Comparing Attacker Behavior Between Personal and Corporate Systems

Julian Burgos, Joseph Jubilee, Trevor Lorin, & Aaron Nemirovsky

Group 2G The Dream Team

HACS 200

Section 1: Executive Summary

In our research, our goal is to answer the question: to what extent do attackers interact with personal systems including personal identifiable information (PII) in comparison to corporate systems including company data? We define attackers as anyone that enters another person's computer system without permission or authorization, which is an activity deemed illegal by the US Computer Fraud and Abuse Act (U.S. Code, 2018). When we identified an attacker by this action, we measured and defined the extent of their interaction with data by how many files and directories the attacker opened, manipulated, or downloaded and if the attacker deployed any malware on the system. Malware consists of any software that an attacker downloads or installs onto a system. The actions revolving around files show that the attacker has an interest in the contents of the files contained in the system (Yuill, 2004).

Our null hypothesis is that once obtaining access to the user system, threat actors' attack methods will not change based on the contents of the user system. Our alternative hypothesis is that attackers will execute more commands to download malware and have higher interaction with files when on a corporate system than on a personal or empty system. In order to collect data to test our hypothesis, we utilized honeypot containers with a Man-In-The-Middle (MITM) server to observe attacker commands related to either file interaction or malware installation while in the system. We created three unique honeypot systems with varying honey files to appear as either a corporate, personal, or empty system. After analyzing the data collected by the MITM server on attacker behavior, we found that there was not enough statistical data to prove a difference in behavior between our three honeypots, leading us to fail to reject the null hypothesis.

Section 2: Background Research

According to Holt (2011) in his article *Examining the Social Networks of Malware Writers and Hackers*, there is a present issue of unauthorized access to systems for both individuals and corporations. Holt (2011) also wrote, "the number of high profile data breaches has increased significantly, with millions and in some cases billions of customer records stolen by small groups of hackers" (p. 2). This research helps back the idea that there are attackers gaining unauthorized access to systems with the purpose to steal the contained data. In order to understand what data attackers target, we will use honeypots with varying types of data as honey. Using false data in files with enticing names such as "passwords.txt" can help monitor and track an attacker's perusal of the file system present in a system, according to Jim Yuill, Mike Zappe, Dorothy E. Denning, and Fred Feer's 2004 research paper *Honeyfiles: deceptive files for intrusion detection*. Furthermore, in their paper, it is supported that attackers are highly vulnerable to honey file deceptions and will interact with these files when present in a system they have infiltrated. Along with just interacting with files, the research from that paper also stated that attackers will look for files with keywords that can often be used in honey files such as "passwords".

Our research aims to further this idea by experimental design to determine if attackers interact with these honey files more if they relate to corporate information rather than personal identifiable information. Similar to our experimental research design, there has previously been research on attacker behavior while utilizing honeypots as the method of obtaining data. In the 2007 research paper *Profiling Attacker Behavior Following SSH Compromises*, the researchers set up four honeypots for attackers to enter using ssh. They then collected data on if the attackers took any of the five actions they specified. The actions were: "checking the configuration,

changing the password, downloading a file, installing/running rogue code, and changing the system configuration" (Ramsbrock, 2007). Our research implements a very similar experimental design, with our data collection being more focused on how much the attackers interact with varying types of files on the honeypot systems. Our research has importance in understanding what the attacker's primary target and goals are. In the 2006 article *Cyber-Warfare Threatens Corporations: Expansion into Commercial Environments* by Kenneth J. Knapp and William R. Boulton, it is stated that most attackers are no longer targeting the government due to the high risk and that "private citizens and commercial infrastructures are most likely to be primary targets". By better understanding whether attackers rather target data from commercial targets or personal information, we can better allocate defense and mitigate data theft or manipulation. For instance, if it is found that attackers tend to install malware more often on computers containing PII data than corporate-like data, personal computer owners might be inclined to invest more into malware protections such as antiviruses.

Section 3: Experiment Design Changes

For our initial experimental design, we planned to create three unique honeypot systems with varying honey files to appear as corporate, personal, and empty systems. Each of these honeypots only varied in the type of data their honey files contained, otherwise having the same functionalities. Our design included a MITM server to monitor attacker commands, a recycling script to allow for follow-up attackers to have a clean and un-modified user system to attack, and an auxiliary script to filter our MITM log file for the six specific commands (section 5) we were looking for to analyze attacker behavior.

Although we had to alter some aspects of the technical implementation of our design due to unforeseen problems, our initial experimental design was the one we stayed with throughout the duration of our project. Our largest technical change we implemented on November 1st, 2022, when we discovered that our MITM server was allowing more than one attacker into the honeypot at a time. To fix this issue so we could collect viable data, we had to create a script that would block all other IP addresses from accessing our honeypots after the first IP connected. Another technical change we implemented was on November 9th, 2022 when we changed our design from one large log file per honeypot into separate log files per attacker session in order to have data that would be separated by the attacker and easier to parse and conduct statistical analysis on.

Another slight design change we made on November 3rd, 2022 when we were experiencing difficulties retrieving the health data of our honeypots through NetData. We had to create our own script which would provide us with information about the health of our honeypots since NetData was not providing accurate and up-to-date alerts. These implementation changes still provide the same functionality as our original experimental design.

In spite of the fact that we experienced some difficulties which hindered the team's progression on the project such as problems launching the honeypot or capturing data, the team never had to change our original research question or key proponents of our experimental design.

Section 4: Research Question and Hypothesis

Our research question was: to what extent do attackers interact with personal systems containing PII in comparison to corporate systems containing non-personal, company data? This

question requires analysis of specific attacker behavior when inside systems containing different types of honey files. Our null hypothesis is that attackers will not change their behavior based on the contents of the system they are in. Our alternative hypothesis is that attackers will execute more commands to download malware and have higher interaction with files when on a corporate system than on a personal or empty system. Our justification for this hypothesis is that a corporate system's data does not pertain to just one individual making its files and information more valuable for attackers to steal. Also compromising a corporation's computer can allow for attackers to move laterally to infiltrate a corporation's complete system making malware use more beneficial for attackers than on just one personal device. We hypothesize that attackers when looking for files will be more enticed by the data in a corporation's files in comparison to a personal computer's contents.

Section 5: Experiment Design

For our experiment, we created three different honeypots: one that represented a personal machine (named JIMS-PC), one that represented a corporation (named APLL), and one that represented a brand new machine (named ubuntu). The honey files that we used for the corporation honeypot were documents and other data that would be on an employee's machine. For example, we created documents for memos, excel sheets for contact lists, and excel spreadsheets for accounting purposes and income statements. The spreadsheets used for accounting purposes were sheets such as balance sheets and accounting journals. Refer to Appendix B for a visualization of the honey file system used in the APLL honeypot. The second honeypot, the personal machine, contained honey that was represented as resumes, password lists, school items, photos, and medical items. To represent medical items we used pdf

screenshots of a COVID-19 card as well as a document that represents medical insurance information. School information is represented as pdf versions of textbooks and documents representing schoolwork. Refer to Appendix B for a visualization of the honey file system used for the JIMS-PC honeypot. The last honeypot created, the brand new machine, contains no specific type of honey. The honeypot is made to seem like a new system with no personal files in the home directory.

We have each honeypot recycle every 30 minutes using crontab to call our recycling script. Before we created our recycling script we took an <lxc-snapshot> of each container with all the honey untouched and with no attackers in the container. We then use the <-r> option in the script to restore each container before starting the container back up to allow attackers back in. Each honeypot is recycled by stopping all the forever processes as well as removing all the <iptables> rules. After all of the honeypots are stopped and restored, our MITM setup script is called (Appendix C - Recycle Script). Before setting an IP to each honeypot, we randomized each recycle by having all the possible IP addresses in a file then using the <shuff> command and assigning the top 3 to each honeypot in the MITM setup script. Each honeypot was set up with <iptables> rules to be open to the internet. In addition to the open <iptables> rules, we set up a connection to the ACES MITM server through <iptables> as well as having that process run using the forever command to ensure that MITM was running for our 30-minute time frame and collecting data to the specified log file (Appendix C - MITM Setup Script).

In addition to adding the MITM rules, we also had to add separate firewall rules that would only allow one attacker at a time. The firewall rules gather the first IP that entered the honeypot, and only allows that IP access to the honeypot, blocking all other IPs (Appendix C - MITM Setup Script). Along with our <iptables> rules, we implemented the given firewall rules

so that our honeypots could be available to attackers outside of the UMD network and open to the public. Refer to Appendix C - Firewall Setup Script for in-depth comments and more information on our firewall configurations.

A health monitoring script was created to check RAM usage, disk space usage, network traffic, and average system load. To help with our experiment we also created other scripts to help with data collection such as our multiple scripts to parse our data (File Grep Script). We filtered the MITM logs to look for six specific commands that related to malware being downloaded and file interaction. These six commands were <cat>, <wget>, <curl>, <install>, <ls>, and <cd>. We parsed attacker commands for the specific commands stated to see if attackers were interacting with our honey and to see if they were downloading anything onto the honeypot. We also created a crontab to run our multiple scripts at reboot to ensure that if our system has to reboot we will continue to collect data. Our crontab also runs our recycling script every 30 minutes and the health monitoring script is run every minute 28 past every hour. Refer to Appendix C to see all scripts including the Crontab.

Section 6: Data Collection

Data collection for our honeypots consisted of using the MITM logs to log commands executed in the honeypots by attackers. We specifically searched and filtered our MITM log files for the commands <cat>, <wget>, <curl>, <install>, <ls>, and <cd>. We searched specifically for <cat>, <ls>, and <cd> because these are commands that would show that attackers were interacting with the honey files. We were looking for whether an attacker's decision on downloading malware or executing other commands would depend on the honey. We searched

for <curl>, <wget>, and <install> commands to see what attackers' would download onto the honeypot. We separated the MITM session logs by an attacker and then further processed the data by using another script to isolate just the commands run by each attacker. We then used our file grep script (Appendix C) to filter for the six specific commands related to our research question. We logged 924 attacker sessions on each of our honeypots for a total of 2772 attacker sessions to collect data from. After filtering for the six specific commands, we found a total of 7661 relevant commands executed across all three of our honeypots through all of our attacker sessions. We also created a script that would count the number of relevant commands each attacker executed in their session which we used for statistical analysis.

Our original design did not have separation of attacker session logs and was quickly found to be difficult to parse into usable data. When we found the design flaw that we were allowing multiple attackers at once, we then also changed our data processing to separate by attacker session in each honeypot. This led us to eliminate our data points collected before we stopped multiple attackers from entering our honeypots at once.

When it came to the data processing we first had to review the original log file and see any recurring trends that would help with processing the data into usable information. After examining the original log files produced by the MITM server to see how we could filter for just the relevant data collected, we found that all the commands were found on a line that contained the phrase "EXEC". We created a script that would <grep> for "EXEC" in the original log file and output the contents to another file inside a directory corresponding to the honeypot type (e.g personal, corporate, empty). With these new directories containing log files with only the commands executed, we continued to process the data by running our grep file for the six specific commands on every log file and counting the number of relevant commands. This data is

grouped by honeypot type and each point represents the data collected from a single attacker session. We are interpreting the data as a measure of if an attacker is interacting with the honey or is executing relevant commands that relate to the honey being present on the system. By grouping the data by honeypot, it allows for the data to be used to analyze variance in attacker commands between the honeypots. The data can be visualized in Appendix B - Box Plot and was used for statistical analysis.

Section 7: Data Analysis

In order to analyze our data, we used our three samples from our different honeypot configurations. We collected data from 924 attacker sessions from each of our three honeypots. The data that we used for our analysis is visualized through a boxplot in Appendix B.

We utilized a Kruskal-Wallis test to compare our data from our three honeypots and conduct statistical analysis on the variance between the samples. We chose this test since it allows for the comparison of two or more samples of different sizes in order to analyze variance. This test also does not assume the data is a normal distribution and can provide accurate results on if there is a statistical difference between the data in our three honeypot samples.

For our experiment, we said that a p-value less than or equal to 0.1 would prove a statistically significant difference between tested samples. When we ran the Kruskal-Wallis test on our data, it resulted in a p-value of 0.4438. Since our statistical test resulted in a p-value much above the threshold of 0.1, we were unable to conclude that there was a statistically significant difference in the data between the three honeypot groups. Since we were unable to prove a

statistically significant difference in the data we collected on attacker behavior between the honeypots, we fail to reject the null hypothesis stated in section 4.

Through our data analysis, however, we also noticed an interesting trend that relates to our results. We found that when attackers executed any commands in our honeypots, they were always in non-interactive mode. The only files that attackers interacted with were files that are present on all systems such as configuration files. There were many attackers that repeated the same command multiple times, leading us to believe a large portion of our attacker data was from automated bot interaction, a possible explanation for the lack of significant difference between the data from the three honeypot samples.

Section 8: Conclusion

With our Kruskal-Wallis test p-value of 0.4438, we concluded that there was no statistically significant difference between the attacker commands executed in the three different honeypots and that we failed to reject our null hypothesis. This p-value means there was not a significant difference between the three honeypots when measuring the amount of the six commands (section 5) executed. This means that attackers behaved similarly in each of our honeypots, regardless of the honey files present. Not only was there not a significant difference between the honeypots, but there were also extremely low measures of the commands relating to file interaction. We can conclude that attackers that downloaded malware were not concerned with the contents of the system. We also observed attackers only interacted with files that were not honey files furthering our conclusion that attackers do not change their behavior due to the

file contents of a system. This leads us to conclude that attackers are not seeking out specific data and are attacking our honeypots for computing resources.

Due to repeated commands executed by the same attacker and the use of non-interactive mode throughout a majority of our attacker sessions we believe that most of our interaction was from automated bots. If we were to continue our research, we would investigate attacker behavior assuming we would get bot interaction. We would like to research how attacker behavior changes if an error message is generated when attackers execute any command in our honeypots which we believe could alter automated attacker behavior.

Appendix A

We were able to learn a great number of lessons by completing this project. First, this project taught us the importance of having good time management skills. This entails having good scheduling skills in order to make deadlines for when we want to finish certain tasks and delegating which group members are responsible for completing such tasks. Additionally, we were able to see the importance of learning from peers. The weekly stand-ups were beneficial in gauging our own progress and seeing the progress of other groups. This allowed us to see whether other groups had similar issues to us and possible solutions found by the other groups. Ultimately, we were able to use debugging techniques to our benefit. This includes allotting time before deadlines to fix issues. In addition, we also learned that running scripts multiple times to ensure they work is a critical step in debugging and assessing script functionality. Especially with our recycling script, bugs only became apparent after multiple cycles and runs of the script.

We thought as a group that this was a good project that simulates an actual work environment. This gave us insight into the processes that go into large projects that include project planning, data collection, and ultimately data analysis. We felt however that we were least prepared for the statistical analysis required for this project. We had very little instruction and the instruction assumed statistical knowledge most of our group members did not have. We felt as if it was a beneficial experience using our skills from HACS101 to create the honeypot however we feel as if we were also required to use statistical skills we were not properly taught.

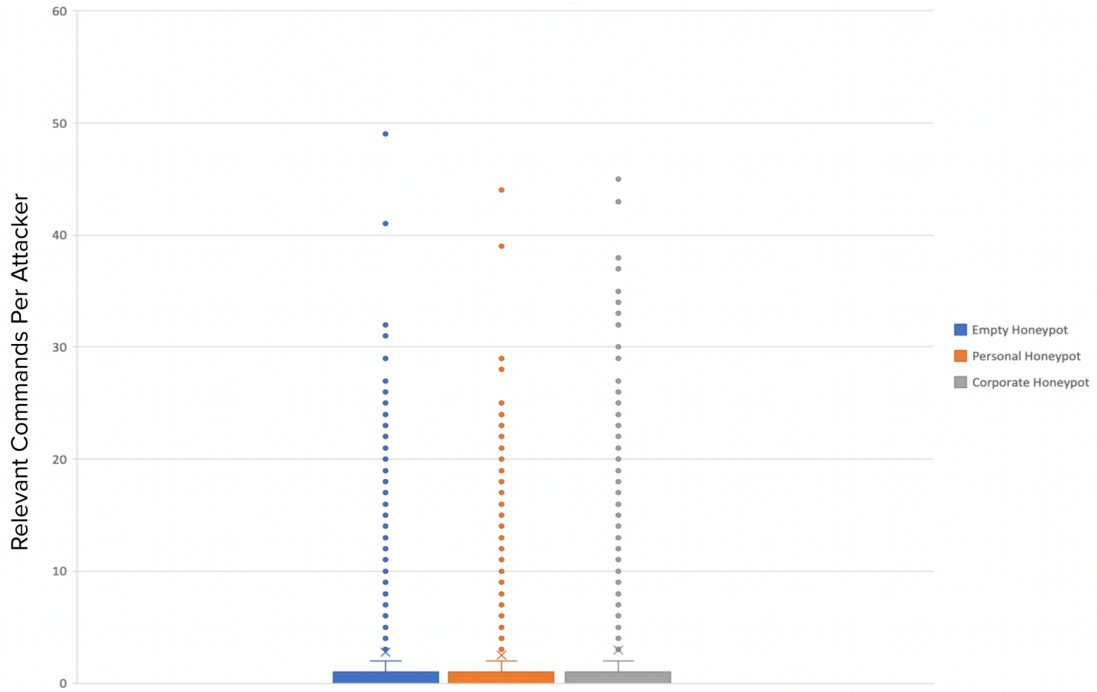
We were partially surprised by the attacker's behavior. At the project's beginning, we expected commands to be used in the interactive mode, but this was not the case. Nearly all the data that we collected was in the noninteractive mode. Ultimately, we were not able to see any trends in any of our three honeypots. We expected some automated bot interaction but were surprised to see almost all of the data appeared to be from an automated source. With both non-interactive mode and repeated commands, it seems that a majority of our attackers were automated.

We encountered a combination of many pitfalls and failures in this experiment. First, we assumed that we set up firewall rules on reboot but at that time did not know how to check whether the firewall rules actually became set up. Once we realized that they were not being set up, we changed the way that we called the script by using a combination of the iptables save and restore to fix this issue. Additionally, we saw that many multiple attackers were able to go into the same honeypot which made the data unusable. As seen in Appendix B under the MITM setup script, we created an iptables blocking script that solved this issue. In terms of monitoring the health of our honeypots, we set up NetData and were able to get it working excluding the fact that we were not receiving any email notifications about the honeypots. This caused us to instead

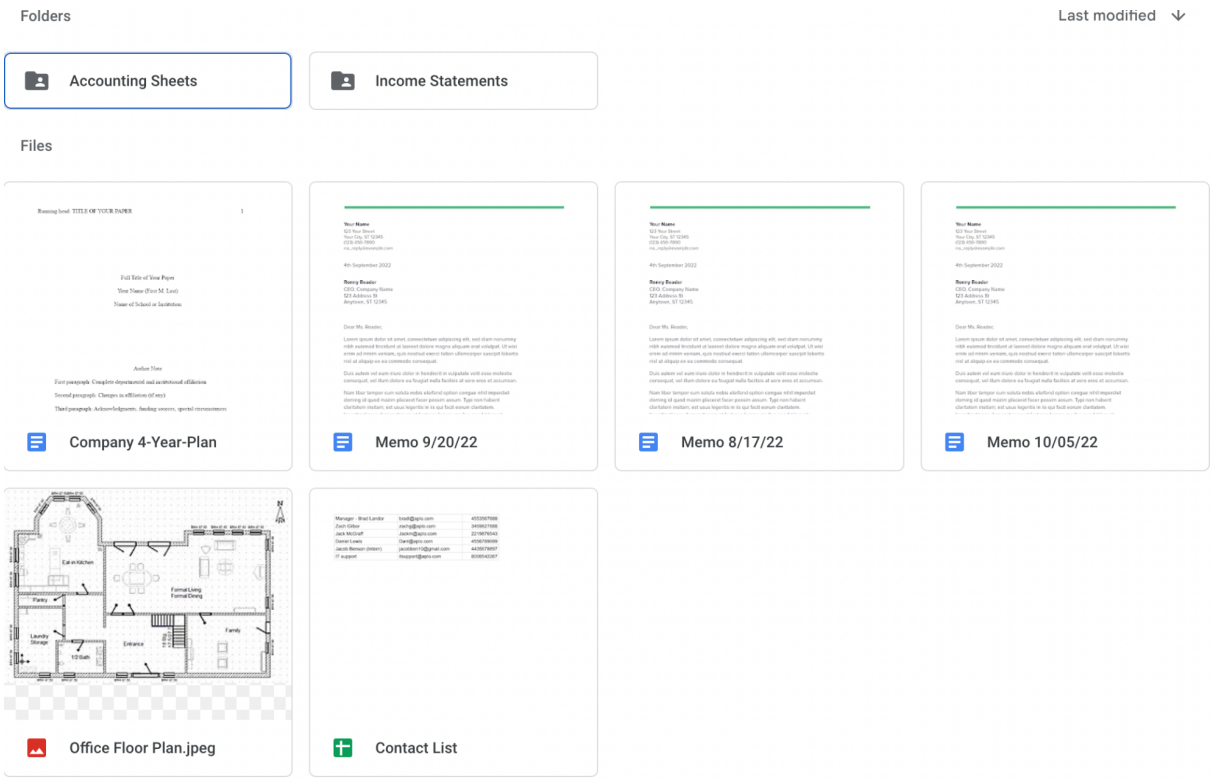
create a script that would be used for monitoring that allowed us to get email notifications. Later on, we realized that net data email notifications were working but the issue was we did not set the threshold low enough to test for notifications previously.

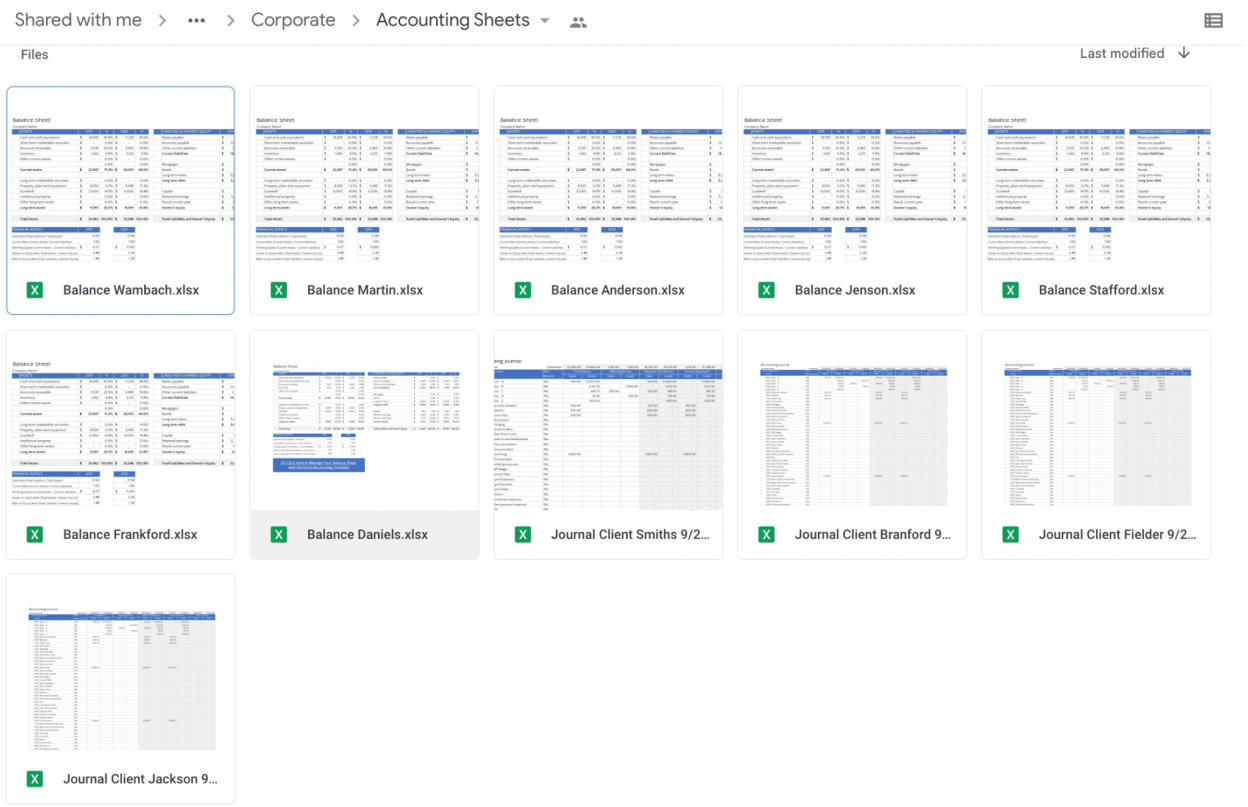
Appendix B

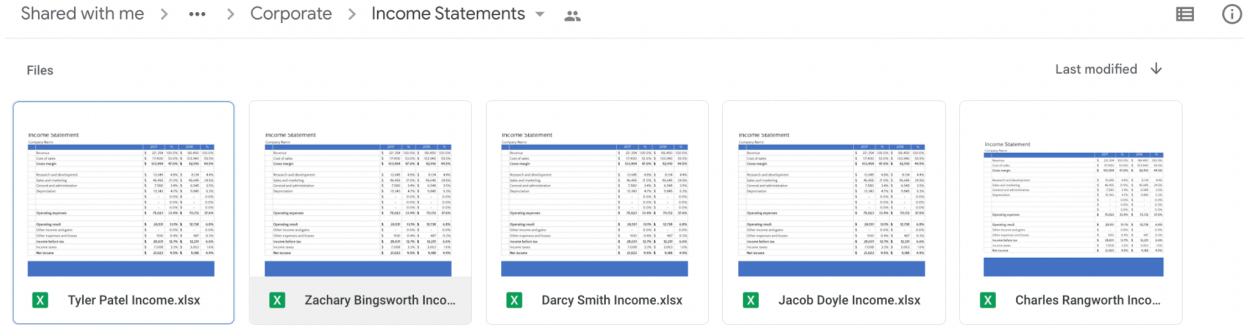
Data Boxplot:



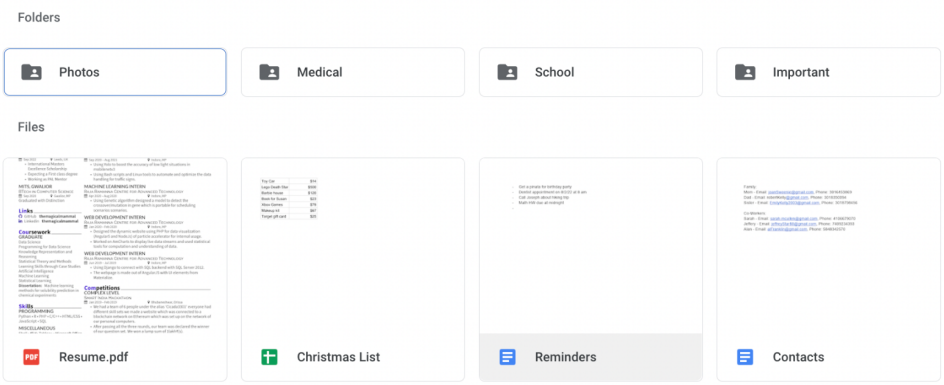
Corporate Honey (Visual Representation):

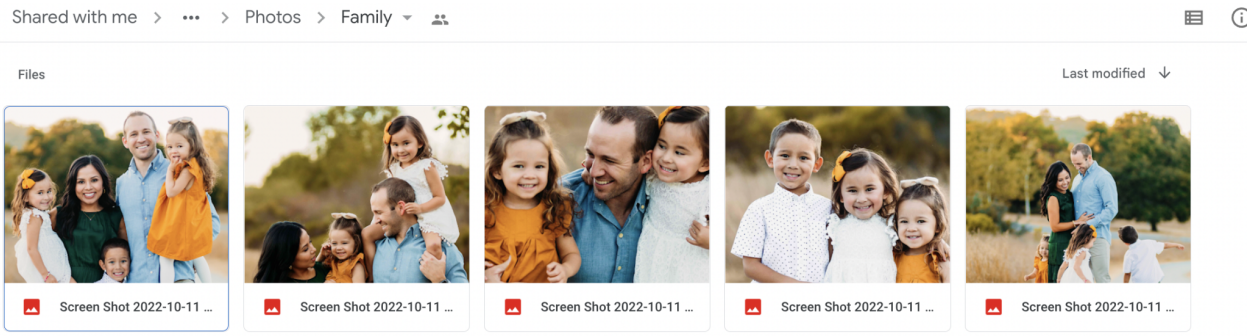


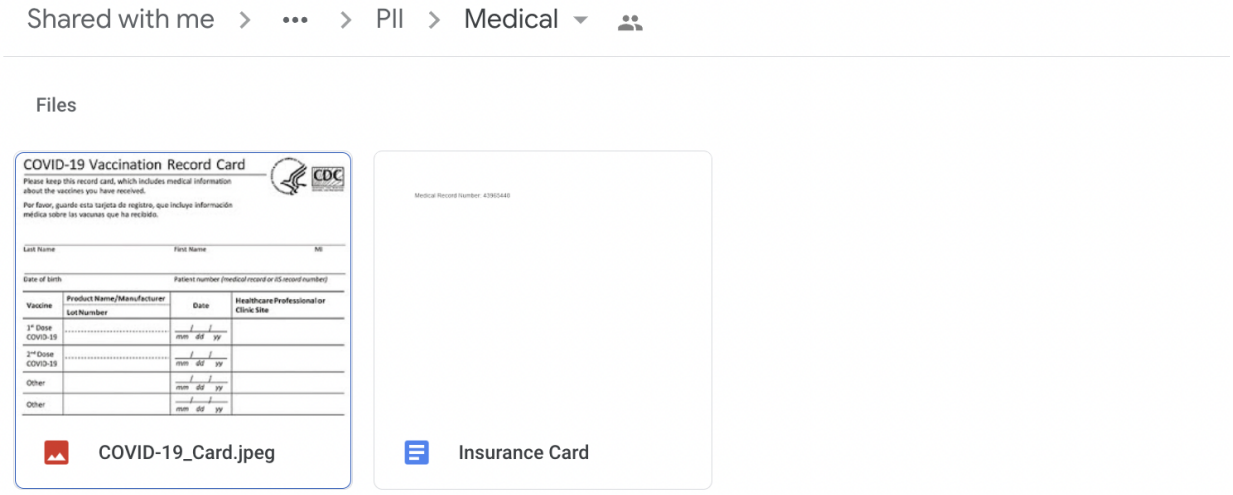




PII Honey(Visual Representation):


Folders
Last modified
Photos
Medical
School
Important
Files
Resume.pdf
Christmas List
Reminders
Contacts


Shared with me
Photos
Family
Files
Last modified
Screen Shot 2022-10-11 ...
Screen Shot 2022-10-11 ...
Screen Shot 2022-10-11 ...
Screen Shot 2022-10-11 ...
Screen Shot 2022-10-11 ...


Shared with me
PII
Medical
Files
COVID-19 Vaccination Record Card
Please keep this record card, which includes medical information about the vaccines you have received.
Por favor, guarde esta tarjeta de registro, que incluye información médica sobre las vacunas que ha recibido.
Last Name
First Name
MI
Date of birth
Patient number (medical record or IIS record number)
Vaccine
Product Name/Manufacturer
Lot Number
Date
Healthcare Professional or Clinic Site
1st Dose COVID-19
2nd Dose COVID-19
Other
Other
mm dd yy
COVID-19_Card.jpeg
Insurance Card

Shared with me > ••• > PII > School

Folders

Textbooks

Files

Bio - Selective Breeding ...

ASTR100 Lab 3/12/21

School UID

Shared with me > ••• > School > Textbooks

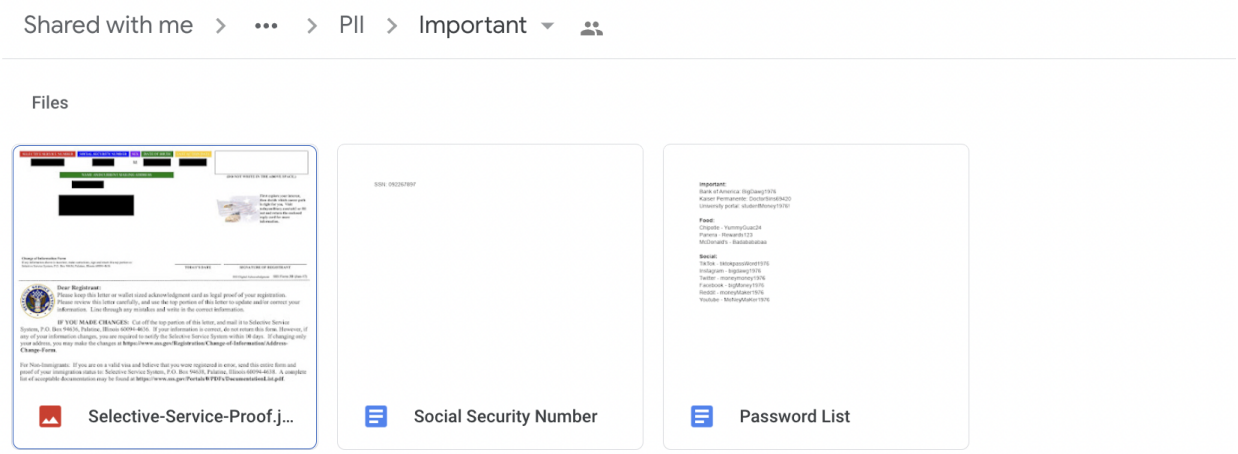
Files

PDF

LINEAR ALGEBRA

PDF

The Essential Cosmic Pe...



Appendix C

MITM Setup Script:

```
# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:03:48]
→ cat mitmSetupAndIPTable.sh
#!/bin/bash

shuf /home/student/HACS200-Group2G/ipAdresses > /home/student/HACS200-Group2G/shuffledIpAddresses
emptyMachine=`head -n 1 /home/student/HACS200-Group2G/shuffledIpAddresses`
personalMachine=`head -n 2 /home/student/HACS200-Group2G/shuffledIpAddresses | tail -n 1`
corporateMachine=`head -n 3 /home/student/HACS200-Group2G/shuffledIpAddresses | tail -n 1`
hostIP='10.0.3.1'
date=`/usr/bin/date`



if [ $# -ne 3 ]
then
  echo "Argument needs to be container names in this order: <empty container> <personal container> <corporate
container>"
  exit 1
else

        emptyContainerIP=`sudo lxc-ls -f | grep -w  $1 | awk '{print $5}'`
        personalContainerIP=`sudo lxc-ls -f | grep -w  $2 | awk '{print $5}'`
        corporateContainerIP=`sudo lxc-ls -f | grep -w  $3 | awk '{print $5}'`
  /usr/bin/touch /home/student/emptyLogs/"emptyContainerLog$date.log"
  sleep 1
  /usr/bin/touch /home/student/personalLogs/"personalContainer$date.log"
  sleep 1
  /usr/bin/touch /home/student/corporateLogs/"corporateContainer$date.log"
  sleep 1

  sleep 3

        sudo sysctl -w net.ipv4.conf.all.route_localnet=1
  sleep 5
        sudo forever -l /home/student/emptyLogs/"emptyContainerLog$date.log" -a start /home/student/MITM/mitm.
js -n $1 -i $emptyContainerIP -p 6900 --auto-access --auto-access-fixed 1 --debug --mitm-ip $hostIP
        sleep 5
  sudo ip link set dev enp4s2 up
        sudo ip addr add $emptyMachine/24 brd + dev "enp4s2"

        sudo iptables --table nat --insert PREROUTING --source 0.0.0.0/0 --destination $emptyMachine --jump DN
AT --to-destination $emptyContainerIP
        sudo iptables --table nat --insert POSTROUTING --source $emptyContainerIP --destination 0.0.0.0/0 --ju
mp SNAT --to-source $emptyMachine
        sudo iptables --table nat --insert PREROUTING --source 0.0.0.0/0 --destination $emptyMachine --protoco
l tcp --dport 22 --jump DNAT --to-destination $hostIP:6900
  sudo iptables --table nat --insert POSTROUTING --source $emptyMachine --destination 0.0.0.0/0 --jump SNAT --
to-source $emptyContainerIP

        sudo sysctl -w net.ipv4.conf.all.route_localnet=1
        sleep 5
```

```
  sudo forever -l /home/student/personalLogs/"personalContainer$date.log" -a start /home/student/MITM/mitm.js
-n $2 -i $personalContainerIP -p 6901 --auto-access --auto-access-fixed 1 --debug --mitm-ip $hostIP
        sleep 5
  sudo ip addr add $personalMachine/24 brd + dev "enp4s2"

        sudo iptables --table nat --insert PREROUTING --source 0.0.0.0/0 --destination $personalMachine --jump
 DNAT --to-destination $personalContainerIP
        sudo iptables --table nat --insert POSTROUTING --source $personalContainerIP --destination 0.0.0.0/0 -
-jump SNAT --to-source $personalMachine
        sudo iptables --table nat --insert PREROUTING --source 0.0.0.0/0 --destination $personalMachine --prot
ocol tcp --dport 22 --jump DNAT --to-destination $hostIP:6901
  sudo iptables --table nat --insert POSTROUTING --source $personalMachine --destination 0.0.0.0/0 --jump SNAT
 --to-source $personalContainerIP

        sudo sysctl -w net.ipv4.conf.all.route_localnet=1
        sleep 5
  sudo forever -l /home/student/corporateLogs/"corporateContainer$date.log" -a start /home/student/MITM/mitm.j
s -n $3 -i $corporateContainerIP -p 6903 --auto-access --auto-access-fixed 1 --debug --mitm-ip $hostIP
        sleep 5
[  sudo ip addr add $corporateMachine/24 brd + dev "enp4s2"                                                    ]

        sudo iptables --table nat --insert PREROUTING --source 0.0.0.0/0 --destination $corporateMachine --jum
p DNAT --to-destination $corporateContainerIP
        sudo iptables --table nat --insert POSTROUTING --source $corporateContainerIP --destination 0.0.0.0/0
[--jump SNAT --to-source $corporateMachine                                                                     ]
        sudo iptables --table nat --insert PREROUTING --source 0.0.0.0/0 --destination $corporateMachine --pro
tocol tcp --dport 22 --jump DNAT --to-destination $hostIP:6903
  sudo iptables --table nat --insert POSTROUTING --source $corporateMachine --destination 0.0.0.0/0 --jump SNA
T --to-source $corporateContainerIP

fi

exit 0
```

```
# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:17:24]
[→ cat iptablesBlocker.sh                                                    ]
#!/bin/bash


  /home/student/HACS200-Group2G/iptablesBlockerC.sh /home/student/MITM/logs/logins/APLL.log
  /home/student/HACS200-Group2G/iptablesBlockerP.sh /home/student/MITM/logs/logins/JIMS-PC.log
  /home/student/HACS200-Group2G/iptablesBlockerE.sh /home/student/MITM/logs/logins/ubuntu.log

# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:18:22]
[→ cat iptablesBlockerC.sh                                                   ]
#!/bin/bash

#grep with argument that should be logfile ex APLL.log
attackIp= `tail -n 1 /home/student/MITM/logs/logins/$1 | colrm 1 24 | cut -d ';' -f1`
#add iptable that blocks everything
iptables --insert FORWARD --source 0.0.0.0 --destination 10.0.3.1 --protocol tcp --destination-port 6903 --jum
p DROP
# add iptable that allows traffic from one ip
iptables --insert FORWARD --source $attackIP --destination 10.0.3.1 --protocol tcp --destination-port 6903 --j
ump ACCEPT

# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:18:39]
[→ cat iptablesBlockerP.sh                                                   ]
#!/bin/bash

#grep with argument that should be logfile ex APLL.log
attackIp= `grep "Auto Access" /home/student/personalContainer.log| tail -n 4 | head -n 1 | cut -d" " -f8 | cut
 -d"," -f1`
#add iptable that blocks everything
iptables --insert FORWARD --source 0.0.0.0 --destination 10.0.3.1 --protocol tcp --destination-port 6901 --jum
p DROP
# add iptable that allows traffic from one ip
iptables --insert FORWARD --source $attackIP --destination 10.0.3.1 --protocol tcp --destination-port 6901 --j
ump ACCEPT

# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:18:46]
[→ cat iptablesBlockerE.sh                                                   ]
#!/bin/bash

#grep with argument that should be logfile ex APLL.log
attackIp= `tail -n 1 /home/student/MITM/logs/logins/$1 | colrm 1 24 | cut -d ';' -f1`
#add iptable that blocks everything
iptables --insert FORWARD --source 0.0.0.0 --destination 10.0.3.1 --protocol tcp --destination-port 6900 --jum
p DROP
# add iptable that allows traffic from one ip
iptables --insert FORWARD --source $attackIP --destination 10.0.3.1 --protocol tcp --destination-port 6900 --j
ump ACCEPT
```

Recycle Script:

```
# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:06:43]
→ cat recycleScript.sh
#!/bin/bash

emptyMachine=`head -n 1 /home/student/HACS200-Group2G/shuffledIpAddresses`
personalMachine=`head -n 2 /home/student/HACS200-Group2G/shuffledIpAddresses | tail -n 1`
corporateMachine=`head -n 3 /home/student/HACS200-Group2G/shuffledIpAddresses | tail -n 1`
hostIP='10.0.3.1'


#Containers should already be made before deployment so that we can recycle the containers by jest redeploying
 their snapshots before the attacker has entered.
if [ $# -ne 3 ]
then
  echo "Argument needs to be container names in this order: <empty container> <personal container> <corporate
container>"
  exit 1
else
  containerExist1=`sudo lxc-ls -f | grep -w $1 | cut -d' ' -f1`
  containerExist2=`sudo lxc-ls -f | grep -w $2 | cut -d' ' -f1`
  containerExist3=`sudo lxc-ls -f | grep -w $3 | cut -d' ' -f1`
  if [[ $containerExist1 = $1 && $containerExist2 = $2 && $containerExist3 = $3 ]]
  then
    sudo forever stopall
    sleep 2

    emptyContainerIP=`sudo lxc-ls -f | grep -w  $1 | awk '{print $5}'`
    personalContainerIP=`sudo lxc-ls -f | grep -w  $2 | awk '{print $5}'`
    corporateContainerIP=`sudo lxc-ls -f | grep -w  $3 | awk '{print $5}'`

    attackIP_E=`grep "Auto Access" /home/student/emptyContainerLog.log| tail -n 4 | head -n 1 | cut -d" " -f8
| cut -d"," -f1`
    attackIP_P=`grep "Auto Access" /home/student/personalContainer.log| tail -n 4 | head -n 1 | cut -d" " -f8
| cut -d"," -f1`
    attackIP_C=`grep "Auto Access" /home/student/personalContainer.log| tail -n 4 | head -n 1 | cut -d" " -f8
| cut -d"," -f1`

#Empty Machine
    sudo iptables --table nat --delete PREROUTING --source 0.0.0.0/0 --destination $emptyMachine --jump DNAT -
-to-destination $emptyContainerIP
    sudo iptables --table nat --delete POSTROUTING --source $emptyContainerIP --destination 0.0.0.0/0 --jump S
NAT --to-source $emptyMachine
    sudo iptables --table nat --delete PREROUTING --source 0.0.0.0/0 --destination $emptyMachine --protocol tc
p --dport 22 --jump DNAT --to-destination $hostIP:6900
    sudo iptables --table nat --delete POSTROUTING --source $emptyMachine --destination 0.0.0.0/0 --jump SNAT
--to-source $emptyContainerIP
    sudo ip addr delete $emptyMachine/24 brd + dev "enp4s2"
    sudo lxc-stop $1
    sleep 2
```

```
#personal machine
    sudo iptables --table nat --delete PREROUTING --source 0.0.0.0/0 --destination $personalMachine --jump DNA
T --to-destination $personalContainerIP
    sudo iptables --table nat --delete POSTROUTING --source $personalContainerIP --destination 0.0.0.0/0 --jum
p SNAT --to-source $personalMachine
    sudo iptables --table nat --delete PREROUTING --source 0.0.0.0/0 --destination $personalMachine --protocol
 tcp --dport 22 --jump DNAT --to-destination $hostIP:6901
    sudo iptables --table nat --delete POSTROUTING --source $personalMachine --destination 0.0.0.0/0 --jump SN
AT --to-source $personalContainerIP
    sudo ip addr delete $personalMachine/24 brd + dev "enp4s2"
    sudo lxc-stop $2
    sleep 2


#Corporate Machine
    sudo iptables --table nat --delete PREROUTING --source 0.0.0.0/0 --destination $corporateMachine --jump DN
AT --to-destination $corporateContainerIP
    sudo iptables --table nat --delete POSTROUTING --source $corporateContainerIP --destination 0.0.0.0/0 --ju
mp SNAT --to-source $corporateMachine
    sudo iptables --table nat --delete PREROUTING --source 0.0.0.0/0 --destination $corporateMachine --protoco
l tcp --dport 22 --jump DNAT --to-destination $hostIP:6903
    sudo ip addr delete $corporateMachine/24 brd + dev "enp4s2"
    sudo iptables --table nat --delete POSTROUTING --source $corporateMachine --destination 0.0.0.0/0 --jump S
NAT --to-source $corporateContainerIP
    sudo lxc-stop $3
    sleep 2

#Snapshots of each container will already be created as well as the contents inside the container will already
 be created before this script will be run.
    sudo lxc-snapshot $1 -r snap0
    sleep 5

    sudo lxc-snapshot $2 -r snap0
    sleep 5

    sudo lxc-snapshot $3 -r snap0
    sleep 5

    sudo lxc-start $1
    sleep 10

    sudo lxc-start $2
    sleep 10

    sudo lxc-start $3
    sleep 10
```

```
#Sets up mitm and IP Table rules for the three containers
    /home/student/HACS200-Group2G/mitmSetupAndIPTable.sh $1 $2 $3
    sleep 5

  else
    echo "Container Does not exist"
    exit 1
  fi
fi
```

Firewall Setup Script:

```
# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:18:51]
[→ cat firewall_rules.sh                                                                                                ]
#!/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 log
s!)
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 "Conta
iner Network DNS TCP"
/sbin/iptables -A INPUT -i $CONTAINER_INTERFACE -p udp -m udp --dport 53 -j ACCEPT -m comment --comment "Conta
iner Network DNS UDP"
/sbin/iptables -A INPUT -i $CONTAINER_INTERFACE -p tcp -m tcp --dport 67 -j ACCEPT -m comment --comment "Conta
iner Network DHCP TCP"
/sbin/iptables -A INPUT -i $CONTAINER_INTERFACE -p udp -m udp --dport 67 -j ACCEPT -m comment --comment "Conta
iner 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 connec
tions 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/est
ablished 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 honey
pots"
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-sr
cmask 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 outgo
ing"

exit 0
```

File Grep Script:

```
# student at ubuntu-vm in ~ [19:22:23]
[→ cat grepDir.sh
#!/bin/bash

cd $1
num=1;

for FILE in *
do
     echo $FILE >> ~/emptyNames;
done
while read p;
do
   echo $p;
  ~/HACS200-Group2G/logfileGrep.sh "$p" ~/emptyGrepFinal/logfile$num;
   ((num=num+1));
done <~/emptyNames
#for FILE in *; do ~/HACS200-Group2G/logfileGrep.sh $FILE ~/perGrep/logfile$num && num++; done;
```

```
# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:09:17]
[→ cat logfileGrep.sh
#!/bin/bash

#argument 1 will be the filename for the original log and argument 2 will be the new log
touch $2
echo ">>>ALL LINES INCLUDING WGET<<<\n" >> $2
grep "wget" "$1" >> $2

echo ">>>ALL LINES INCLUDING CURL<<<\n" >> $2
grep "curl" "$1" >> $2

echo ">>>ALL LINES INCLUDING INSTALL<<<\n" >> $2
grep "install" "$1" >> $2

echo ">>>ALL LINES INCLUDING LS<<<\n" >> $2
grep " ls" "$1" >> $2

echo ">>>ALL LINES INCLUDING CAT<<<\n" >> $2
grep " cat" "$1" >> $2

echo ">>>ALL LINES INCLUDING CD<<<\n" >> $2
grep "cd" "$1" >> $2
```

```
# student at ubuntu-vm in ~ [19:22:43]
[→ cat dataCountC.sh                                                                ]
#!/bin/bash

cd $1
num=1;

for FILE in *
do
     cat $FILE | grep -v ">>>" | wc -l >> corpCount
done
exit 0

# student at ubuntu-vm in ~ [19:23:26]
[→ cat dataCountE.sh                                                                ]
#!/bin/bash

cd $1
num=1;

for FILE in *
do
     cat $FILE | grep -v ">>>" | wc -l >> empCount
done
exit 0

# student at ubuntu-vm in ~ [19:23:46]
[→ cat dataCountP.sh                                                                ]
#!/bin/bash

cd $1
num=1;

for FILE in *
do
     cat $FILE | grep -v ">>>" | wc -l >> perCount
done
exit 0
```

Health Log Script:

```
# student at ubuntu-vm in ~/HACS200-Group2G on git:main ✖ [19:21:52]
[→ cat healthLogger.sh                                                              ]
#!/bin/bash

date=`/usr/bin/date`
/usr/bin/touch /home/student/"healthLog$date"
#Checking RAM
echo ">>>RAM<<<" >> /home/student/"healthLog$date"
/usr/bin/free -m >> /home/student/"healthLog$date"
#Checking Disk Space
echo ">>>Disk Space<<<" >> /home/student/"healthLog$date"
/usr/bin/df >> /home/student/"healthLog$date"
#Checking Average System Load
echo ">>>Average System Load<<<" >> /home/student/"healthLog$date"
echo `/usr/bin/uptime` >> /home/student/"healthLog$date"
#Checking Network Traffic
echo ">>>Network Traffic<<<" >> /home/student/"healthLog$date"
/usr/bin/netstat -altn >> /home/student/"healthLog$date"
```

Sudo Crontab:

```
# student at ubuntu-vm in ~ [20:44:12]
[→ sudo crontab -l                                                              ]
[[sudo] password for student:                                                   ]
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
 @reboot             sleep 5;/usr/sbin/modprobe br_netfilter
 @reboot             sleep 10;/usr/sbin/sysctl -p /etc/sysctl.conf
 @reboot             sleep 15;/usr/sbin/iptables-restore /etc/firewall.v4
 @reboot             sleep 10;/usr/bin/lxc-start ubuntu
 @reboot             sleep 10;/usr/bin/lxc-start JIMS-PC
 @reboot             sleep 10;/usr/bin/lxc-start APLL
 @reboot             sleep 60;/home/student/HACS200-Group2G/recycleScript.sh ubu
ntu JIMS-PC APLL > /home/student/cron_log

 */30 * * * *          /home/student/HACS200-Group2G/recycleScript.sh ubuntu JIM
S-PC APLL > /home/student/crontest
 28 */1 * * *           /home/student/HACS200-Group2G/healthLogger.sh

# @reboot             /home/student/HACS200-Group2G/iptablesBlocker.sh
```

Bibliography

Holt, T. J., Strumsky, D., Smirnova, O., & Kilger, M. (2012). Examining the social networks of malware writers and hackers. *International Journal of Cyber Criminology*, *6*(1).

Knapp, K. J., & Boulton, W. R. (2006). Cyber-warfare threatens corporations: Expansion into commercial environments. *Information Systems Management*, *23*(2), 76–87. https://doi.org/10.1201/1078.10580530/45925.23.2.20060301/92675.8

D. Ramsbrock, R. Berthier and M. Cukier, "Profiling Attacker Behavior Following SSH Compromises," 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), 2007, pp. 119-124, doi: 10.1109/DSN.2007.76.

J. Yuill, M. Zappe, D. Denning and F. Feer, "Honeyfiles: deceptive files for intrusion detection," Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004., 2004, pp. 116-122, doi: 10.1109/IAW.2004.1437806.