currently managing, which prevents the side effects caused by NAT rules. In this way, restarting the MITM server alone would *definitively* ensure that new attackers can connect to a container since restarting the MITM server would reset its active SSH session count to zero, which would allow a new attacker to connect.

Another example of technical support could be providing students with a robust recycling script and have them customize it to their needs. For example, they could be given a `systemctl` service (which seemed to work quite well for us) that they can then duplicate and change depending on their experiment. It can even be a shell of a service, with aspects that groups need to fill in, but having the structure would likely immensely lessen the difficulty of the project.

Following the theme of providing more assistance, we believe there should have been significantly more classes, or at least homeworks, related to analyzing and interpreting data, given the standards we were expected to match in our analysis. The data we were collecting in this project had significantly more caveats and nuances in analyzing it than the few homeworks we were given and the classes we were taught. Perhaps, instead of only having work days after deployment, if some of those days were dedicated lectures about more advanced/specialized data analysis, we think we'd have been much more prepared to tackle the tasks given to us.

3. Pitfalls and failures encountered in the experiment

As said before, we failed to anticipate the usage of `chattr`, which makes files immutable, which resulted in a broken recycling process and rendered some of our honeypots unusable for a few hours while we diagnosed and fixed the issue manually. Moreover, we initially planned to recycle after every attacker's disconnect. However, we failed to anticipate the high rate of non-interactive mode SSH bot attackers which kept our containers in a constant recycling state. This also likely decreased the amount of potential non-bot attackers as the honeypot is inaccessible during the recycling process and bot attackers are more likely to compromise the honeypot faster than non-bot attackers.

4. Were you surprised by the attacks? The attackers' behavior?

We were overall unsurprised by the attacks as attackers majorly ran one noninteractive command, of which the most common being uname. However, an interesting behavior came from some attackers using `chattr` to make a file they created immutable, which disrupted our recycling process as we did not anticipate this type of behavior.

5. Swipe Honeypot - Additional Analysis

One of the most interesting honeypots was the Swipe honeypot. When looking at ***Figure B7***, one can see that Swipe's ratios of command categories (low, medium, and high) to total commands are very different from the other honeypots. For instance, Swipe's medium category occupied 80% of its total commands, whereas the other honeypots' medium categories occupied around 55-68% of their commands. Swipe's unusual command distribution may be due to IP bias, but also could be due to the fact that Swipe was the only web page that had IoT related content. IoT devices are "popular targets of attack", and thus may attract different commands than the other honeypots.[13] This supports the idea that differing web page content (in this case, Swipe's IoT related information) promotes different levels of command invasiveness.

[13] Javier Franco et al., "A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems," IEEE Communications Surveys &Amp; Tutorials 23, no. 4 (August 21, 2021): pp. 2351-2383, https://doi.org/10.1109/comst.2021.3106669.

Appendix B

Mann-Whitney	U-value	Z-score	p value	Significance
blank-t and blank-t-2	2599	10.17431	< .00001	Y
it-t and it-t-2	3777	7.80216	< .00001	Y
pwd-t and pwd-t-2	5161	8.0383	< .00001	Y
swipe-t and swipe-t-2	5944.5	3.08265	0.00208	Y
Kruskal-Wallis		**H-statistic**	**p value**	**Significance**
control-t-1, control-t-2 and control-t-3		86.2493	< .00001	Y

Figure B1: Results of Mann-Whitney and Kruskal-Wallis test to determine if there's a statistically significant difference between containers of the same website design.

Kruskal Wallis Test	H statistic	p-value
Controls	86.2493	1.86725E-19

Pair	Mean Rank difference	Z	SE	Critical value	p-value	p-value/2
x_1-x_2	-41.7493	1.8591	22.4565	53.7588	0.06301	0.03151
x_1-x_3	-266.3357	9.256	28.7743	68.883	0	0
x_2-x_3	-224.5865	7.2073	31.1609	74.5962	5.707e-13	2.853e-13

Group	control-t-2_running	control-t-3_running
control-t_running	**-41.75**	**-266.34**
control-t-2_running	0	**-224.59**

Figure B2: Results of Kruskal-Wallis test of the 3 control honeypots. *p*-value of less than 0.05 means there is a statistically significant difference between the number of commands run per ip among the 3 containers. Further analysis through Mann-Whitney test shows that only control-t is significantly different from the other controls (control-t2 and control t-3)

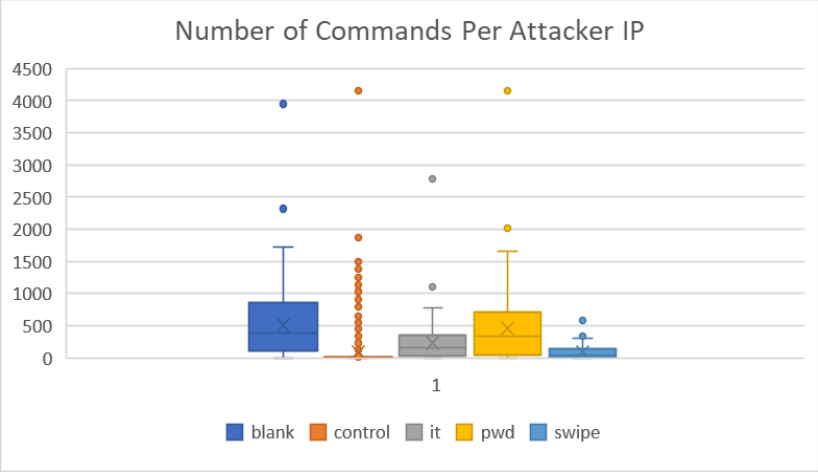

Figure B3: Boxplot of the Number of Commands Per Attacker IP in each container (containers of the same website design were combined). The distribution is not normal and sample sizes differ.

Kruskal Wallis Test	H statistic	p-value
All honeypots	520.1694	2.9069E-111
All honeypots w/ web servers	103.2113	3.16886E-22

Figure B4: Results of Kruskal-Wallis Test run on all honeypots (H1) and all honeypots with web servers (H3). There is a statistically significant difference in the number of commands run per ip address when adding a web server or changing website design.

Pair	Mean Rank difference	Z	SE	Critical value	p-value	p-value/2
x_1-x_2	616.6728	16.6151	37.1152	104.1835	0	0
x_1-x_3	149.4045	3.0029	49.7528	139.6577	0.002674	0.001337
x_1-x_4	52.8204	1.11	47.5879	133.5809	0.267	0.1335
x_1-x_5	316.1287	6.271	50.4112	141.5059	3.587e-10	1.794e-10
x_2-x_3	-467.2683	11.9802	39.0034	109.4838	0	0
x_2-x_4	-563.8524	15.5755	36.2013	101.6183	0	0
x_2-x_5	-300.5441	7.5438	39.8398	111.8318	4.574e-14	2.287e-14
x_3-x_4	-96.5842	1.9681	49.0748	137.7547	0.04906	0.02453
x_3-x_5	166.7242	3.2175	51.8171	145.4525	0.001293	0.0006465
x_4-x_5	263.3083	5.2935	49.7422	139.6281	1.2e-7	6.001e-8

Group	control	it	pwd	swipe
blank	**616.67**	**149.4**	**52.82**	**316.13**
control	0	**-467.27**	**-563.85**	**-300.54**
it	**-467.27**	0	**-96.58**	**166.72**
pwd	**-563.85**	**-96.58**	0	**263.31**

Figure B5: Mann-Whitney Test between all honeypots (x1-blank, x2-control, x3-it, x4-PWD, x5-Swipe) for H1. All groups are significantly different from one another except for PWD-blank and PWD-it. All comparisons involving the control honeypot are significant.

Pair	Mean Rank difference	Z	SE	Critical value	p-value	p-value/2
x_1-x_2	99.1544	4.9206	20.1508	53.1634	8.628e-7	4.314e-7
x_1-x_3	34.1433	1.7458	19.5571	51.5968	0.08084	0.04042
x_1-x_4	193.6646	9.4852	20.4175	53.8669	0	0
x_2-x_3	-65.0111	3.2262	20.1508	53.1634	0.001254	0.0006272
x_2-x_4	94.5103	4.5033	20.9869	55.3692	0.000006691	0.000003345
x_3-x_4	159.5213	7.813	20.4175	53.8669	5.551e-15	2.776e-15

Group	it	pwd	swipe
blank	**99.15**	**34.14**	**193.66**
it	0	**-65.01**	**94.51**
pwd	**-65.01**	0	**159.52**

Figure B6: Mann-Whitney between all honeypots with web servers (x1-blank, x2-it, x3-PWD, x4-Swipe). All comparisons are significant except blank to PWD (H3)

Command Keyword	**Classification**	**Reasoning**
`hostname, nproc, ls, cpu, cloud print, vga, uname, lspci, echo, cat, cd, grep, wc -l, perl, history`	Low level invasive	These commands are mainly used to get information about the honeypot or its files, and do not change the honeypot in any way; they have minimal impact on the honeypot and are thus minimally invasive.
`mkdir, rmdir, rm, chmod, echo >, Hive-password, cat >`	Medium-level invasive	These commands change the honeypot's structure in some way or change its files. The results of these commands are between low-level and high-level impact, and thus are classified as medium level invasive.

`nc localhost, wget, curl, bash -s, apt-get install, ssh-key, tftp, ./`	High-level invasive	These commands are much more dangerous than medium-level commands, because they can be used to dramatically alter the honeypot (such as downloading malware). Thus, they are classified as high-level invasive.
`0-3, 2d, 3d\\, fault, Binary, payload`	Unknown commands	Some of these commands resulted from corrupted log files, and some were simply mysterious. Despite looking into these commands, we are unsure of what they do. These were not considered as data points.

Figure B7: This depicts the command classification for levels of invasiveness for H2 and H4. The keywords (not the entire command) for identifying certain commands can be found in column 1, the level of invasiveness in column 2, and the reasoning for why the commands were classified in their respective categories in column 3.

	Unknown	**Low**	**Medium**	**High**
Blank	0	31078	46820	5826
Control	33	30108	47823	6210
IT	0	7081	22644	3495
PWD	0	27226	45592	7114

Swipe	0	1927	10007	548

Figure B8: Results of categorizing commands from all honeypots (leftmost column) into unknown, low, medium, and high levels of intensity for H2 and H4.

Honeypots Tested	**Chi-Square Test of Independence Results**
All Honeypots (Control, Blank, IT, PWD, Swipe)	chi-square: 5558.191810913713 p-value: < 1e^-100 degrees of freedom: 8
Control & Blank	Chi-square: 37.22280377463221 p-value: 8.263629928292738e-09 Degrees of freedom: 2
Control & IT	chi-square: 2366.0934019196347 p-value: < 1e^-100 degrees of freedom: 2
Control & PWD	chi-square: 151.61085255003024 p-value: 1.1970758529566932e-33 degrees of freedom: 2
Control & Swipe	chi-square: 2484.779504110799 p-value: < 1e^-100 degrees of freedom: 2
Blank & Swipe	chi-square: 2677.9749909386783

	p-value: < 1e^-100 degrees of freedom: 2
Blank & IT	chi-square: 2799.1964137233294 p-value: < 1e^-100 degrees of freedom: 2
Blank & PWD	chi-square: 311.3172795593492 p-value: 2.502140445894987e-68 degrees of freedom: 2

Figure B9: Results of chi-square tests of independence between honeypots for H2 & H4. The pink section shows a chi-square test between all honeypots, the blue section refers specifically to the tests done for evaluating how the presence of a web server impacts invasiveness of commands, and the green section refers to tests done for evaluating how web page content impacts invasiveness of commands.

Note that some of these p-values (<10e-100) appear as "0.0" in the Colab notebook. This is because the number was so small the computer could not contain it as a float value.

	% High/Total	% Medium/Total	% Low/Total
Blank	6.958578185	55.92183842	37.11958339
Control	7.380468499	56.83673833	35.78279317
IT	10.52077062	68.16375677	21.31547261
PWD	8.900065055	57.03848271	34.06145223
Swipe	4.390322064	80.17144688	15.43823105

Figure B10: Percentage of category of commands per total commands for each honeypot. For instance,

column 2 depicts the percentage of high-invasive commands over the number of total commands for each honeypot.

147.182.233.56 on blank	# Times Run
uname -a	2014
PATH=/dev/shm/:/tmp/:./:/var/tmp/:/root/:/etc/:$PATH nc localhost 1234	304
Total	2318

Figure B11: Depicts a "spam attack" on the Blank honeypot. One can see that the command `uname -a` was repeated 2014 times, which would make Blank's "low invasive" command count value (see Figure B7) unusually high.

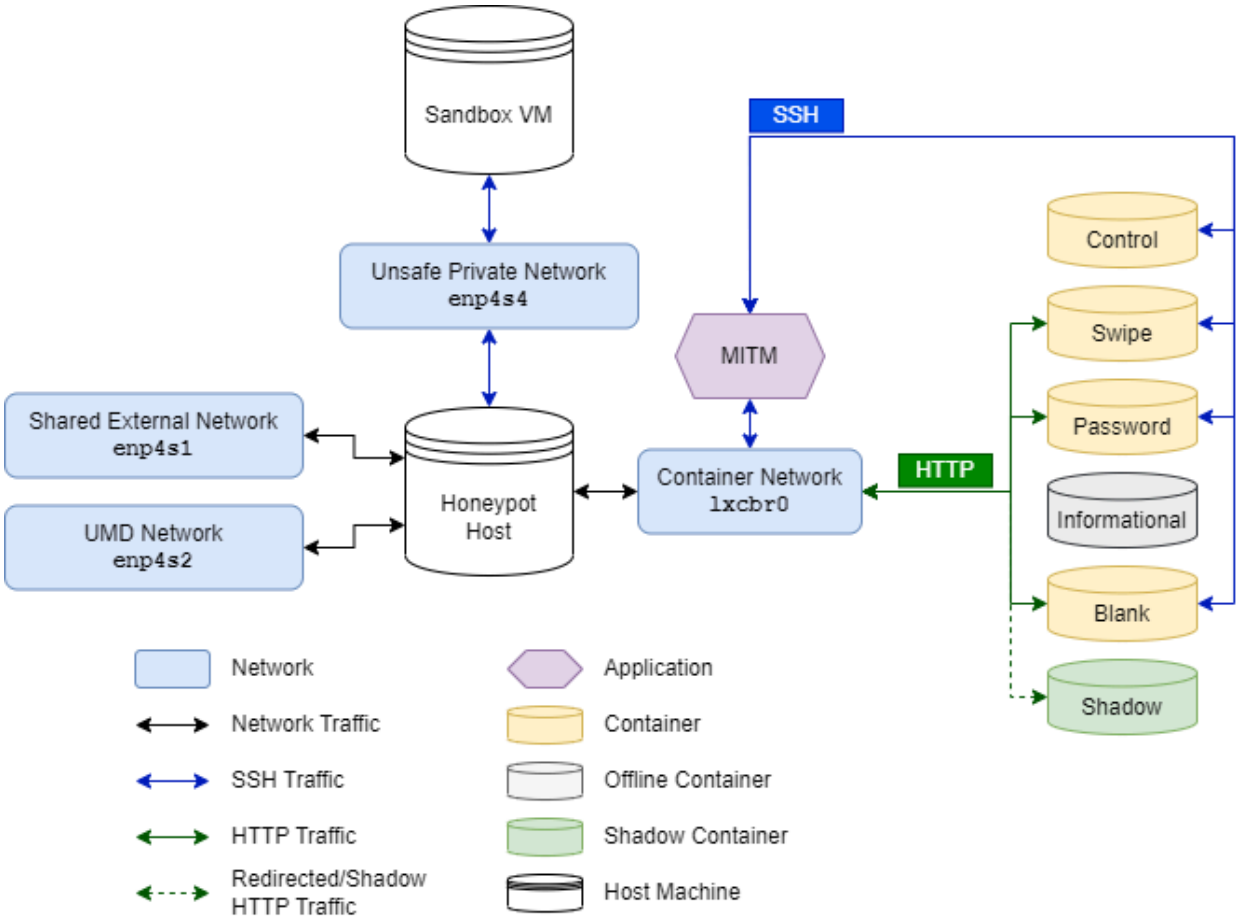

Figure B12: A network topology of our experiment. In the diagram, we use the informational website container (in gray) as an example of what happens when a container goes down for recycling; the shadow

container (in green) temporarily serves the website design on the same public IP address until the container is back online, but does not accept attacker SSH traffic.

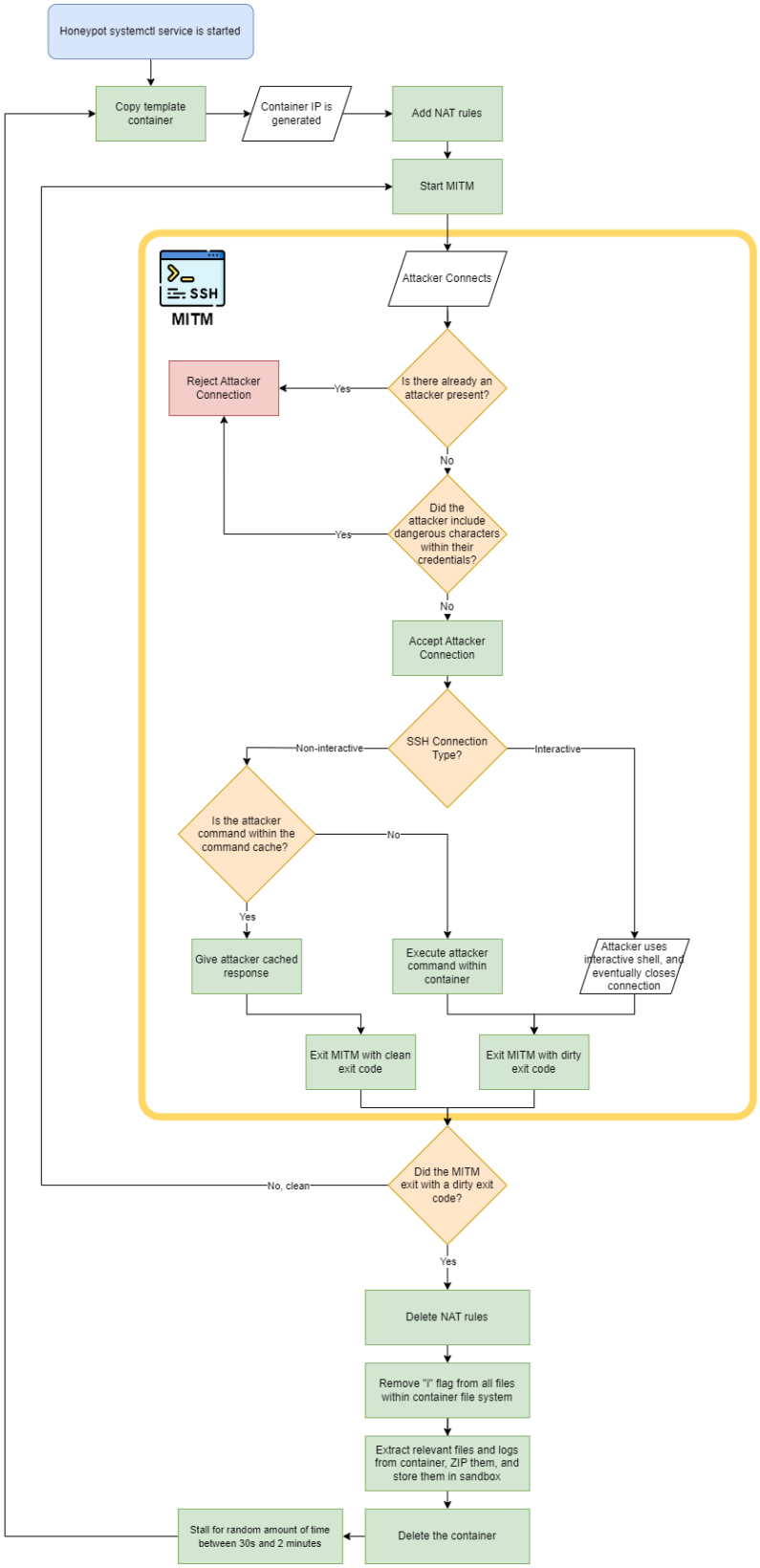

Figure B13: A diagram detailing the recycling control flow of our experiment design. The highlighted box is the control flow within our modified MITM, while everything else is our recycling bash script.

Median # of Commands Ran by Each Attacker IP	
Blank	380.5
Control	3
IT	155
PWD	333.5
Swipe	49

Figure B14: Medians of the number of commands ran by each attacker ip in Blank, Control, IT, PWD, and Swipe (H1).

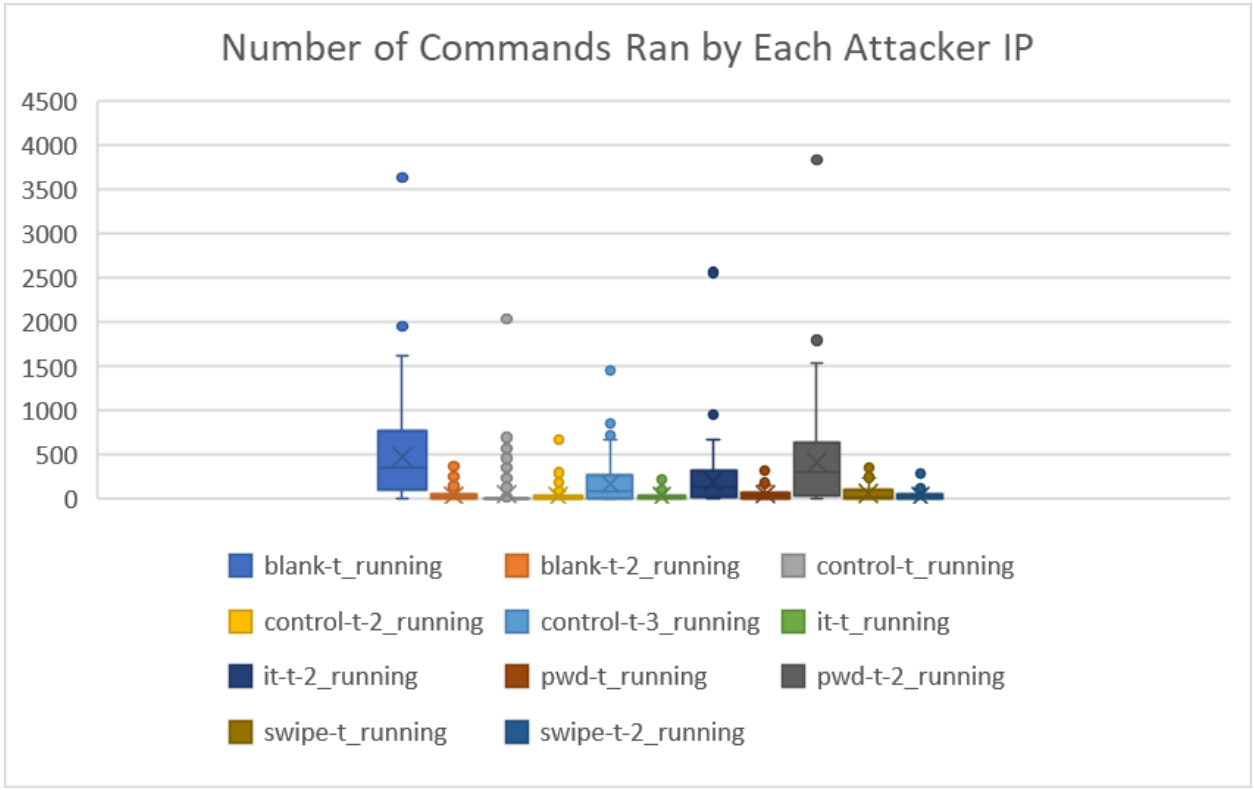

Figure B15. Box plot of the number of commands ran by each attacker IP in each container. See ***Section 6*** for container naming conventions.

Appendix C

Script C1: Data Processing Script

This script will process data where an MITM session log is zipped together with an apache log, resulting in cleaned CSV files containing data for (1) MITM sessions, (2) MITM commands per session, and (3) HTTP Apache request data.

https://github.com/friendly-besties-incorporated/all-honeypot-code/blob/main/data_processing/process_basic_data.sh

Script C2: Data Processing Notebook

This is a direct link to the data processing iPython Notebook in a public github repository, as it was too lengthy to include directly in this paper. This notebook further processes data pre-processed by Script C1.

https://github.com/friendly-besties-incorporated/all-honeypot-code/blob/main/hacs200master.ipynb

Script C3: Chi-Square Analysis Notebook

This is a direct link to the iPython Notebook used for the chi-square analysis of H2 and H4. If you wish to run this notebook, upload “outApacheCMDS1” and “outApacheCMDS2” from processed_data.zip in the Github to the notebook and then run it.

https://colab.research.google.com/drive/17YeavbwGCn3XZ8bijLRpbACWWLVQYlOd?usp=sharing

Works Cited / References

Amin, Sayyed Mehdi, Esmaeil Kheirkhah, Hediyeh AmirJahanshahi Sistani, and Haridas Acharya. “An Experimental Study of SSH Attacks by Using Honeypot Decoys.” *Indian Journal of Science and Technology* 6, no. 12 (2013): 1–12. https://doi.org/10.17485/ijst/2013/v6i12.8.

Canali, Davide, and Davide Balzarotti. “Behind the Scenes of Online Attacks: an Analysis of Exploitation Behaviors on the Web.” *20th Annual Network & Distributed System Security Symposium (NDSS 2013)*, February 2013. https://doi.org/hal-00799082.

Franco, Javier, Ahmet Aris, Berk Canberk, and A. Selcuk Uluagac. “A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems.” *IEEE Communications Surveys & Tutorials* 23, no. 4 (August 21, 2021): 2351–83. https://doi.org/10.1109/comst.2021.3106669.

“Hackers Are Now Scanning for SSH Keys to Exploit.” PAM solutions, Key Management Systems, Secure File Transfers. SSH Communications Security, March 25, 2022. https://www.ssh.com/blog/ssh-key-scan-attack-honeypot.

Maunder, Mark. “New Attacker Scanning for SSH Private Keys on Websites.” Wordfence, October 18, 2017. https://www.wordfence.com/blog/2017/10/ssh-key-website-scans/.

Melese, Solomon Z, and P.S. Avadhani. “Honeypot System for Attacks on SSH Protocol.” *International Journal of Computer Network and Information Security* 8, no. 9 (2016): 19–26. https://doi.org/10.5815/ijcnis.2016.09.03.

Nicomette, Vincent, Mohamed Kaâniche, Eric Alata, and Matthieu Herrb. “Set-up and Deployment of a High-Interaction Honeypot: Experiment and Lessons Learned.” *Journal in Computer Virology* 7, no. 2 (2010): 143–57. https://doi.org/10.1007/s11416-010-0144-2.

Rahmatullah, Dandy Kalma, Suiya Michrandi Nasution, and Fairuz Azmi. “Implementation of Low Interaction Web Server Honeypot Using Cubieboard.” *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, January 16, 2016. https://doi.org/10.1109/iccerec.2016.7814970.

Roche, Xavier and Yann Philippot. *HTTrack: Website Copier*. V. 3.49-2. PC. 2017

Vasek, Marie, John Wadleigh, and Tyler Moore. “Hacking Is Not Random: A Case-Control Study of Webserver-Compromise Risk.” *IEEE Transactions on Dependable and Secure Computing* 13, no. 2 (2016): 206–19. https://doi.org/10.1109/tdsc.2015.2427847.

Alata, E., V. Nicomette, M. Kaâniche, M. Dacier, and M Herrbeeexplore.ieee.org. “Lessons Learned from the Deployment of a High-Interaction Honeypot.” eeexplore.ieee.org, December 11, 2006. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4020829.