Analysis of Antivirus Effects on Attacker Behavior

Within a Honeypot Environment

Group K

James Andrews, Alex Barker, Anjali Paliyam, Emma Pellegrino, Rishi Rajesh

HACS202 - 0101

May 7, 2024

Table of Contents

Executive Summary

We aim to investigate the impact of the presence or announcement of antivirus software on the behavior of attackers within a honeypot environment. Through this testing, we are looking to answer our research question, 'What are the impacts of both the perceived presence and the actual presence of antivirus software on the behavior of attackers?' Our research aims to contribute to the understanding of attacker behavior in response to antivirus presence or announcement within honeypot environments. By analyzing the effectiveness of antivirus software in deterring attackers, we hope to provide insights that enhance cybersecurity strategies.

We hypothesize that the presence of an antivirus, regardless of whether it's announced, will decrease the attacker's interaction with the system, due to perceived risks, increased time and effort required for evasion, and concerns about exposure. Additionally, we believe that disclosing the use of an antivirus on a machine that is not equipped with an antivirus will decrease how much the attacker interacts with the system.

To begin to analyze attacker behavior, we collected attacker IPs, session IDs, time spent, and number of commands. In our analysis of this data, we found that since our Fisher's Exact Test's p-value was $0.000500 < 0.05$, we reject the null hypothesis and have evidence supporting a significant association between the honeypot type and the commands. Additionally, since our Kruskal-Wallis H-test p-value was 0.480822, we failed to reject the null hypothesis and cannot conclude that there is a significant difference in the median session length between honeypots.

Through our analysis, we found that while there is an association between honeypot type and the commands entered, there was no significant difference in attackers' session length

(interaction) between the different honeypot treatments, so our data does not support our hypothesis.

Background research

In researching similar honeypot projects, we found that there is a lack of experimentation on antivirus effects on honeypot visitors. Research tends to focus on either the performance of the antivirus or the malware deployed by honeypot visitors.

In 2016, researchers used a high-interaction honeypot to compare various antivirus' ability to protect against malware. They then used this data to draw conclusions on the effectiveness of each antivirus program (Algaith et al., 2016). This study helped show us the capability of honeypots for measuring antivirus effects, along with how to compare environments with different levels of protection.

In *Antivirus Evasion Methods in Modern Operating Systems*, Dominik Samociuk discusses how antivirus software can be easily evaded with small changes to malware. According to the paper, one common antivirus detection mechanism is signatures. A signature is essentially the program's unique code pattern and is how similar programs can be identified. Malicious programs' signatures are stored in a database and compared when new programs are downloaded to determine if they are malicious or not.

Another method of detecting malware is through behavioral detection, which is an analysis of the program's operations. In this way, new programs that act similarly to previously flagged malware can be detected and quarantined. Heuristic detection is performed by examining the program's intentions using machine learning or similar processes. This can be done statically by reading through the program's decompiled code or dynamically by running the program in a

sandbox, emulating the host computer while containing any potentially threatening actions. While this process may lead to false positives, it is more effective at detecting new malware than other methods.

The researchers then discussed several ways malware can evade antivirus responses. One such way is through manipulation of antivirus signature databases, causing the antivirus to quarantine the user. Attackers can also change the code and signature of their malware so that it doesn't match known signatures, but this may not be effective against heuristic detection. Additionally, malware can be encrypted, transferred to the victim, then decrypted and executed to avoid static analysis. A similar method is done with morphism, where the virus has a different appearance each time it is copied, making it harder for antiviruses to recognize them. Finally, with process injection, malware is placed into another process, providing new permissions and making it very difficult to detect.

Some examples of evasion programs were given, namely Hyperion, TheFatRat, and Shellter. Hyperion encrypts the payload with a random key, then uses brute force decryption to allow the payload to be executed. TheFatRat encrypts the payload, alters its signature, and generates unique payloads with unique code each time it runs. Shellter injects a payload into a secure executable program in order to mask the payload's presence. These methods are used to avoid reverse engineering and heuristic analysis. After applying a variety of treatment combinations to six popular antiviruses, the researchers found that Hyperion was not effective at hiding the payload because obfuscation of malware is not effective against heuristic and behavioral analysis. They noticed that Shellter was very effective in injecting malware into reputable programs, especially when using .raw files. The authors also performed subsequent tests after 8 months and found that most of the antiviruses improved in detection rates. These

results suggest that with newly developed antivirus evasion techniques, attackers may be able to bypass antivirus software. This would decrease the impact the antivirus has on their engagement with the honeypot**.**

Modifications made in the design of our experiment

We primarily stuck to our original project plan and experimental design. Throughout our implementation, we made some changes based on insights we gained along the way and project limitations. These changes focused on improving our original plans for our honeypot and stayed true to our project's original purpose and methodology.

We decided to change some of our dependent variables to better reflect the purpose of our experiment. Originally we planned to track “time spent by an attacker per session, attacker IP addresses, number and type of commands entered by the attacker, number of independent visits, number of repeated visits by an attacker, any programs downloaded or run, and outbound connections.” We decided not to track the number of independent visits by the attacker like we originally planned, as we decided the number of commands was a much better metric for the attacker's behavior levels. We also decided to add in the date of login, time of commands, and login credentials of the attacker, to help us better categorize their behavior.

We initially planned on using Avast as our antivirus program for our ‘AV’ and ‘BannerAV’ containers but ended up needing to pivot after learning the free version of Avast does not work for Ubuntu 20.04. We wanted to find another program that was free, reliable, and would be recognizable to attackers as an effective antivirus. After researching we settled on ClamAV, as it seems to be one of the most recognizable free antivirus programs available for Ubuntu 20.04, according to multiple websites and forums.

Summary of data collection

Going into this experiment, we planned to collect the attacker's IP address, session ID, and number of independent visits, along with the date and time of entry. As our experiment came to a conclusion we decided to gather more data points about the attacker. The full list amounted to: date of login, time of commands, number of commands, content of commands, username of the attacker, password of the attacker, and the IP address of the attacker.

We collected this data using man in the middle implementation with our four honeypots. We started our experiment on the 12th of April and ended on the 2nd of May. In total, the experiment was run for 21 days. During this time, we created and destroyed 288 honeypots. 96 AV, 92 Banner, 52 Banner-AV, and 48 Control. This is about where we expected the number to be as we destroyed the honeypots 4 hours after the initial login with an estimated 1 hour between attackers. This data was then used to form a basis to help answer our research question on how an attacker responds to the perceived presence of an antivirus.

After extracting the data into a CSV, we then cleaned the data. This was done in Python using the Pandas library. It consisted of fixing the data type of the Date column to be datetime so that we could perform calculations on the dates and consolidating variations of the same command entered so that we would have a better understanding of the overall trends in the data. We also sorted the dataframe chronologically since the data was extracted arbitrarily. Then, we identified “sessions” by checking if each command happened within four hours of the previous one and had the same username and password. If this was the case, we reasoned that both commands likely came from the same attacker, regardless of a change in IP address. We denoted this by adding a new Session column to our dataframe.

Honeypot Type	**Commands logged**
AV	129
Banner	147
Banner-AV	100
Control	62

Data Analysis

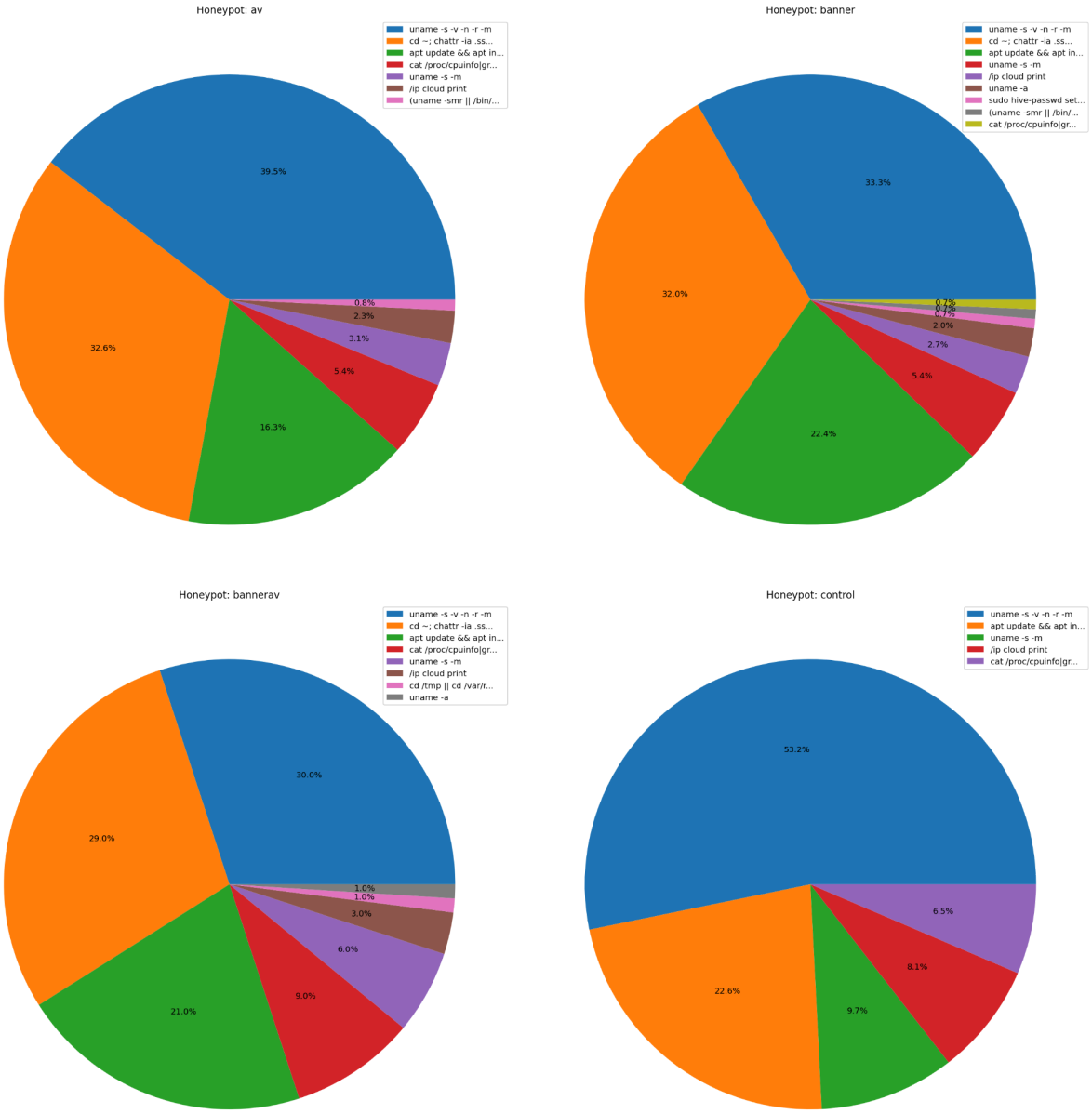
We first analyzed the relative frequencies of the commands entered by creating pie charts for each honeypot. Since we saw that the distributions of the commands were different, we created a contingency table and performed a Fisher's Exact Test to see if there was a significant association between the type of honeypot and the commands run on them. We did this test instead of a chi-squared test because some commands appeared a few times or not at all in some honeypots. The results of the Fisher's Exact Test gave us a p-value of 0.000500. Based on this p-value, we rejected the null hypothesis and have evidence supporting a significant association between the honeypot type and the commands.

Honeypot x Command Contingency Table

	uname -s -v -n -r -m	cd ~; chattr -ia .ssh; lockr -ia .ssh	apt update && apt install sudo curl -y…	uname -s -m	cat /proc/cpuinfo| grep name|cut -f2 -d':'|...
av	51	42	21	4	7
banner	49	47	33	8	1
bannerav	30	29	21	6	9

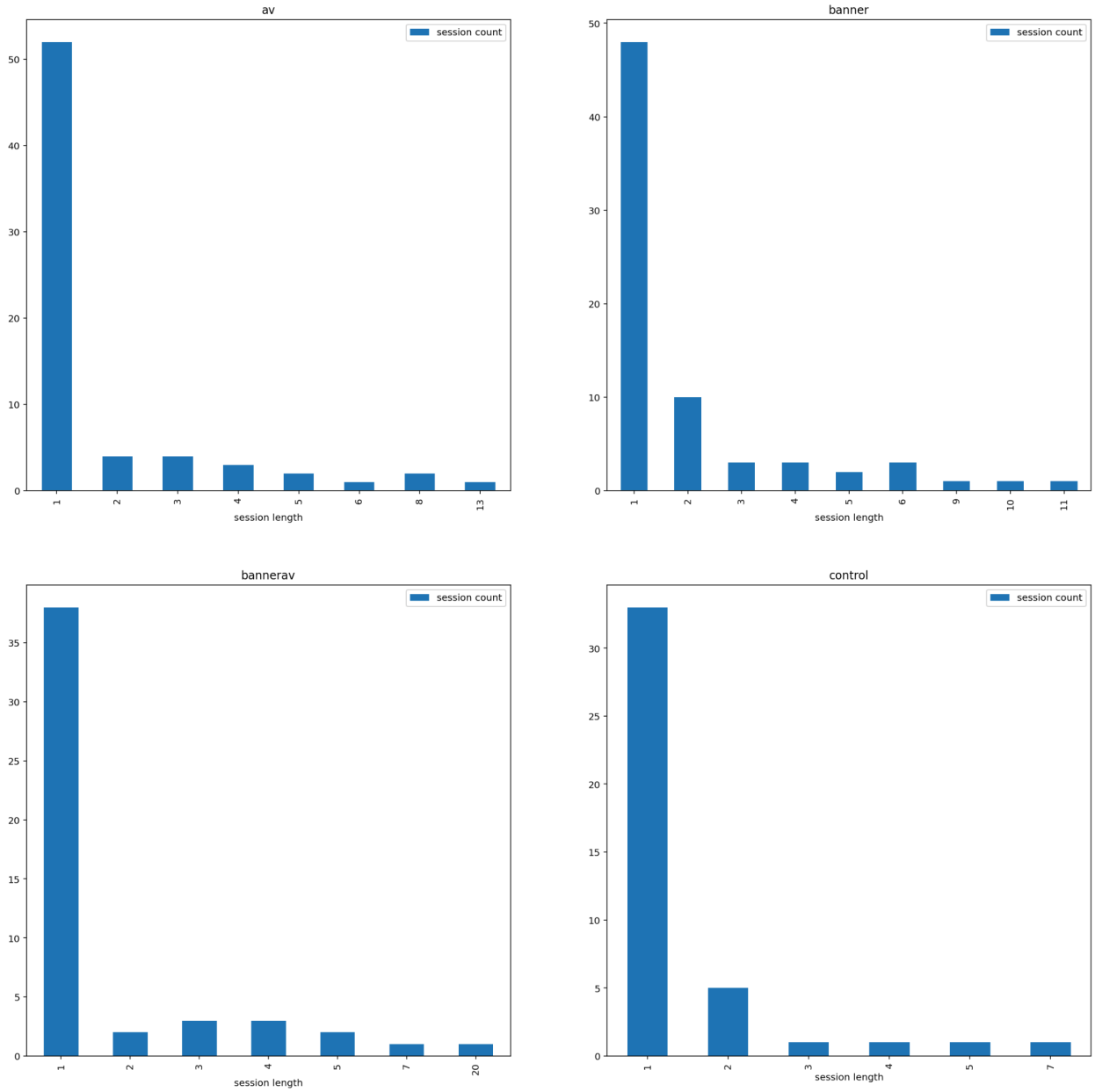
control	33	14		6	4

	/ip cloud print	uname -a	(uname -smr || /bin/uname -smr || …	sudo hive-passwd set ifjeeisurofmioufi…	cd /tmp || cd /var/run || cd /mnt || cd /root || …
av	3		1		
banner	4	3	1	1	
bannerav	3	1			1
control	5				



Then, after examining the session lengths, we performed a Kruskal-Wallis H-test on them to see if there was a significant difference in the median session duration between the honeypots. With a p-value of 0.480822, we failed to reject the null hypothesis, and cannot conclude that the median session length between the honeypots differs significantly.

Session length	**av**	**banner**	**bannerav**	**control**
1	52	48	38	33
2	4	10	2	5
3	4	3	3	1
4	3	3	3	1
5	2	2	2	1
6	1	3		
7			1	1
8	2			
9		1		
10		1		
11		1		
12				
13	1			
…	…	…	…	…
20			1	



An interesting result of our data analysis was finding that most attackers used the same few commands as soon as they entered the honeypot. Most of these commands involved either checking the specs of the machine they were on or installing new software to gain further access. In an actual attack, this could be the first step taken, and understanding their results could help defend against future actions.

Command	**Expected Result**	**Potential Purpose**
`uname -s -v -n -r -m / uname -a`	Prints the system information, specifically the kernel name, the release details, version info, nodename and machine hardware name.	Known vulnerabilities on old versions of software could be exploited. Knowing the details about the machine is also essential to being able to make use of it.
`cd ~; chattr -ia .ssh; lockr -ia .ssh`	After getting into the home directory, the *".ssh"* directory's attributes are changed so it can be modified by the attacker. *lockr* does something similar to chattr but is not valid in the LXC container environment.	The *.ssh* directory is central to the configuration of the ssh protocol. If the attacker modifies it they can change the authorized keys, public and private key, configuration file and known hosts. This could be used to plant a backdoor for future use.
`apt update && apt install sudo curl -y && sudo useradd -m -p $(openssl passwd -********) system && sudo usermod -aG sudo system`	After updating the existing packages, *sudo* and *curl* are installed with *apt.* Then the "system" user is created with a separate home directory and a password encrypted with *MD5 hash* through openssl. The "system" user is then added to the sudo user group.	This command is powerful because it adds a new sudo user to the system, allowing the attacker to have full access to the system if successful. This can be used as a backdoor. *curl* can be used to send out data to another device or server.
`cat /proc/cpuinfo|grep name|cut -f2 -d':'|uniq -c ; uname -a`	This prints information about the kind of processor, OS, kernel and version by printing a part of the *cpuinfo* file and by using the *uname* command to print all details of the system.	Knowing system info can clarify whether the attacker even wants to use the system. If it does not have good specs, it might not be of enough benefit to justify taking it over. Uname's use is as previously explained.

Command	Expected Result	Potential Purpose
`/ip cloud print`	This command is used in the MikroTik RouterOS to check if the cloud feature is enabled or disabled as well as details of the cloud service.	This would be useful for gaining info about cloud services of a MikroTik Router but is useless in this particular environment.

Conclusion

We observed from the Fisher Exact statistical test that there is a significant association between the honeypots and the types of commands used by attackers. From this, we conclude that having a banner and/or antivirus on a system does affect the types of commands an attacker may use. Interestingly, the `cd ~; chattr` command which we believe may allow return visits to the honeypot was never used on the control honeypot. This contradicts our theory that an attacker would be dissuaded from using a machine with an antivirus or antivirus warning on it. One possible cause for this trend could be that the presence of an antivirus or banner gave the impression of a legitimate machine. The attacker might assume that the user could change their password, so they wanted to create a backdoor in case that happened. Additionally, finding a machine named “control” with an easy password to get into may make attackers suspect they’ve found a honeypot since controls are typically used for experiments.

From the second statistical test done between session lengths of the different honeypots with the Kruskal-Wallis test, we found no correlation between the session lengths and which honeypot the attacker was on. We can infer that the configuration of banners or antivirus on the honeypot does not affect the length of an attacker’s session.

Our research was limited by the amount of time our containers were running. We believe that if we had more time to run these containers, we would have collected more concise results, leading to further, more concrete relationships.

As shown in the session length bar charts above, we observed that almost all of the attackers stopped using the honeypot after entering exactly one command, many of which were simply for getting system details. We found that this was far too little interaction to make any significant claims about. This might have been fixed with more time and data.

To remedy the above limitations and extend our work, in the future, we could look at comparing the behavior of attackers on honeypot systems with different types of antivirus installed on them over a longer period of time. This would not only give us more data but also a wider perspective of how different antivirus software can defend against malware. Additionally, since malware is most often found on MS Windows, it should be included in testing the effectiveness of antiviruses.

Appendix A

Throughout this project, we learned a lot about attacker behavior and how those behaviors impact our ability to do analysis. We learned that for most attackers, there is a fairly small set of first commands on which they can base the future progression of their attack. We also learned that we cannot accurately predict the behavior of attackers in such a short timeframe.

We found this project to be insightful into how systems can be compromised and what attackers do on compromised systems. We learned that when an attacker gains access to a target computer, they usually first create a backdoor into the system to be used later. They then often check the specifications of the system next, then leave.

From our hypothesis, we expected attackers' behavior to decrease between the honeypots without an antivirus and the honeypots with an antivirus. However, according to our data, attackers' behavior stayed mostly consistent between our four containers, (save for the control as discussed earlier). This did surprise us as we originally hypothesized that if an attacker detected an antivirus on the system, they would react differently. They even revisited the containers that had an antivirus installed, further reinforcing that they did not care.

A central problem of this experiment was the lack of time and hence data. An improvement in the speed of data collection might have been made if the system was Windows or there was better "honey" i.e. an underlying system with better hardware specifications.

Appendix B

Main crontab cycle script

```
#!/bin/bash


message="attacker in "

url='https://discord.com/api/webhooks/1212217598377463858/H0kD4liXihVI7RHDBv48UmEZ7F_KNklH5N0bojlvigOJbNi8E9C0bEucNhy3RDyHsEHS'


for hp in control av banner bannerav

do

      if [ -s /root/MITM_data/logins/$hp.txt ] && [ ! -f /root/flags/$hp ]; then

            echo "attacker in $hp"

            curl -H "Content-Type: application/json" -X POST -d '{"content":"'"${message}${hp}"'"}' $url

            screen -dmS ${hp}_cycle /root/scripts/hp_cycle.sh $hp

      fi

done
```

Honeypot cycle script

```
#!/bin/bash


wait="starting wait"

stop="starting lxc stop wait buffer"

recycle="recycling"

redeploy="redeploying"
```

```
url='https://discord.com/api/webhooks/1212217598377463858/H0kD4liXihVI7RHDBv48
UmEZ7F_KNklH5N0bojlvigOJbNi8E9C0bEucNhy3RDyHsEHS'


touch /root/flags/$1


curl -H "Content-Type: application/json" -X POST -d '{"content":"'"$1
${wait}"'"}' $url


sleep 4h


/root/scripts/mitm_stop.sh $1


/snap/bin/lxc stop $1


curl -H "Content-Type: application/json" -X POST -d '{"content":"'"$1
${stop}"'"}' $url


sleep 30s


cat /root/MITM_data/logins/$1.txt >> /root/data/$1_data/$1_logins.txt


rm /root/MITM_data/logins/$1.txt


curl -H "Content-Type: application/json" -X POST -d '{"content":"'"$1
${recycle}"'"}' $url


/snap/bin/lxc restore $1 ${1}snap
```

```
sleep 20s


/snap/bin/lxc start $1


/root/scripts/mitm_start.sh $1


curl -H "Content-Type: application/json" -X POST -d '{"content":"'"$1
${redeploy}"'"}' $url


rm /root/flags/$1
```

Honeypot reset script

```
#!/bin/bash


echo Removing logins and login_attempts

rm /root/MITM_data/logins/*

rm /root/MITM_data/login_attempts/*


echo Stopping MITM instances

/root/scripts/mitm_stop.sh av control banner bannerav


echo Stopping containers

lxc stop --all


echo Restoring from snapshots

for hp in av control banner bannerav ; do
```

```
      lxc restore $hp ${hp}snap
done


echo Starting containers
lxc start --all


echo Starting MITM instances
/root/scripts/mitm_start.sh av control banner bannerav
```

Works Cited

A. Algaith, I. Gashi, B. Sobesto, M. Cukier, S. Haxhijaha and G. Bajrami, "Comparing Detection Capabilities of AntiVirus Products: An Empirical Study with Different Versions of Products from the Same Vendors," 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), Toulouse, France, 2016, pp. 48-53, doi: 10.1109/DSN-W.2016.45.

Samociuk D. Antivirus Evasion Methods in Modern Operating Systems. Applied Sciences. 2023; 13(8):5083. https://doi.org/10.3390/app13085083/