Final Report

Antivirus Detection of Containerized Malware in Linux Distributions

CBER 710 - Capstone Project

Centennial College

School of Engineering Technology and Applied Science

To:

Dr. Sk Md Mizanur Rahman

On:

2024-04-16

Submitted by:

Student Name	Student ID
Oscar Cordoba	301214625
Krisuv Bohara	301274636
Rajan Boudel	301365245
Ampem Darko	301154778

Abstract

This project “Antivirus detection of containerized malware in Linux distributions” investigates the effectiveness of antivirus detection in identifying containerized malware within Linux distributions, using the Metasploit Framework to create and distribute a payload. The study thoroughly assesses the detection capabilities of four popular antivirus software packages, giving insight into their effectiveness in detecting dangerous content within Docker containers. Furthermore, the test involves setting up two Ubuntu OS instances using VirtualBox to imitate real-world conditions and improve the research's dependability.

Furthermore, the study investigates the anomaly detection system included in Docker images, which employs machine learning approaches to identify anomalies inside the dataset including Docker images. This study's machine learning approach includes use of Scikit-learn's Isolation Forest algorithm, as well as StandardScaler for data preparation. Isolation Forest effectively detects abnormalities in high-dimensional data, and StandardScaler enables optimal data standardization. These strategies improve the accuracy of anomaly detection in Docker settings, hence strengthening cybersecurity defenses against emerging threats.

The research's multifaceted approach seeks to provide comprehensive insights into the existing level of security measures and prospective areas for development in protecting containerized environments from malware attacks.

Table of Contents

Table of Figures

Table of Table

Introduction

Antivirus detection of containerized malware in Linux distributions is difficult due to the rapidly evolving nature of threats, the encapsulated architecture of containers limiting deep visibility, difficulties monitoring distributed environments, the high volume of container deployments, and the dynamic nature of container interactions, which increases attack surface. These difficulties need the development of more advanced detection methods, increased visibility, and tighter interface with container orchestration platforms to better containerized security.

Antivirus detection of containerized malware in Linux distributions is critical as containerization becomes more widespread, particularly in critical infrastructure environments. Given the widespread use of Linux servers to host critical services, effective detection mechanisms are essential for preventing data breaches and service disruptions. Furthermore, the dynamic nature of containers and their reliance on shared resources necessitate advanced antivirus solutions that can adapt to these specific challenges. Improving detection capabilities in containerized environments is critical for ensuring the integrity and security of digital ecosystems in the face of evolving cybersecurity threats. [1]

Antivirus detection of containerized malware in Linux distributions operates within a dynamic ecosystem shaped by the rise of containerization technology, such as Docker. This technology encapsulates applications and their dependencies into portable units, revolutionizing software deployment but also introducing security challenges. Traditional antivirus solutions, designed for host operating systems, face difficulties in adapting to the unique characteristics of containerized environments, including shared kernels and rapid container lifecycle changes. As a result, there's a growing need for antivirus solutions tailored specifically for container environments to effectively detect and mitigate malware threats. Understanding this background information is crucial for comprehending the complexities and significance of research efforts in this field. [2]

Project TimeLine

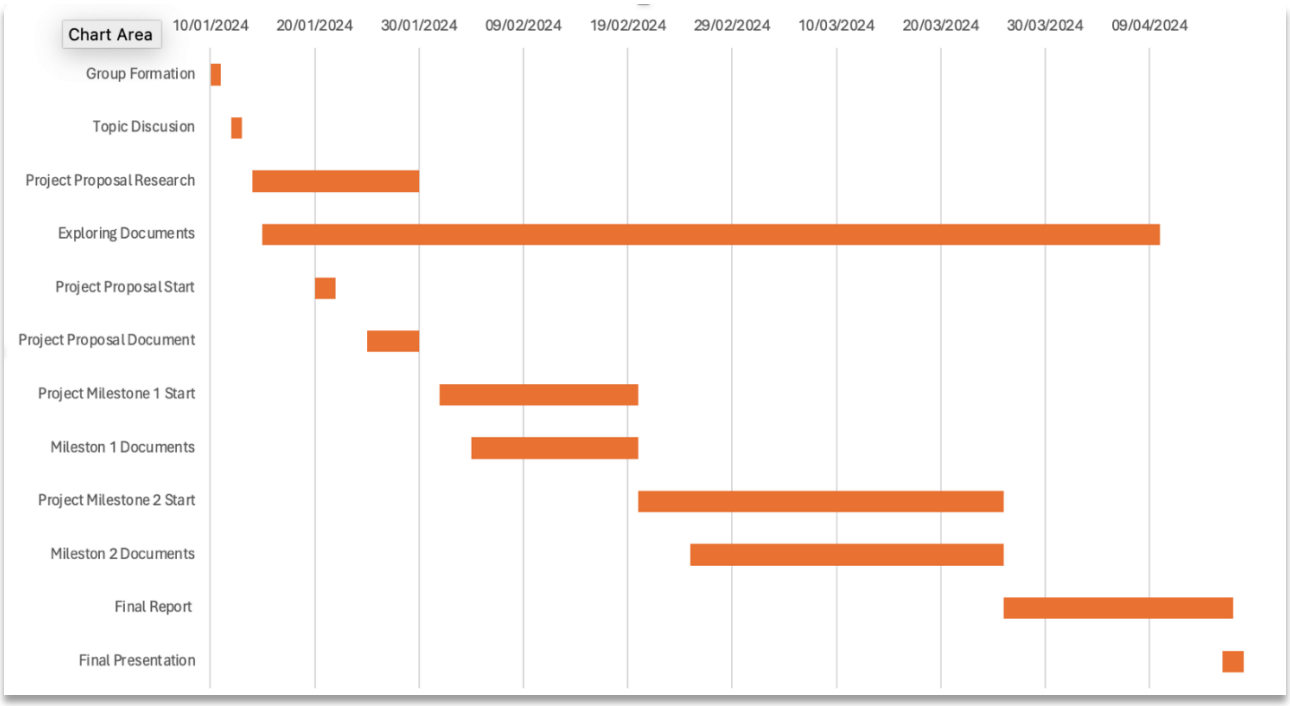

Figure 1. Gantt Chart

The diagram illustrates the division of the project into four stages: Topic Discussion, Project Proposal, Group Formation, and Project Execution. On January 10, 2023, the first stage, Group Formation, got underway. On January 20, 2023, the second phase, Topic Discussion, got underway. On February 9, 2023, the third phase Project Proposal began. Project Execution, the fourth phase, is slated to start on March 10, 2024.

Project Requirements

System Requirements for running 2 Linux OS in a Windows System

Requirements	Description
Host System	Windows Laptop / PC
Virtualization Software	Virtual Box
Hardware Requirements	Minimum 4 gb for each VBOX, 20 GB Space
OS	Linux
Installation Media	Linux ISO
Software Requirements	Virtual Box

Table 1 Requirements Table One

System Requirements for running Anaconda.

Requirements	Description
OS	Windows, MACOS, Linux
Processor	1.6hz or faster
RAM	Minimum 4gb,
Storage	3GB Minimum
Web Browser	Supported web browsers
Internet Connection	Required for packages

Table 2 Requirements Table Two

Project Architecture

Virtual Box Architecture

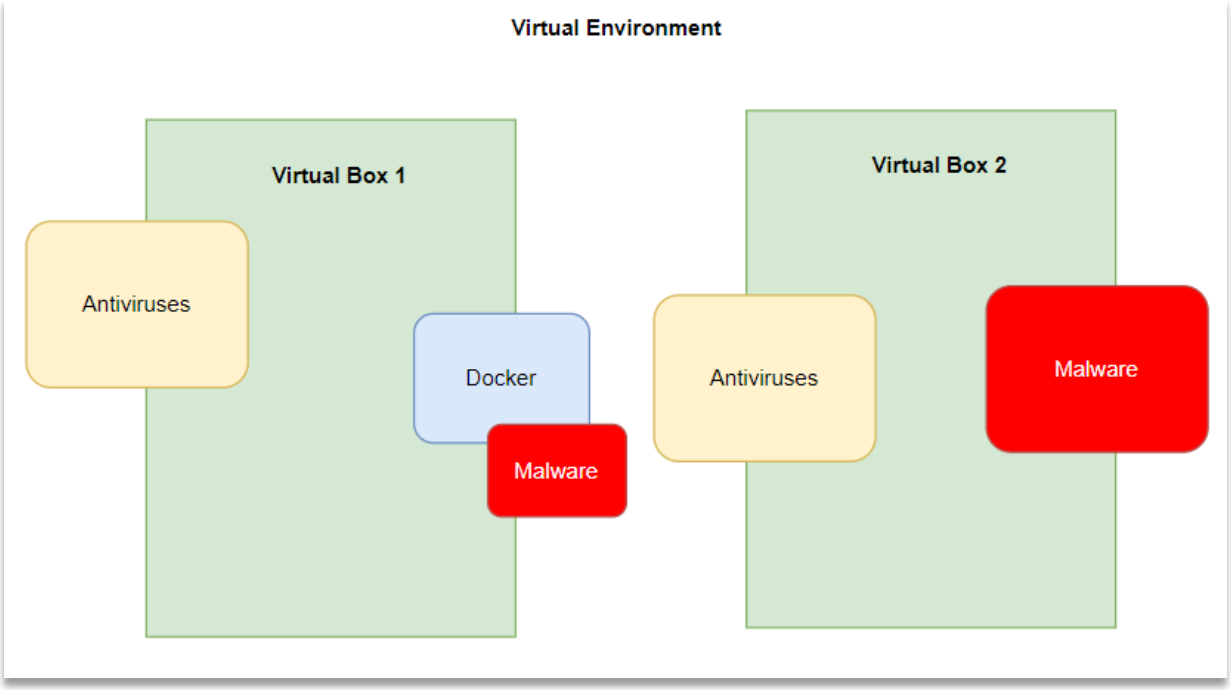

Figure 2. Visual Representation of Virtual Box

Antivirus detection of containerized malware in Linux distributions is critical for determining the security efficacy of containerization technologies. It aids in determining how well antivirus solutions detect and mitigate threats in isolated environments, ensuring that organizations can adequately protect their systems and data. This research helps to identify vulnerabilities, improve security practices, and remain compliant with regulatory standards. It also contributes to the development of proactive measures against emerging cybersecurity threats, thereby strengthening overall cybersecurity defenses in modern IT environments. Containerization is gaining popularity, but research into how well antivirus software detects malware in these environments is lacking. This gap creates a blind spot in our security. Studying this area is critical for staying ahead of evolving threats, improving current container antivirus solutions, and developing informed security practices for container-using organizations. [3]

Docker Architecture

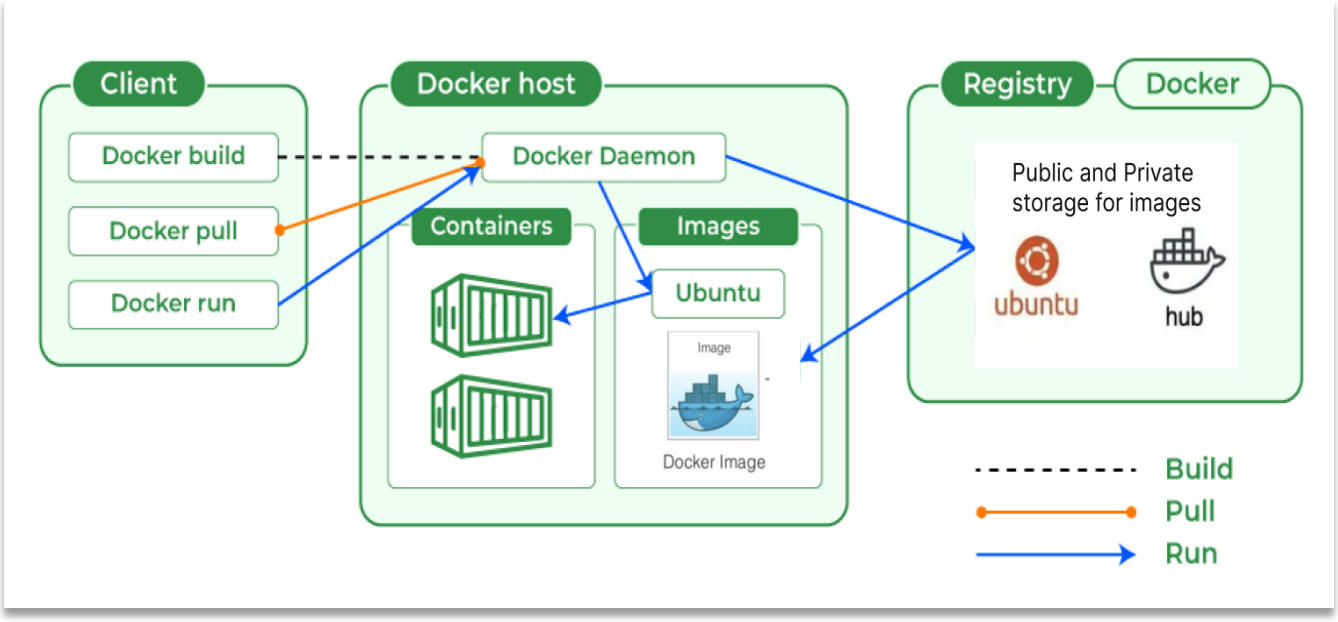

Figure 3. Docker Architecture

Docker containers are software modules that are self-contained and contain all necessary dependencies, making deployments consistent and portable. The Docker daemon and the client communicate via Docker commands and options. Building, executing, and maintaining Docker containers fall under the purview of the Docker daemon, a background application on a Docker host. A registry is a repository that holds Docker images, which users can pull to their Docker host. Public and private registries, like Docker Hub, are also options.

To build an image, first create a Docker file containing the assembly instructions for the application, and then use the docker build command to create the image. Using the docker run command to start an image's running instance is the process of running containers. The bottom section shows the lifecycle of a Docker image: "Build" generates a new Docker image, "Pull" gets an existing Docker image from the registry, and "Run" builds a Docker container from an Figure 4.

Approach 1

Creating a Payload

Approach Architecture

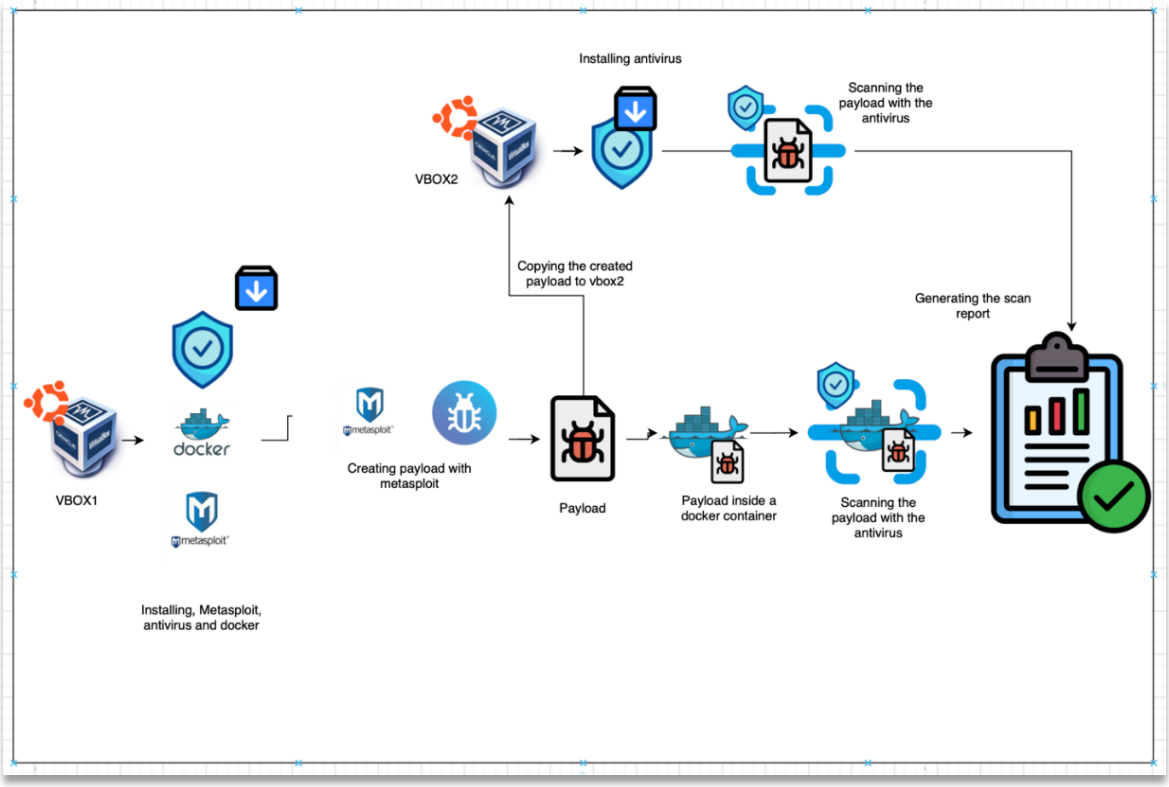

Figure 4. Approach 1 Architecture

We start our investigation by configuring two virtual machines running Ubuntu on our local system. We simulate real-world malicious activities by creating malware payloads using the Metasploit framework on one of these virtual machines. We then move these payloads to the second virtual machine so that we can examine them in more detail. To guarantee complete protection, we carefully install different antivirus software packages on each virtual machine. Interestingly, one of the virtual machines is the only one with Docker installed, which enhances our research by modelling containerized environments.

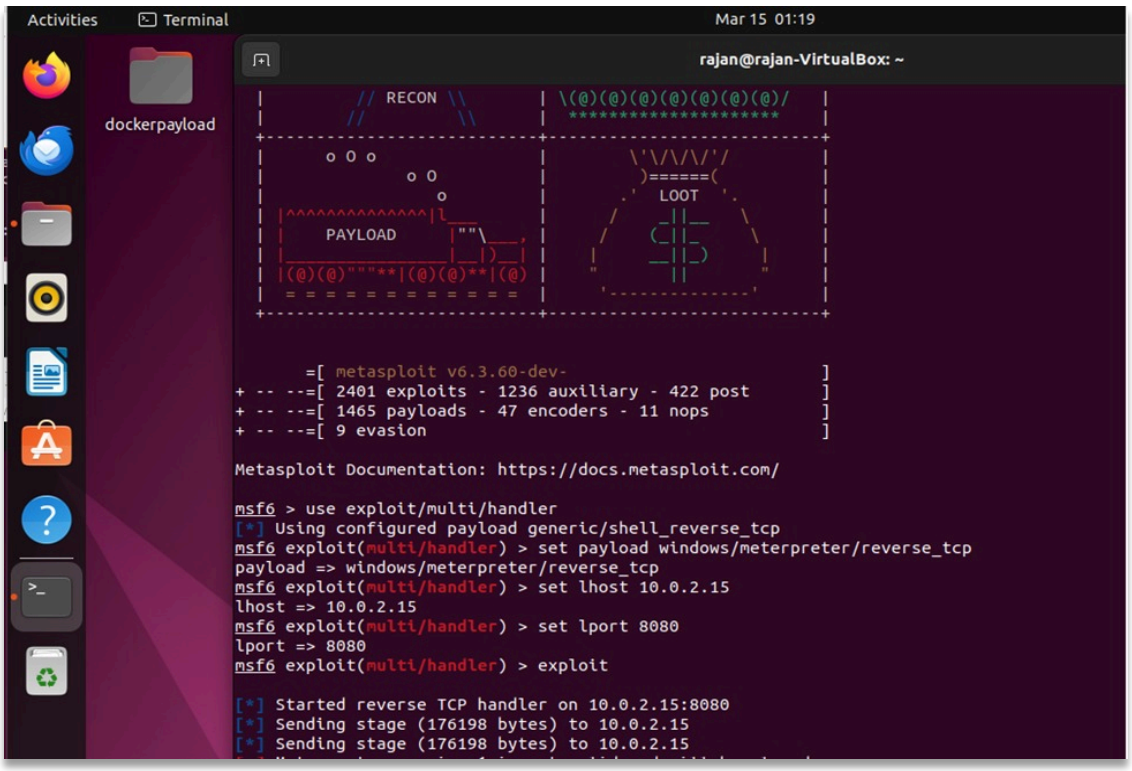

Figure 5. Running Metasploit

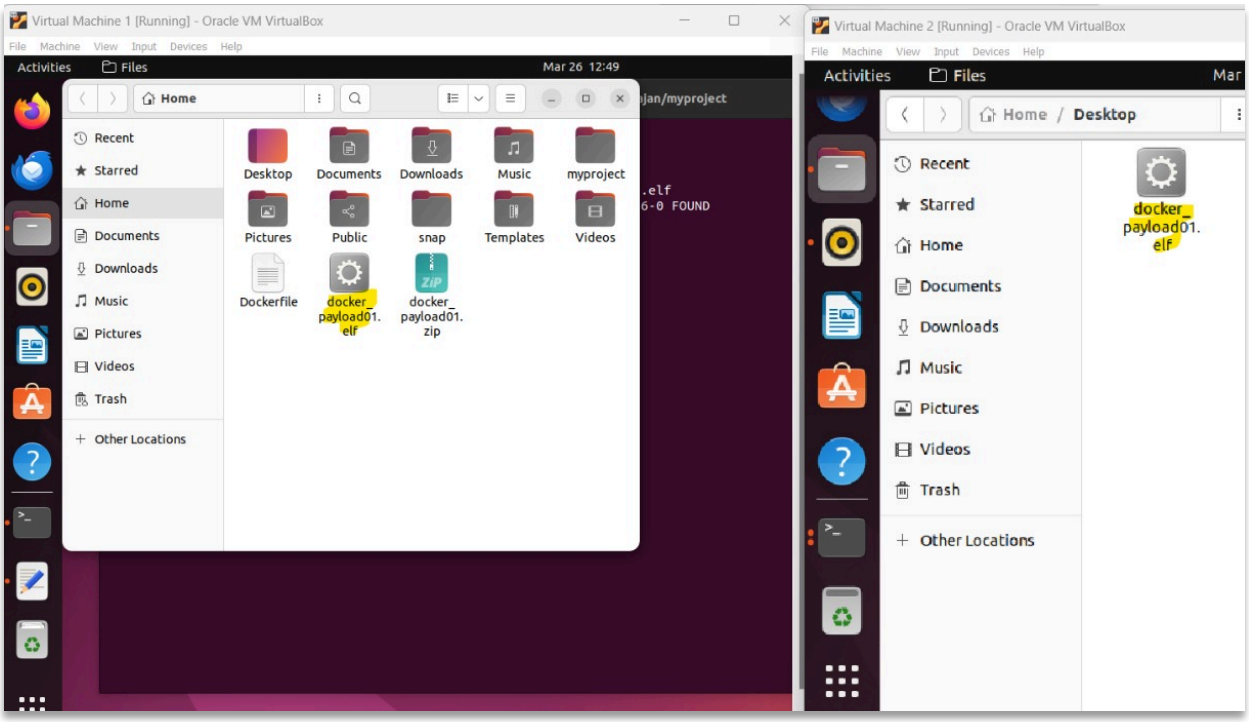

Figure 6. Infected file as a Docker Payload

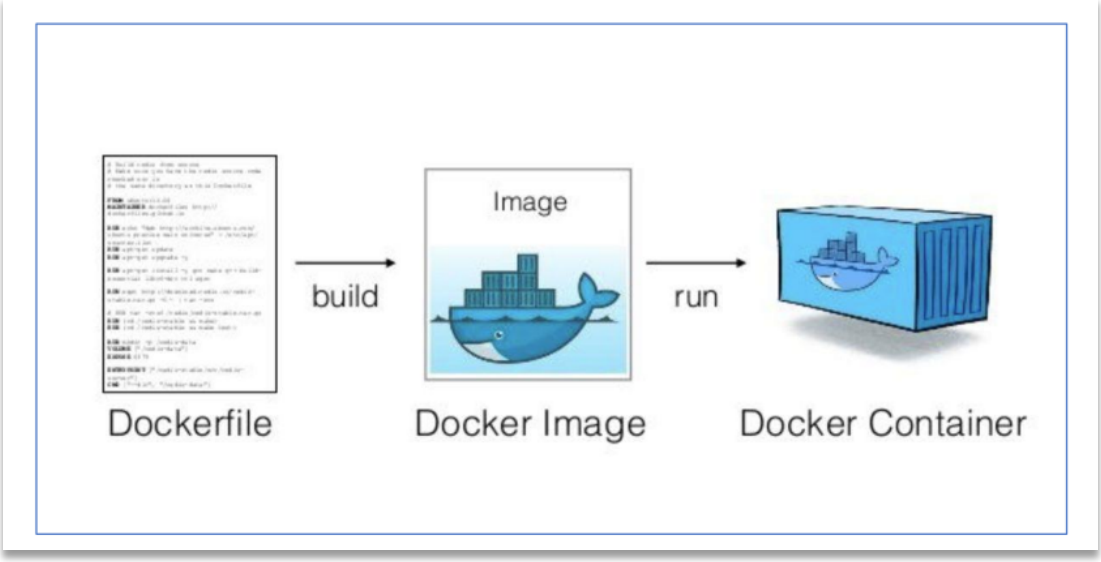

Figure 7. Docker architecture used.

After the setup is finished, we start the scanning procedure. We run all antivirus programs on the payloads in the virtual machine without Docker and examine the results of the detections. After this stage is over, we switch our attention to the virtual machine that has Docker installed. Here, we use a Docker container to encapsulate the payloads and run them through identical scans with the same antivirus programs. The subsequent examination of detection outcomes permits a comparative evaluation of antivirus effectiveness between environments that use Docker and those that do not, offering important insights into the security consequences of containerization.

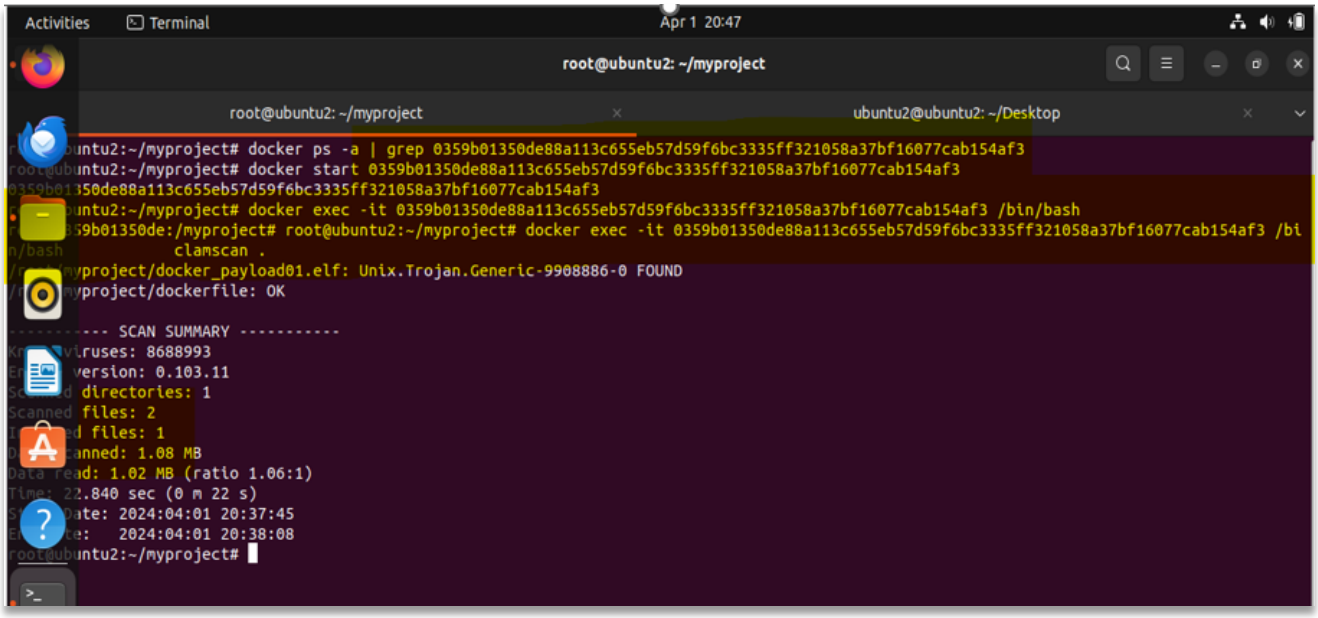

Figure 8. Running antivirus scan

Benefits

- **Real-world Simulation:** We simulate real-world malicious scenarios by using the Metasploit framework to create malware payloads, which makes our experiment more realistic.
- **Extensive Coverage:** Installing several antivirus programs guarantees a variety of detection techniques, producing outcomes that are more exhaustive.

Drawbacks:

- **Resource-intensive**: Conducting scans on both virtual machines using different antivirus programs may use a lot of computer power and extend the time needed for experimentation.
- **Restricted Scope**: Although our experimental setup sheds light on antivirus detection in Docker environments, it might not fully account for the subtleties and complexities of real-world situations, which could limit the generalizability of the results.

Approach 2

Downloading a Payload

Approach Architecture

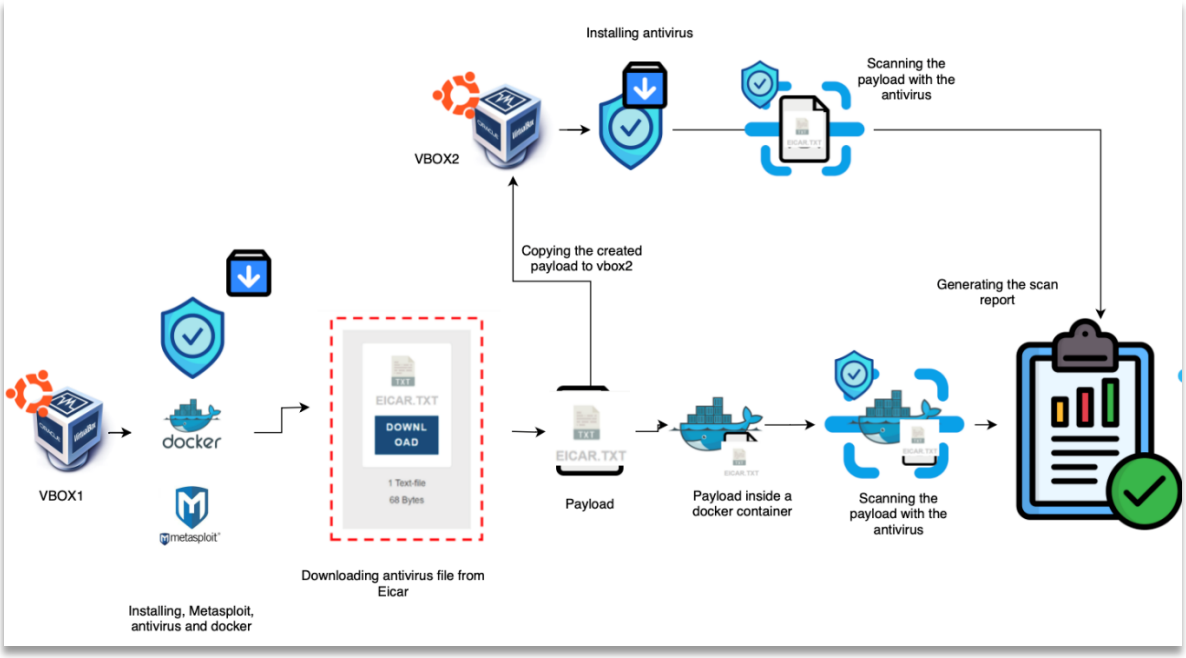

Figure 9. Approach Architecture

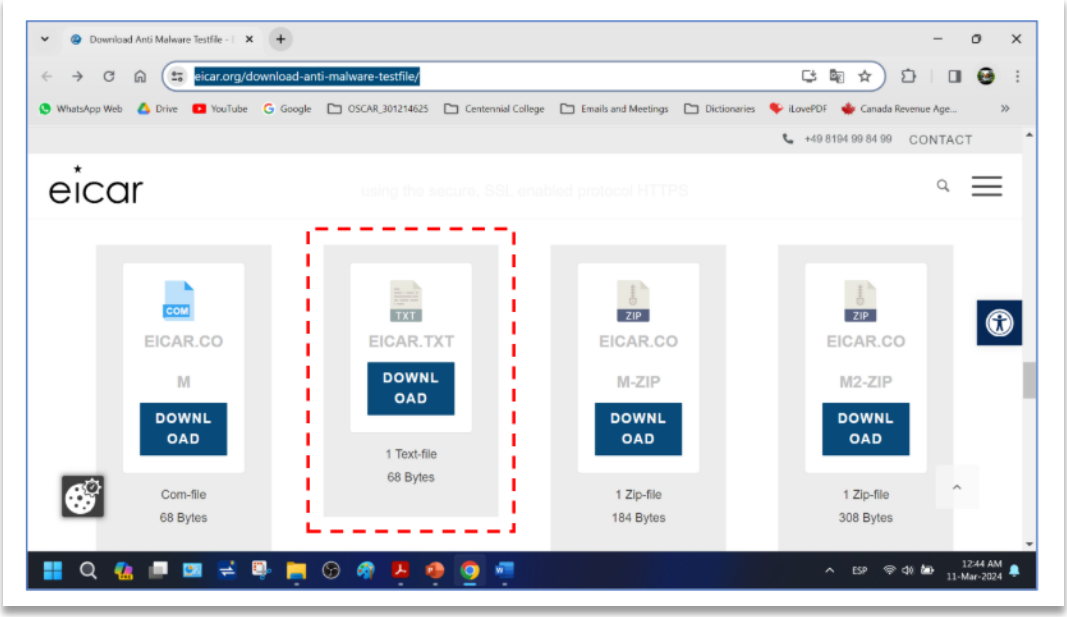

Figure 10. EICAR.txt malware file used as a test for this example

We set up two Ubuntu virtual machines on our system to start our investigation. The European Institute for Computer Antivirus Research (EICAR) provides us with an antivirus test file after that. On both virtual machines, we install four different antivirus software packages. Docker is exclusively installed on one of the virtual machines in parallel. After every installation is complete, the scanning procedure starts. First, on the virtual machine without Docker, we run scans on the payload and carefully review the results that each of the antivirus programs produces.

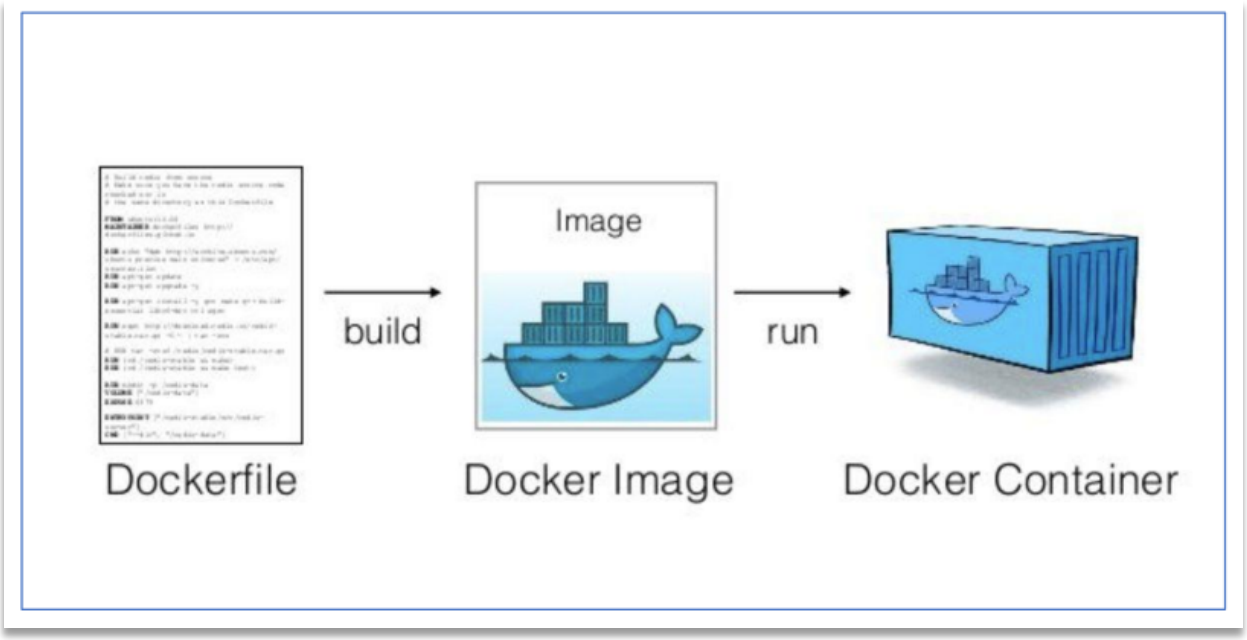

Figure 11. Docker Architecture Used

Additionally, Docker is only installed on one of the virtual machines. Once all installations are completed, we begin the scanning process. In the virtual machine without Docker, we scan the payload and compare the results from all four antivirus programs. Following the completion of the scan on the first virtual machine, we shift our attention to the virtual machine with Docker. We integrate the payload into a Docker container and run scans with all four antivirus programs. We then analyze the results from each antivirus program.

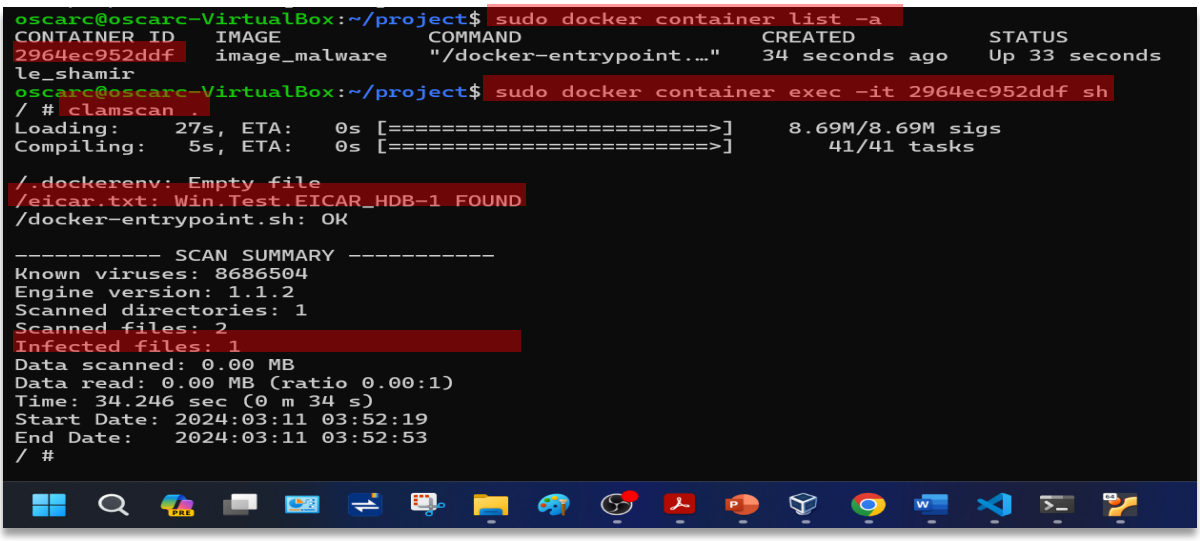

Figure 12. Running antivirus on the 2964ec952ddf container in the VM

After the scan on the first virtual machine is finished, we turn our attention to the virtual machine that contains Docker. Here, we use the same antivirus programs to perform scans after encasing the payload inside a Docker container. Next, we perform a comprehensive analysis of the outcomes provided by every antivirus programmed. This painstakingly crafted test configuration makes it easier to compare the effectiveness of antivirus detection in Docker and non-Docker environments.

Benefits

- **Controlled Comparison:** The study ensures a controlled comparison by running scans on identical virtual machine configurations in both Docker and non-Docker environments. This minimizes confounding variables and makes it easier to accurately assess the efficacy of antivirus detection.
- **Practical Relevance:** By reflecting real-world cybersecurity scenarios, the EICAR antivirus test file and Docker containers increase the study's practical relevance and applicability.

Drawbacks:

- **Restricted Scope**: The study may not cover other facets of containerized malware detection, such as runtime behavior analysis or network-based detection mechanisms, as it primarily focuses on antivirus detection efficacy.

- **Static Payload:** The results of the study may not be as broadly applicable if a single predefined payload is used, as this may not accurately reflect the variety of malware that is encountered in real-world situations.

Approach 3

Anomaly Detection using Isolation Tree Model

By effectively detecting abnormalities within the containerized environment, the Isolation Tree Model aids in the detection of containerized malware by antivirus software in Linux distributions using Docker images. The model is capable of efficiently isolating and identifying suspicious behavior suggestive of malware presence using isolation trees, which divide data into increasingly isolated subsets. This strategy is especially helpful in containerized environments, such as Docker, where the dynamic and encapsulated nature of containers may make traditional antivirus solutions ineffective. Because isolation trees are so good at identifying anomalies and abnormalities, antivirus software can identify possible dangers in Docker images. Thus, adding the Isolation Tree Model to Linux distributions with Docker deployments strengthens overall security measures by improving antivirus software's detection capabilities. [4]

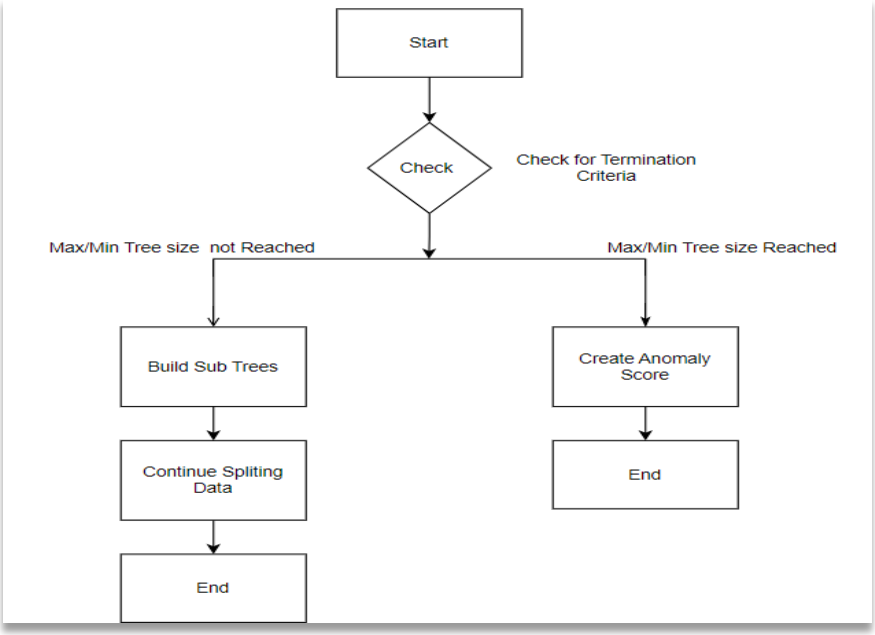

Figure 13. Flow Chart Isolation Tree Model

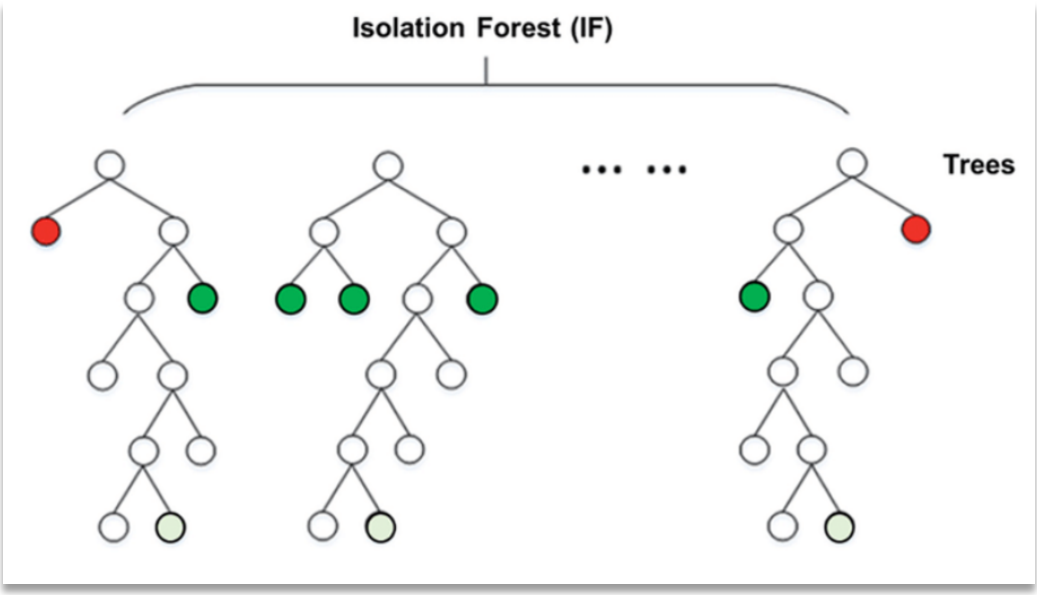

Figure 14. Isolation Forest Scheme

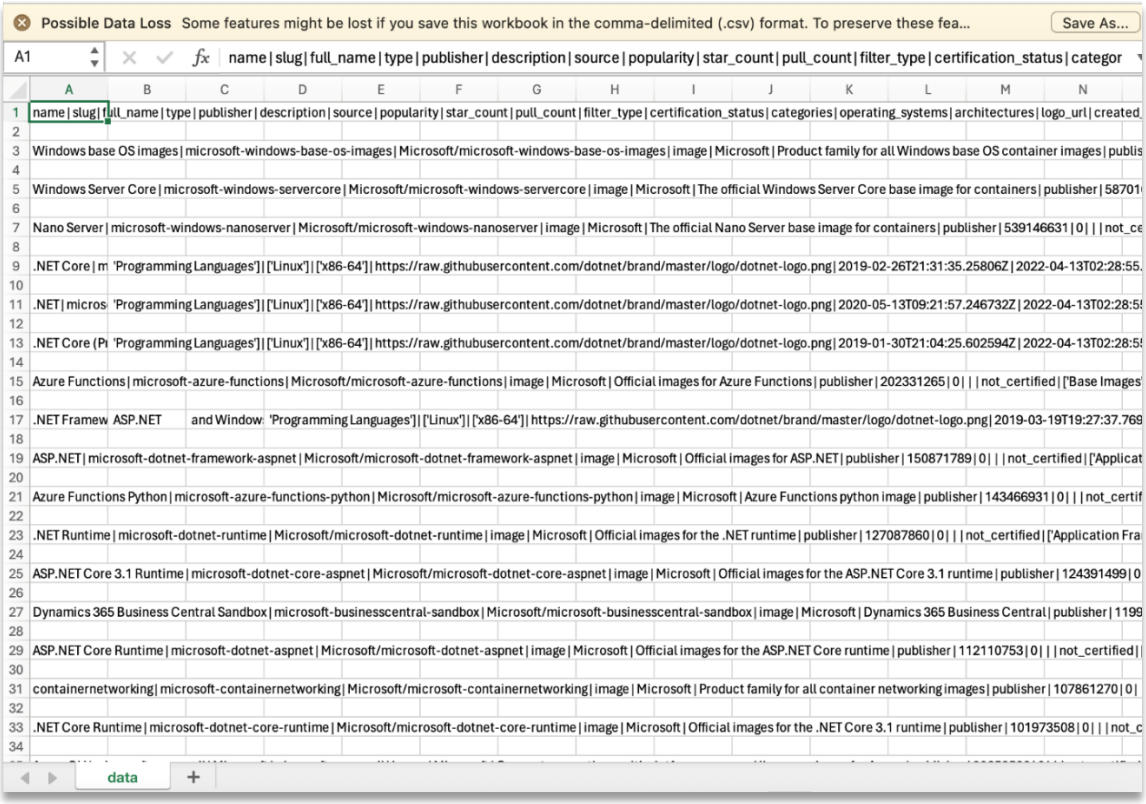

Figure 15. Docker Image Dataset

We carefully verified that the dataset we used for our analysis was in line with our research goals before beginning our analysis. After that, we started a thorough preprocessing

journey using a variety of methods, such as meticulous data cleaning, normalization, and thoughtful feature engineering. Ensuring the quality and appropriateness of the dataset for further analysis required this preparatory stage. After preprocessing, we used the Isolation Forest model to fully utilize machine learning. After a thorough training process on the improved dataset, the model was expertly adjusted to detect anomalies. By utilizing its innate capacity to distinguish abnormalities from the rest of the data, the Isolation Forest model proved to be a reliable instrument in our toolbox for identifying anomalies and outliers. [5]

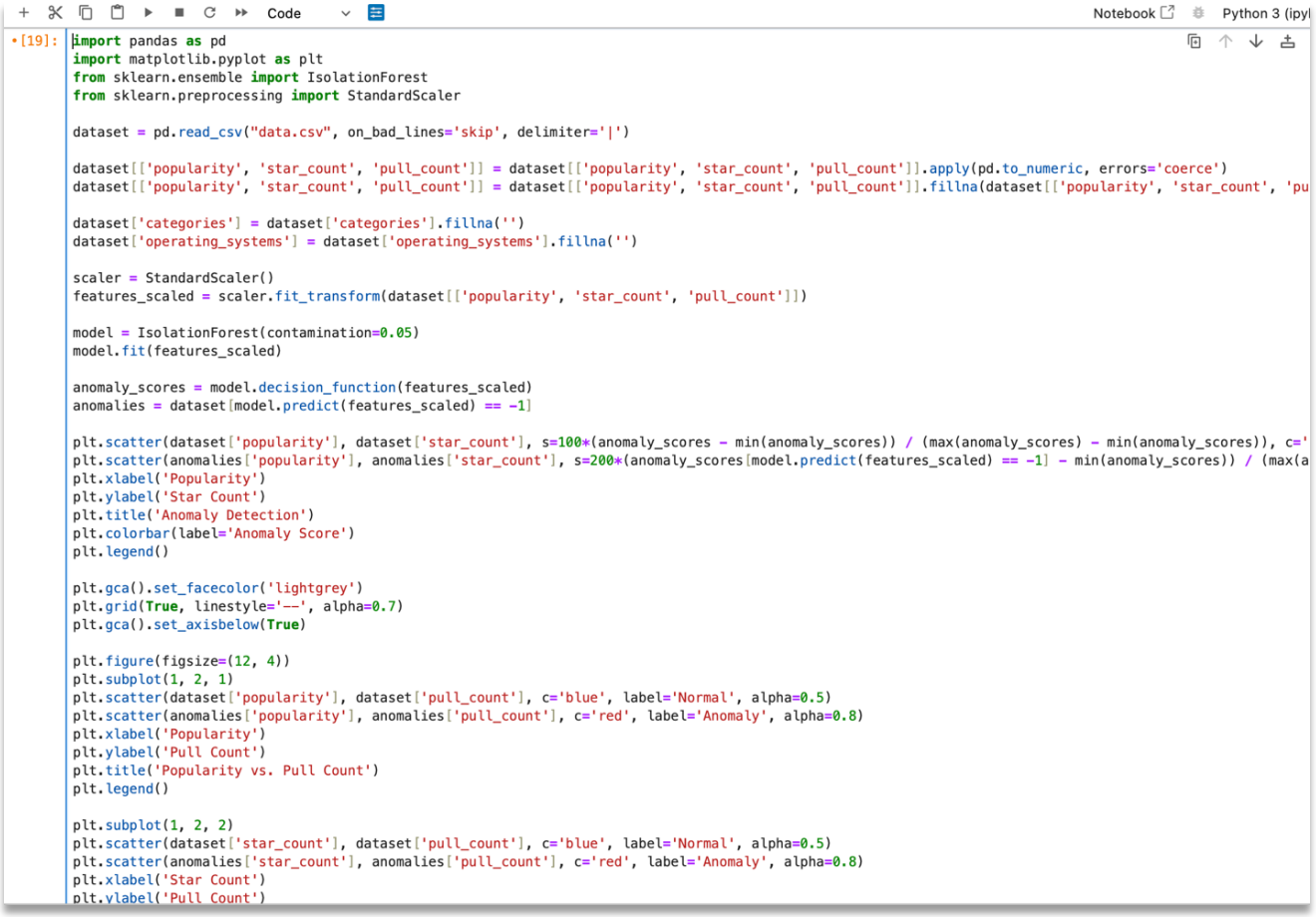

Figure 16. Coding Jupyter Notebook Anomaly detection

To gain additional insights and facilitate data interpretation, we used the StandardScaler data visualization tool to visually represent the dataset, allowing us to better understand the underlying patterns and distributions.

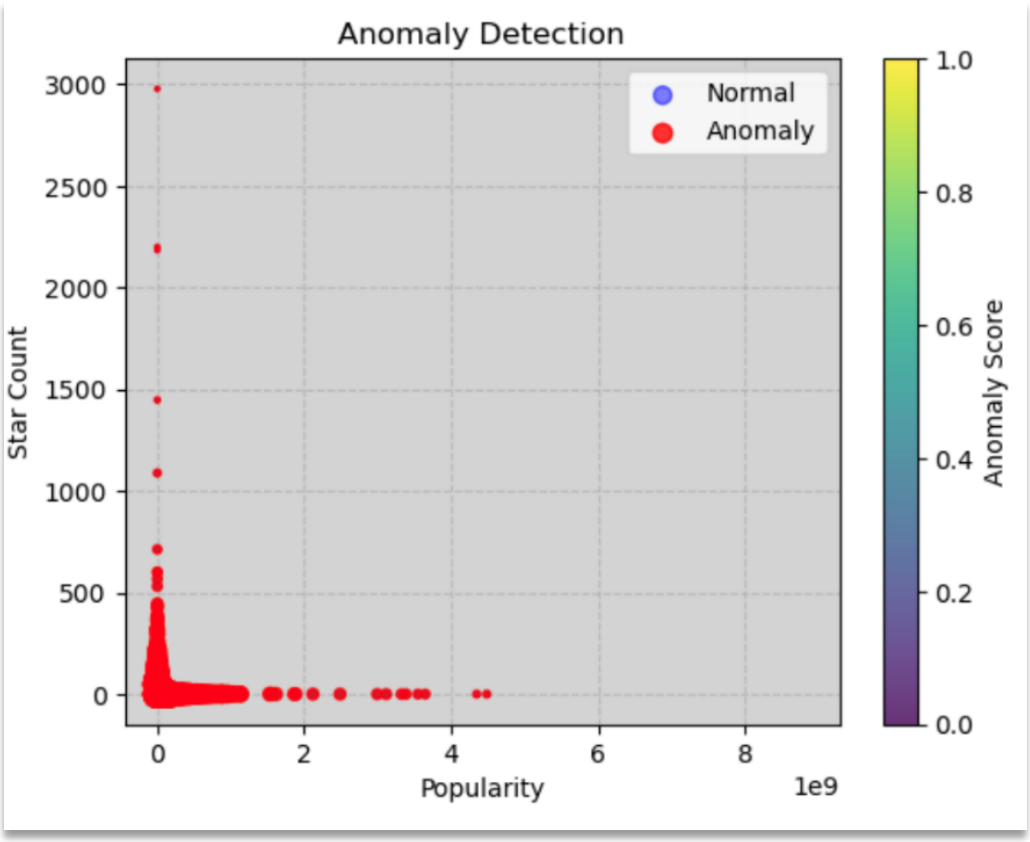

Figure 17. Visual Representation of anomalies

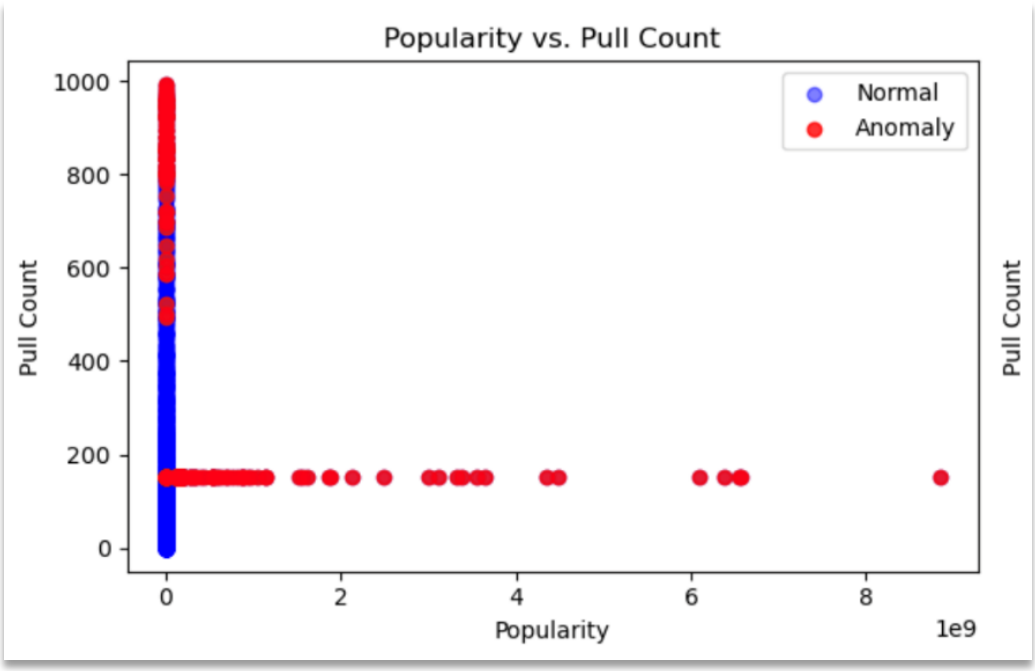

Figure 18. Popularity vs Pull Count

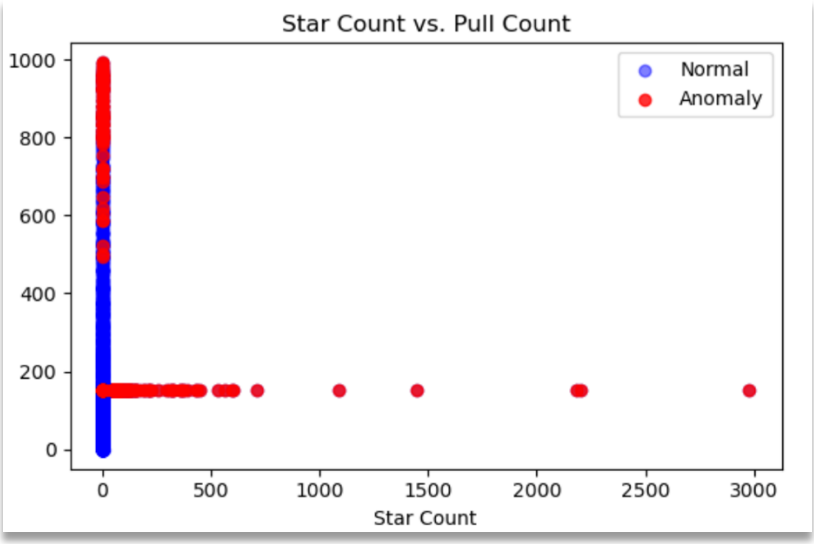

Figure 19. Star Count Vs Pull Count

Finally, to preserve our findings and allow for future analysis, we archived the generated report or data in a structured format, such as a CSV file.

	name	slug	full_name	type	publisher	description	source	popularity
1	Windows base OS images	windows-base-os-images	windows-base-os-images	image	Microsoft	ase OS container images	publisher	1134293415
2	Windows Server Core	osoft-windows-servercore	osoft-windows-servercore	image	Microsoft	ase image for containers	publisher	587010244
3	Nano Server	soft-windows-nanoserver	soft-windows-nanoserver	image	Microsoft	ase image for containers	publisher	539146631
4	.NET Core	microsoft-dotnet-core	soft/microsoft-dotnet-core	image	Microsoft	.1 and ASP.NET Core 3.1	publisher	279330629
5	.NET	microsoft-dotnet	Microsoft/microsoft-dotnet	image	Microsoft	.NET and ASP.NET Core	publisher	258250089
6	.NET Core (Preview)	rosoft-dotnet-core-nightly	rosoft-dotnet-core-nightly	image	Microsoft	.1 and ASP.NET Core 3.1	publisher	207260057
7	Azure Functions	microsoft-azure-functions	microsoft-azure-functions	image	Microsoft	ages for Azure Functions	publisher	202331265
8	.NET Framework	icrosoft-dotnet-framework	icrosoft-dotnet-framework	image	Microsoft	ication Framework (WCF)	publisher	166458767
9	ASP.NET	dotnet-framework-aspnet	dotnet-framework-aspnet	image	Microsoft	ficial images for ASP.NET	publisher	150871789
10	Azure Functions Python	ft-azure-functions-python	ft-azure-functions-python	image	Microsoft	e Functions python image	publisher	143466931
11	.NET Runtime	microsoft-dotnet-runtime	t/microsoft-dotnet-runtime	image	Microsoft	ages for the .NET runtime	publisher	127087860
12	SP.NET Core 3.1 Runtime	rosoft-dotnet-core-aspnet	rosoft-dotnet-core-aspnet	image	Microsoft	.SP.NET Core 3.1 runtime	publisher	124391499
13	usiness Central Sandbox	businesscentral-sandbox	businesscentral-sandbox	image	Microsoft	ics 365 Business Central	publisher	119995238
14	ASP.NET Core Runtime	microsoft-dotnet-aspnet	ft/microsoft-dotnet-aspnet	image	Microsoft	e ASP.NET Core runtime	publisher	112110753
15	containernetworking	osoft-containernetworking	osoft-containernetworking	image	Microsoft	tainer networking images	publisher	107861270
16	.NET Core Runtime	osoft-dotnet-core-runtime	osoft-dotnet-core-runtime	image	Microsoft	he .NET Core 3.1 runtime	publisher	101973508
17	alpine	alpine	Docker/alpine	image	Docker	lex and only 5 MB in size!	library	883679323
18	nginx	nginx	Docker/nginx	image	Docker	Official build of Nginx.	library	6564206660
19	ubuntu	ubuntu	Docker/ubuntu	image	Docker	n based on free software.	library	6550308885
20	busybox	busybox	Docker/busybox	image	Docker	Busybox base image.	library	6376492961
21	python	python	Docker/python	image	Docker	e programming language.	library	6096661402
22	postgres	postgres	Docker/postgres	image	Docker	liability and data integrity.	library	4483544973
23	redis	redis	Docker/redis	image	Docker	s a data structure server.	library	4345411211
24	httpd	httpd	Docker/httpd	image	Docker	che HTTP Server Project	library	3646315686
25	node	node	Docker/node	image	Docker	networking applications.	library	3544696705
26	mongo	mongo	Docker/mongo	image	Docker	bility and easy scalability.	library	3381970322
27	mysql	mysql	Docker/mysql	image	Docker	ement system (RDBMS).	library	3321173132
28	memcached	memcached	Docker/memcached	image	Docker	ry object caching system.	library	3113547451
29	traefik	traefik	Docker/traefik	image	Docker	Cloud Native Edge Router	library	2991055626
30	mariadb	mariadb	Docker/mariadb	image	Docker	ase, forked from MySQL.	library	2486468055
31	docker	docker	Docker/docker	image	Docker	Docker in Docker!	library	2117191777
32	rabbitmq	rabbitmq	Docker/rabbitmq	image	Docker	rotocol messaging broker.	library	1882363942

Figure 20. Final Dataset

In addition to the benefits already listed, this strategy also has the following two benefits:

- **Flexibility**: The Isolation Forest model is comparatively unaffected by the presence of unimportant features and can handle high-dimensional data. This adaptability eliminates the need for intensive feature engineering and enables effective anomaly detection across a variety of datasets.

- **Scalability**: The Isolation Forest model demonstrates scalable performance due to its innate ability to partition data into isolated subsets. This feature makes it appropriate for handling sizable datasets that are frequently encountered in real-world scenarios.

It's crucial to consider any potential drawbacks, though:

- **Sensitivity to contamination level:** If the number of anomalies in the dataset (i.e., the contamination level) is excessively high, the Isolation Forest model's performance may suffer. In these situations, it might be essential to adjust the model's parameters or investigate different anomaly detection methods.
- **Interpretability issues:** Although the model detects anomalies well, it may be difficult to understand the underlying causes of their classification. It could be necessary to have more contextual knowledge and domain expertise to comprehend the context and implications of detected anomalies.

Chapter 3: Project Process

Installing Metasploit Framework

- Creation of payload on one VMware using Metasploit framework

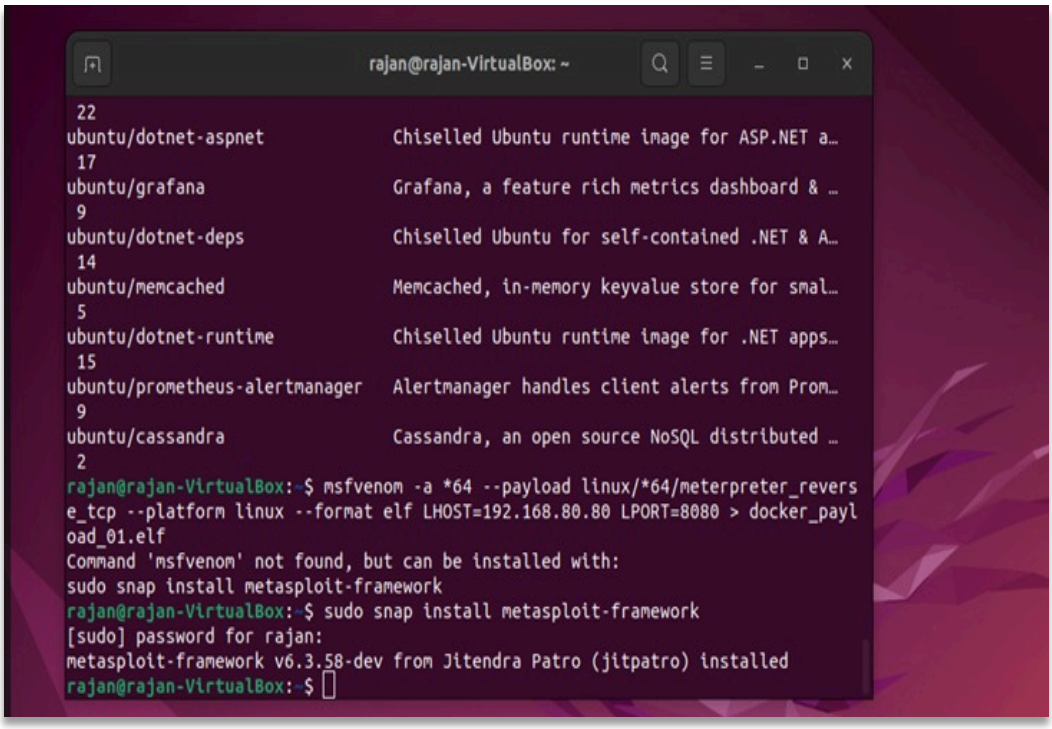

Figure 21 Metasploit Framework

- Running the Metasploit framework

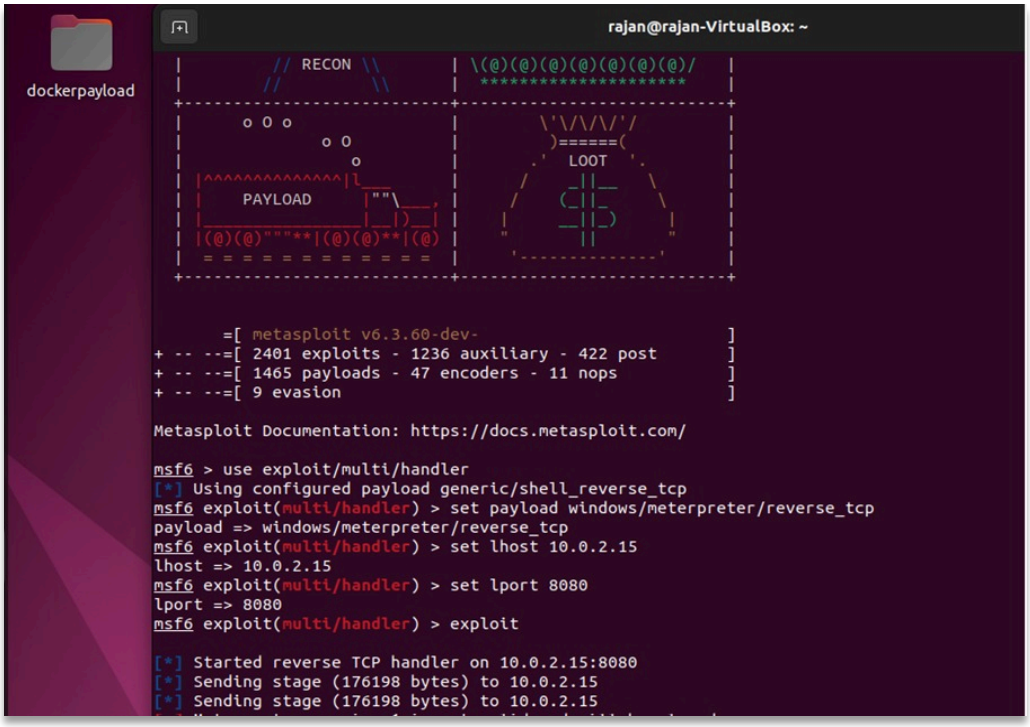

Figure 22 Metasploit Framework

- Payload creation

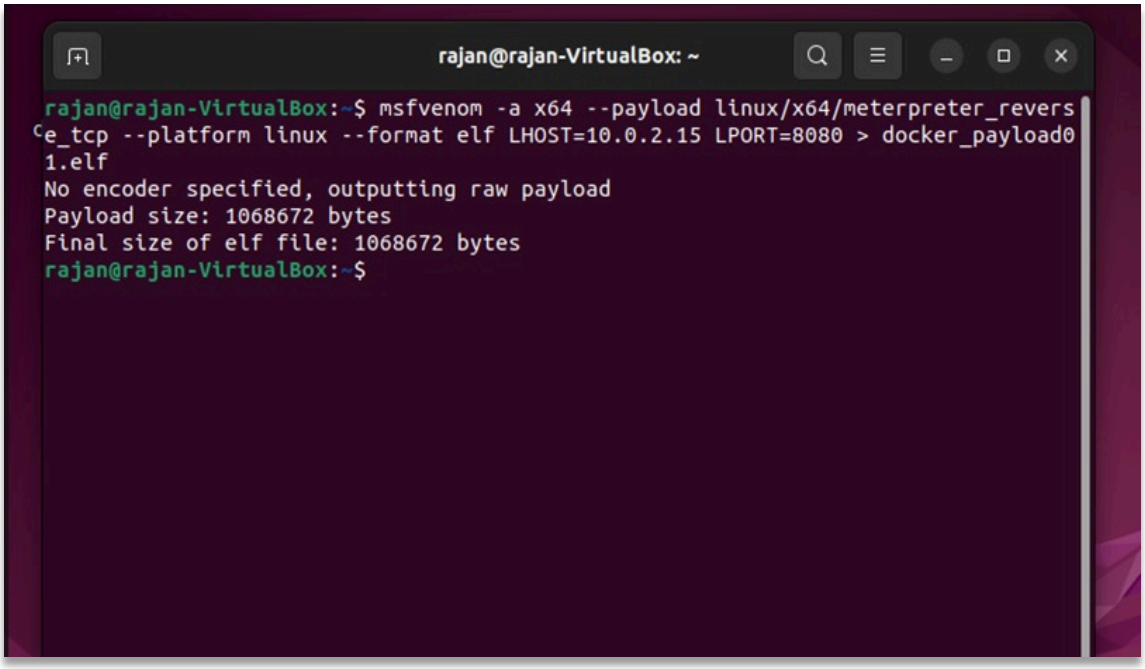

Figure 23 Payload Creation

- Creating the payload docker_payload01.elf in both the virtual machines.

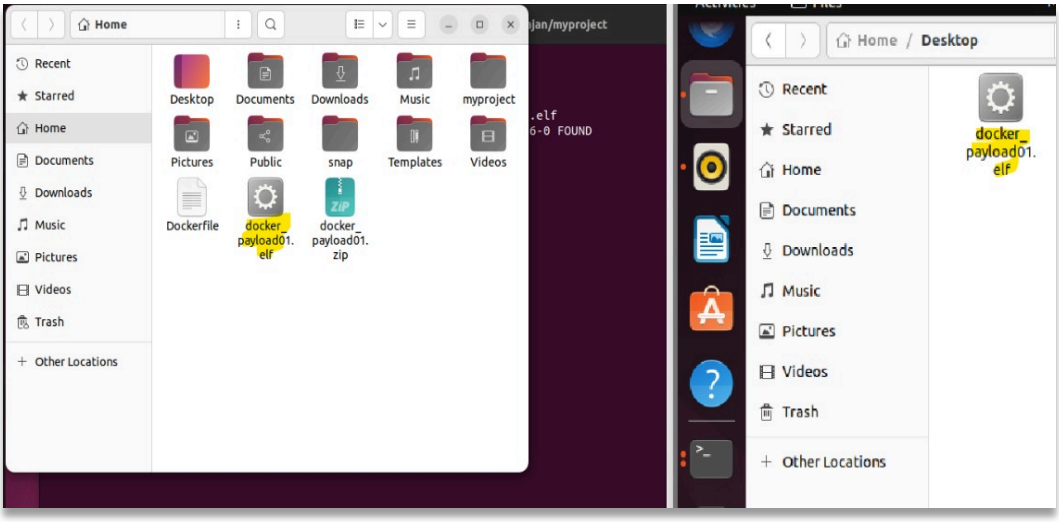

Figure 24 Payload Display

Installing the Docker on One of the Virtual Box

- Virtual machine1

Figure 25 Docker Version Check

Installing the clamscan on both the virtual machine

- Screenshot of clamscan installation on virtual machine 1

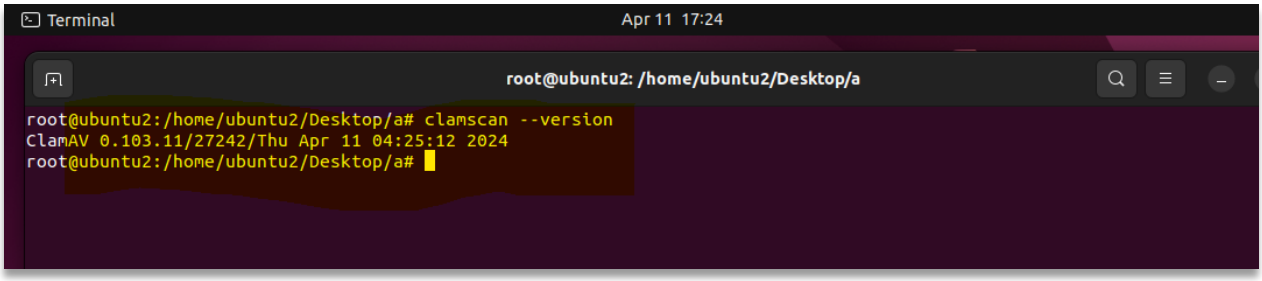

Figure 26 Clamscan Version Check

- Scanning the payload file outside the container within vm1

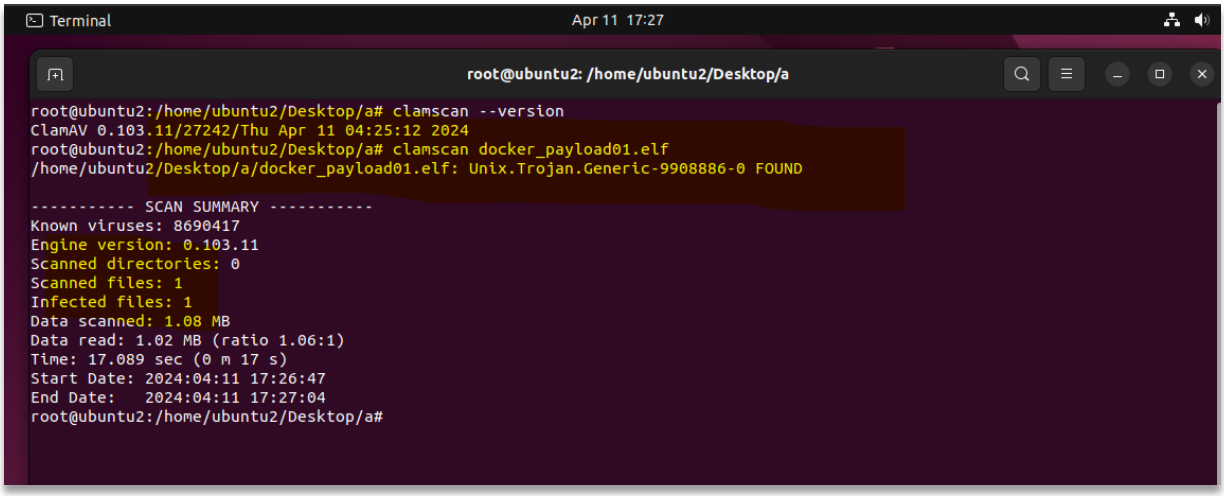

Figure 27 Scanning clamscan

- Creating a docker file for clamscan

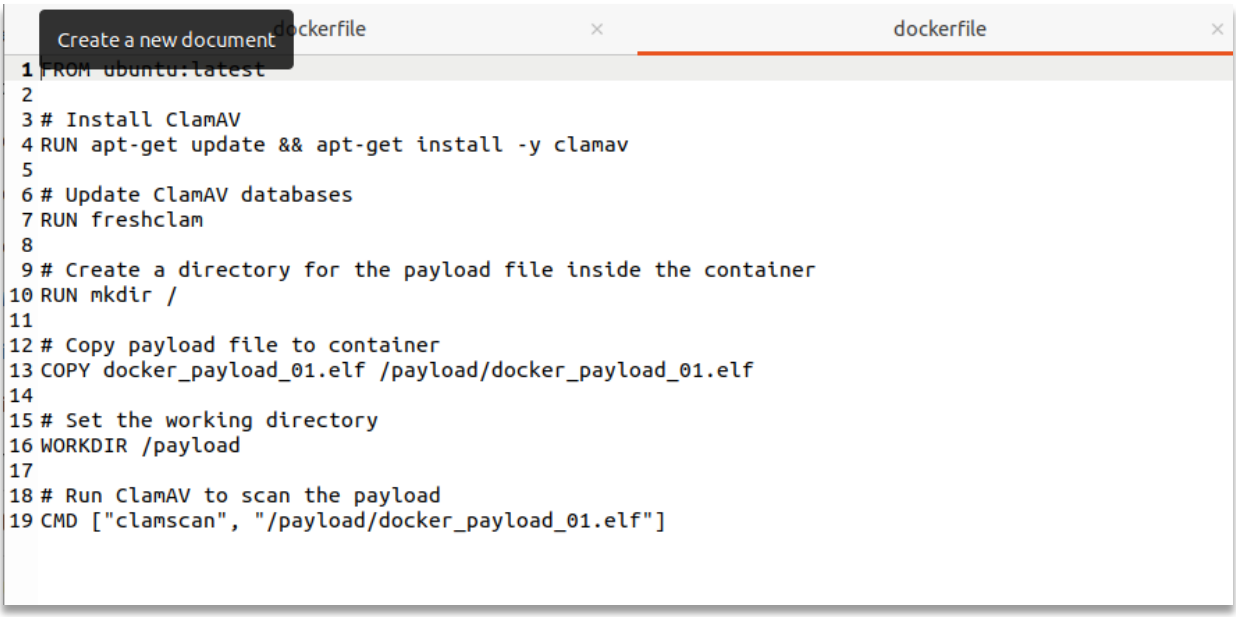

Figure 28 Docker File

- Now after the payload file is stored in the docker and run the docker container. Here, we have created a list of containers regarding the same payload file with each unique container id for each built in image inside the docker.

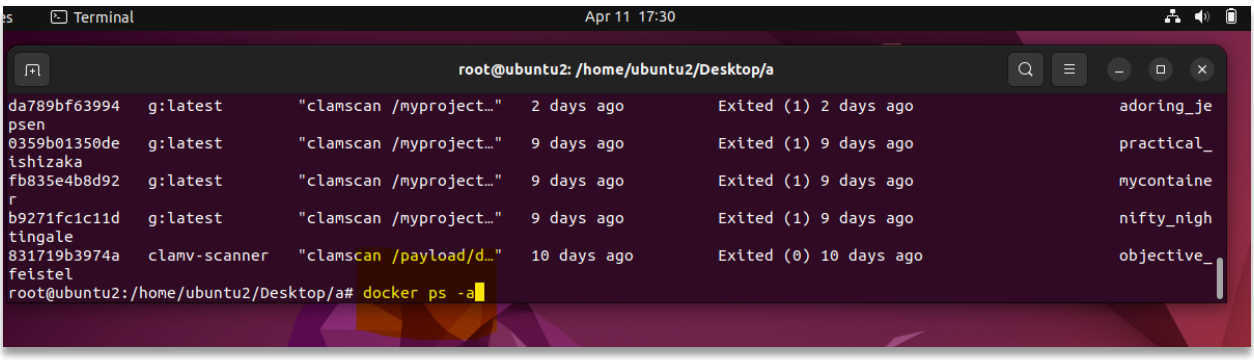

Figure 29. Docker containers

- Starting the container

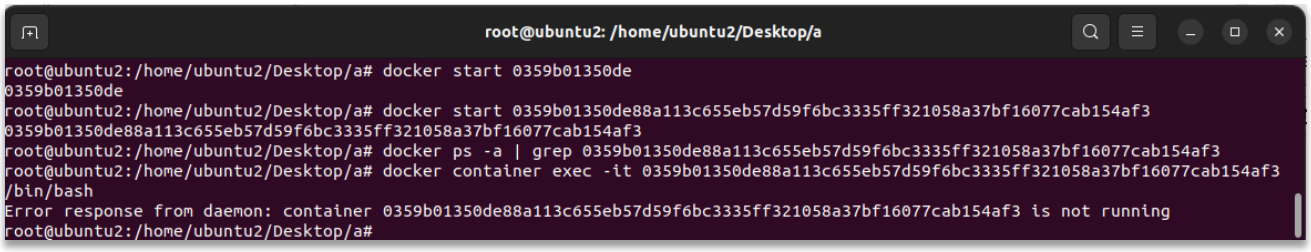

Figure 30. Starting Container

- Scanning the file using clamav on virtualmachine1 with docker

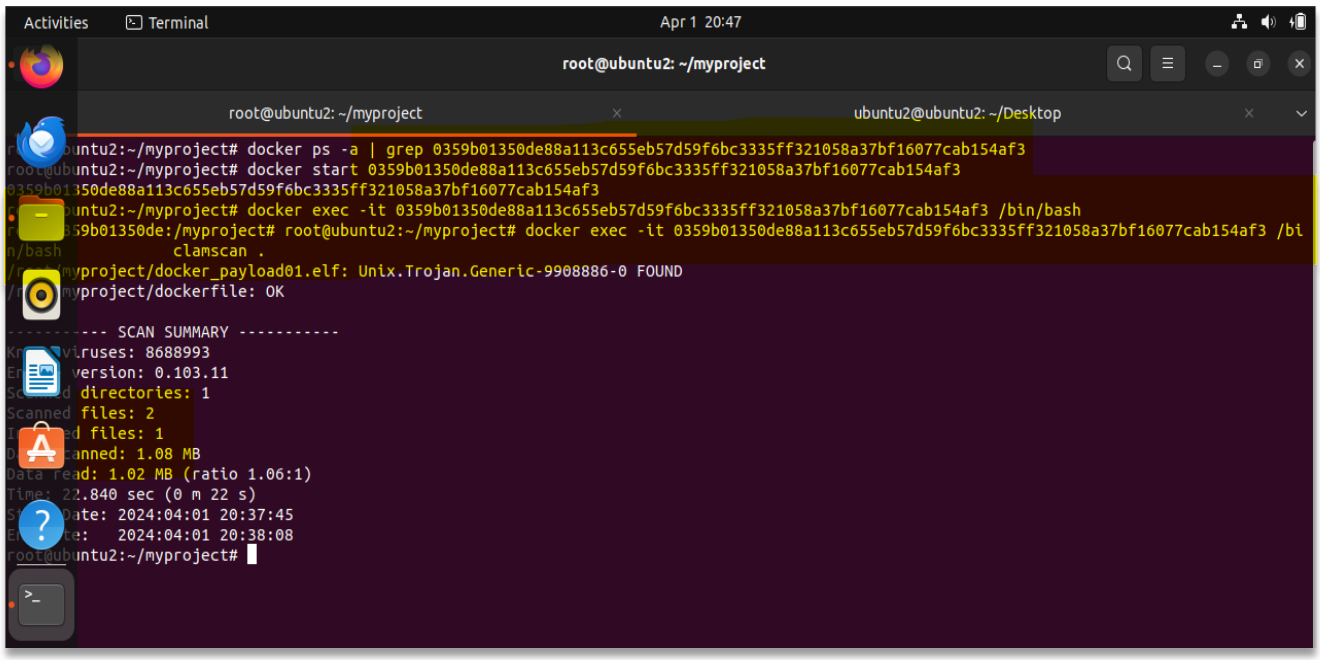

Figure 31. Scanning the docker container with clamscan

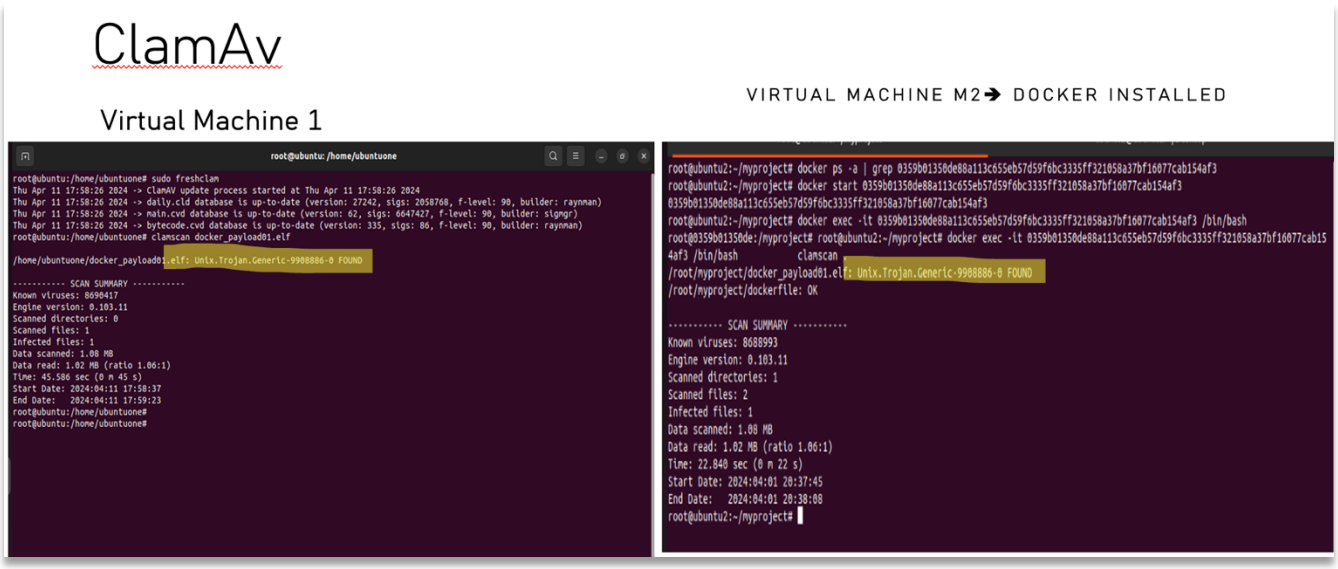

Figure 32 Final Result

Result: ClamScan

ClamAV was able to scan malware on both docker container and ubuntu OS.

Installing the CHKRootKit on both the virtual machine

- Virtual machine 1: First installing the chkrootkit antivirus.

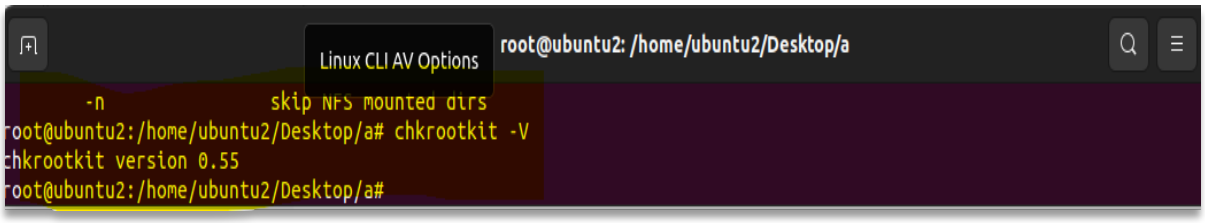

Figure 33 Chkrootkit Version Check

- Creating the dockerfile and copy the payload file on the same directory.

Figure 34 CHK file setup

- Commands for installing the antivirus and load the payload file inside the dockerfile

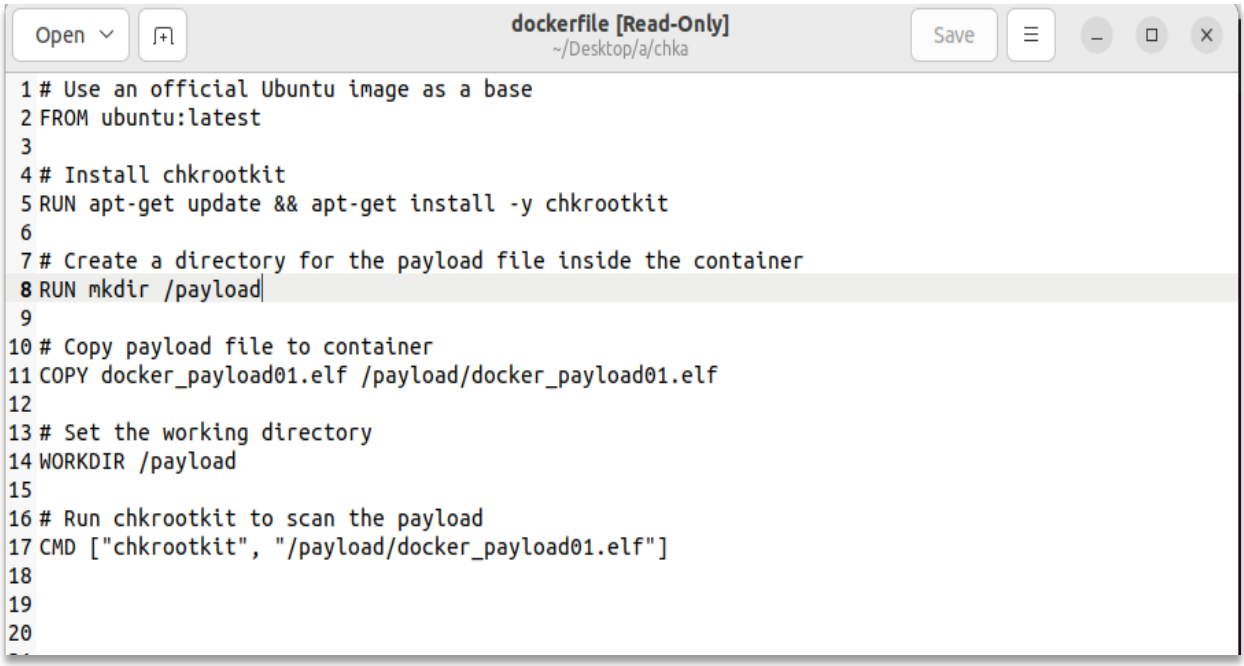

Figure 35 chk dockerfile

- Building the images inside the docker.

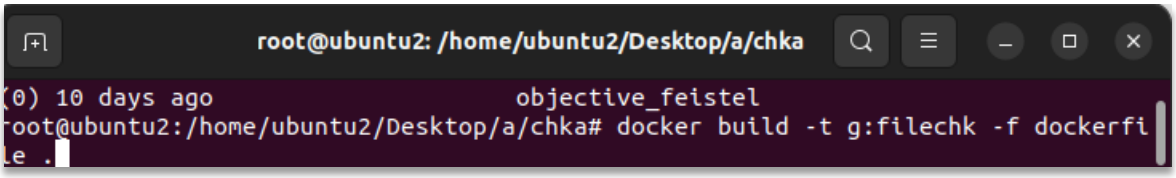

Figure 36 Image building

- Scanning the file inside docker

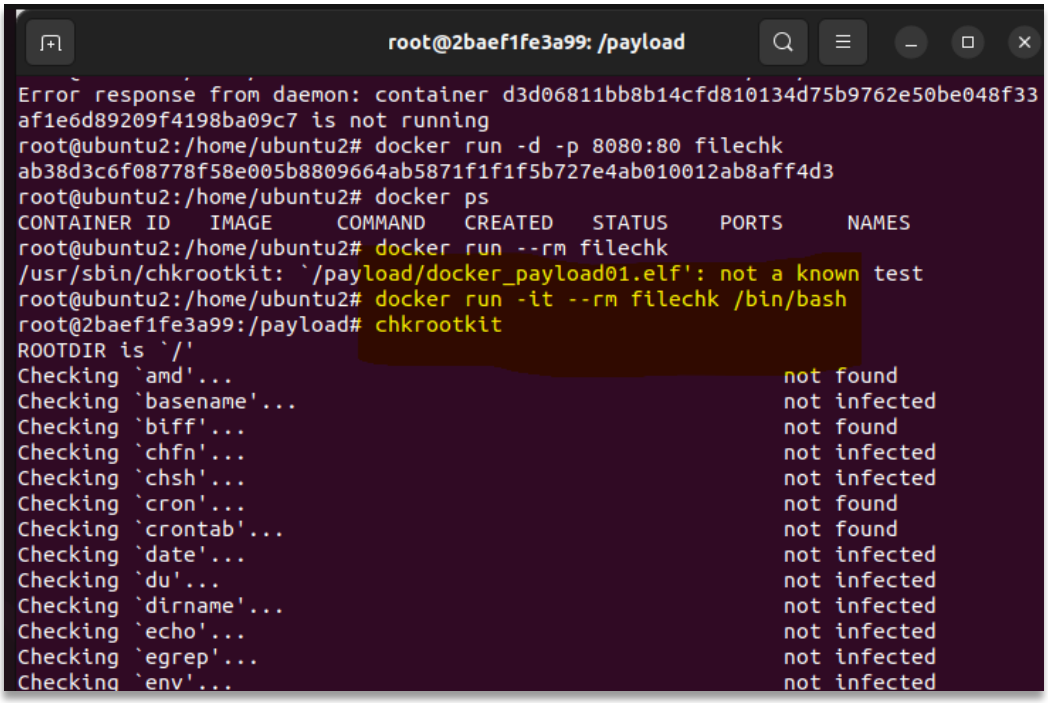

Figure 37 Scanning inside docker

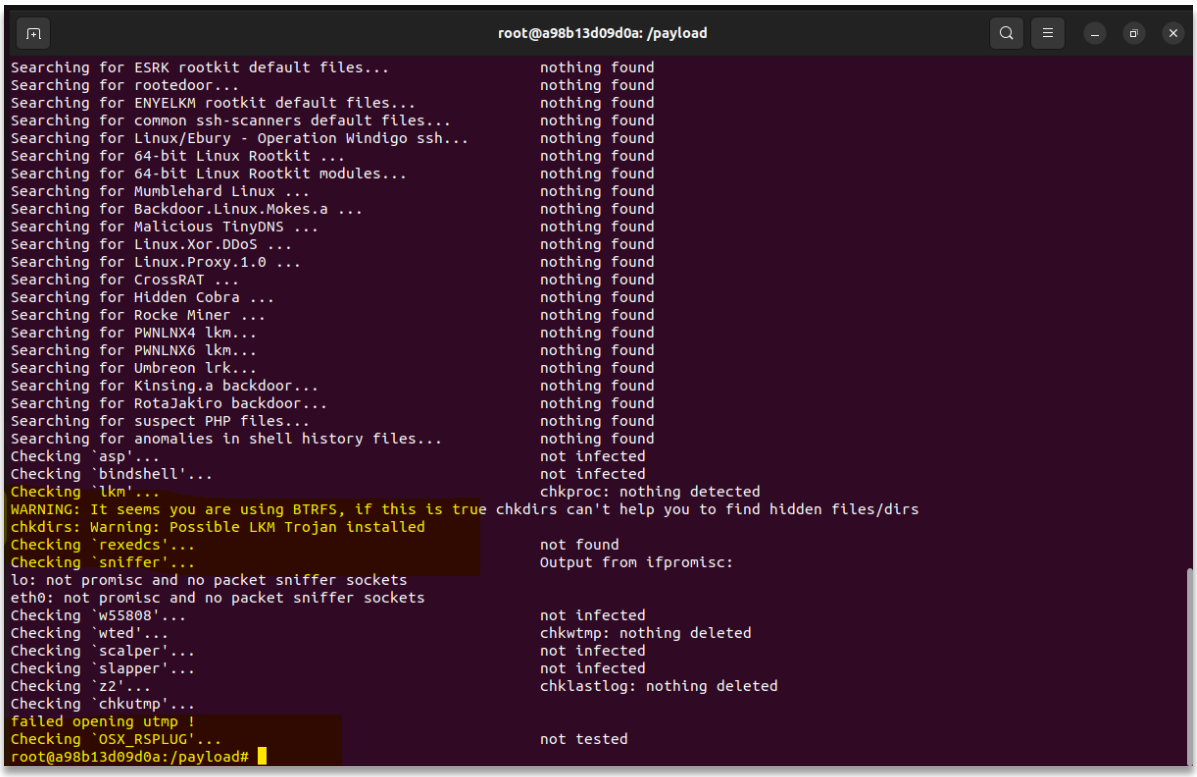

Figure 38 Scanning inside docker result.

- Virtual Machine 2: Installing the chkrootkit antivirus.

```
root@ubuntu:/home/ubuntuone# chkrootkit -V
chkrootkit version 0.55
root@ubuntu:/home/ubuntuone#
```

Figure 39 CHKrootikt version.

- Scanning the file using chkrootkit

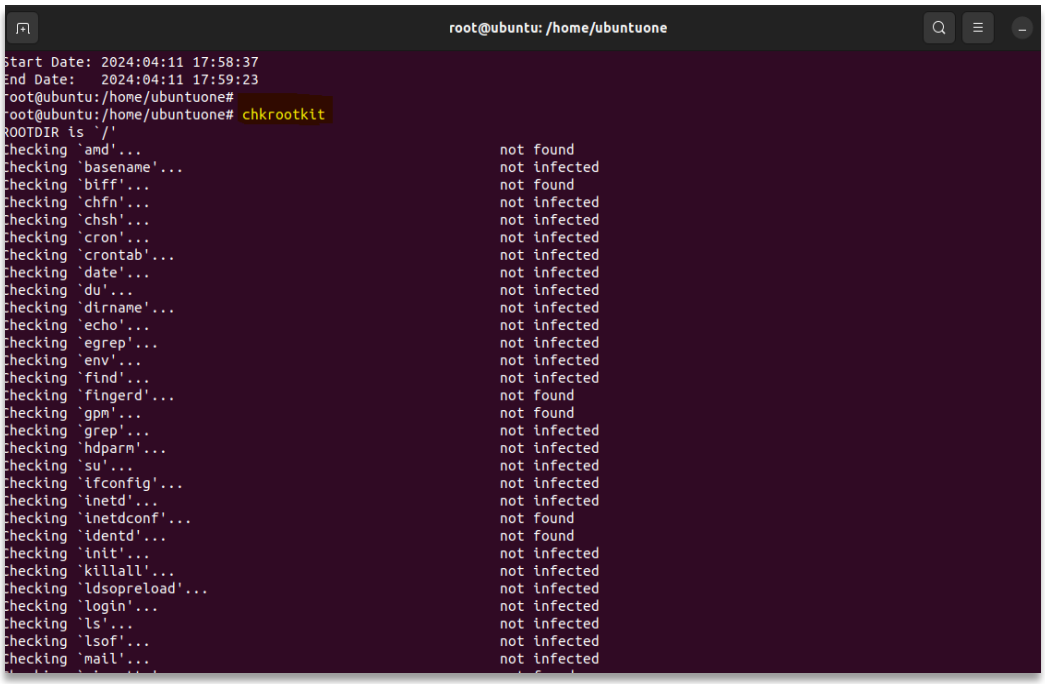

Figure 40 Scanning Chkrootkit

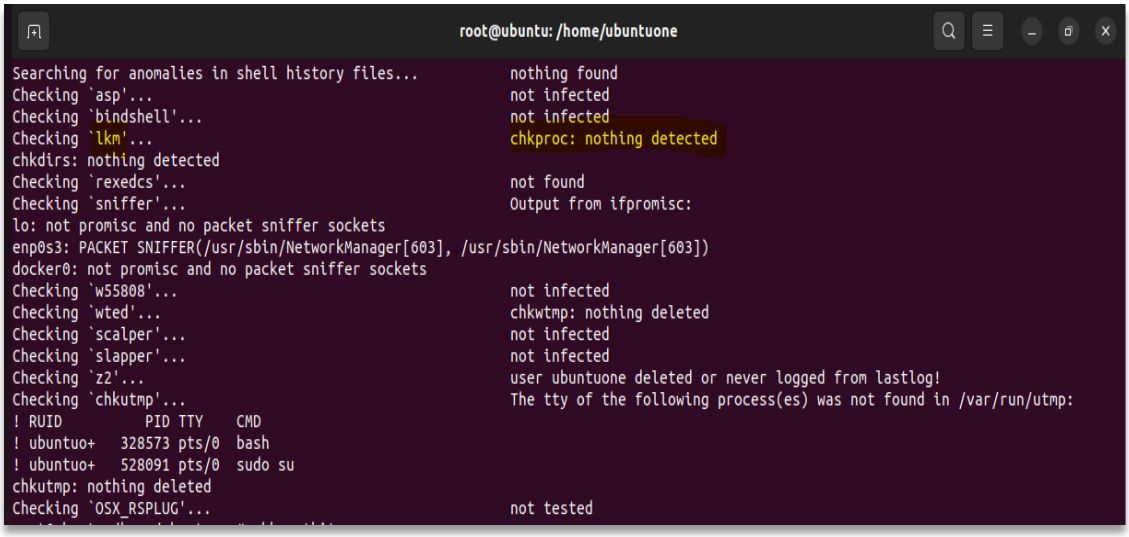

Figure 41 Scanning Chkrootkit Result

Results:

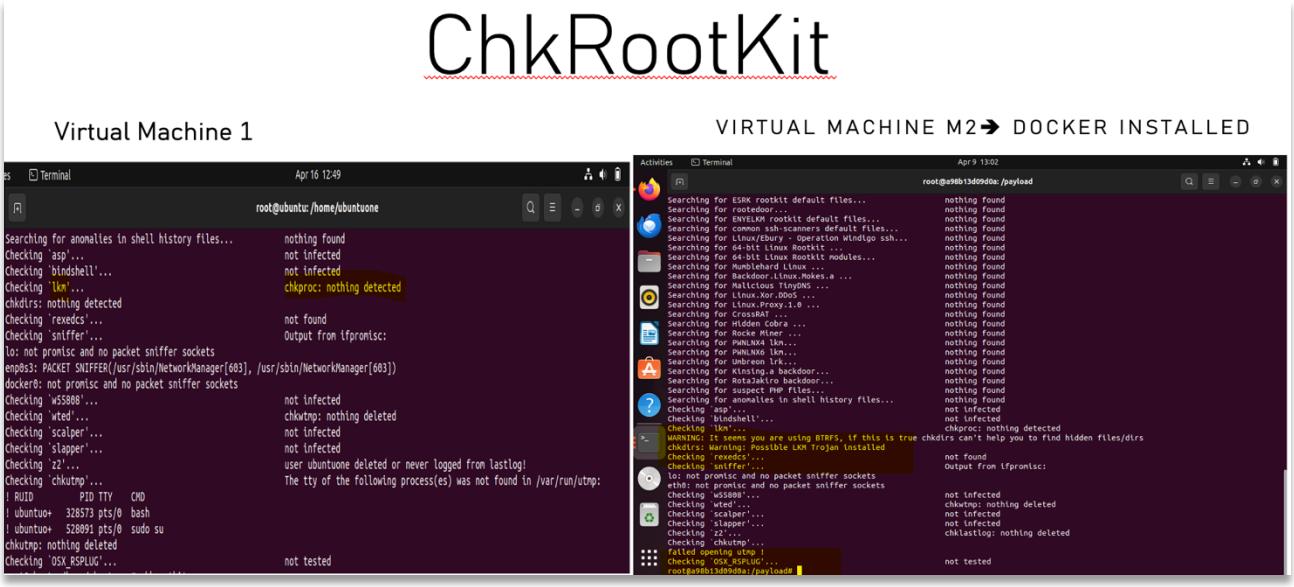

Figure 42 CHKrootkit result

Although ChkRootKit was able to detect malicious file in docker, it failed to report it in virtual box without docker or normal environment.

Installing the RootKitHunter on both the virtual machine

- Virtual machine1: Creating the dockerfile file and installing the antivirus.

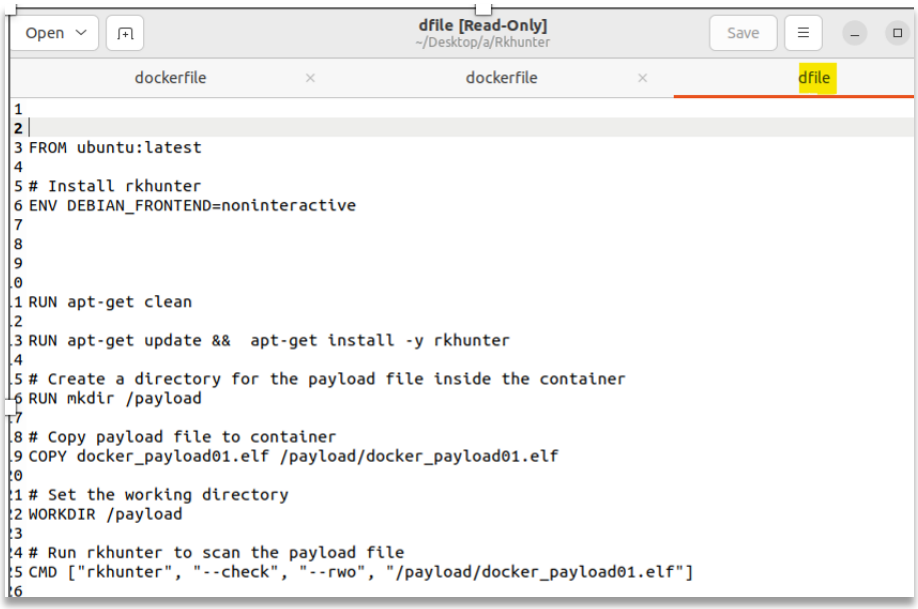

Figure 43 RKHunter dockerfile

- Installing Rootkithunter on both Virtual Machines

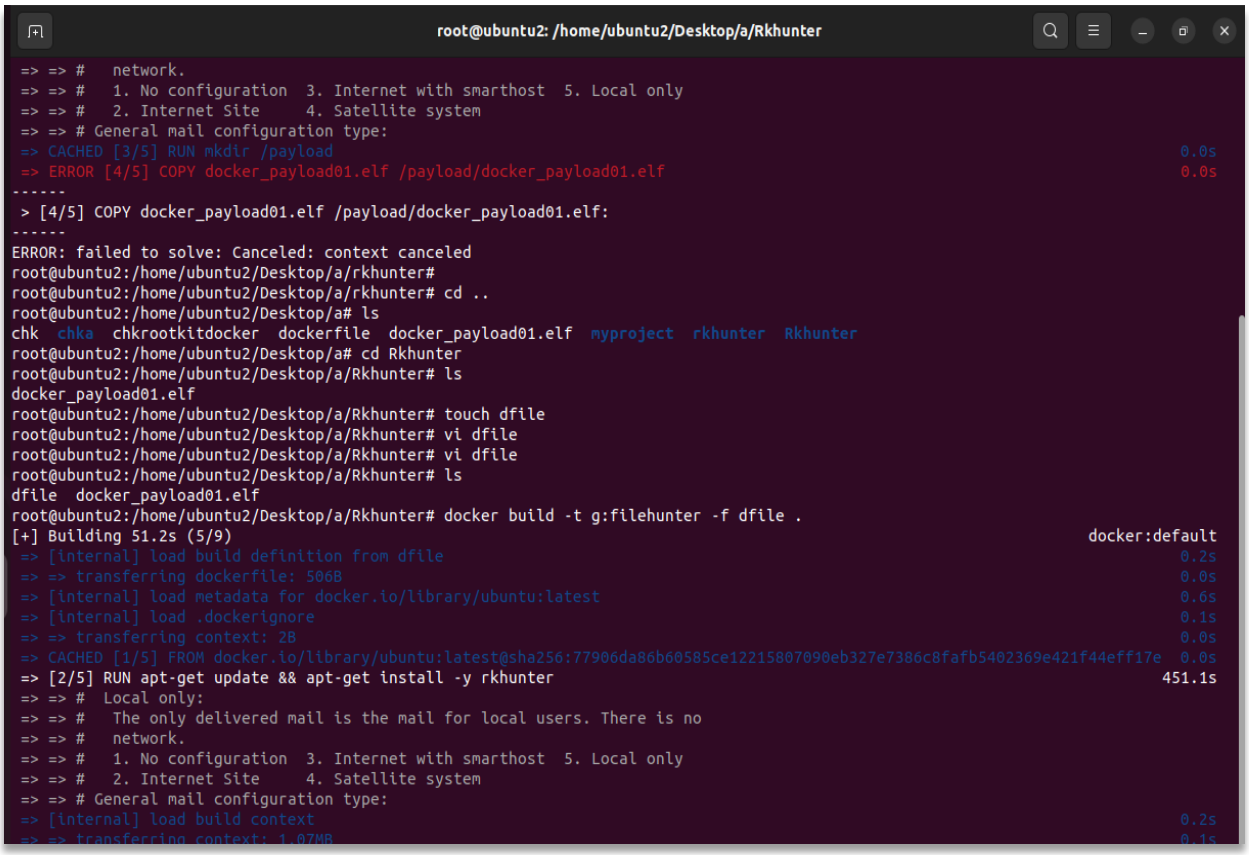

Figure 44 Rkhunter Installation

- Scanning inside the docker container

```
root@3017e449994a:/payload# rkhunter --check
```

Figure 45. RKhunter Scan

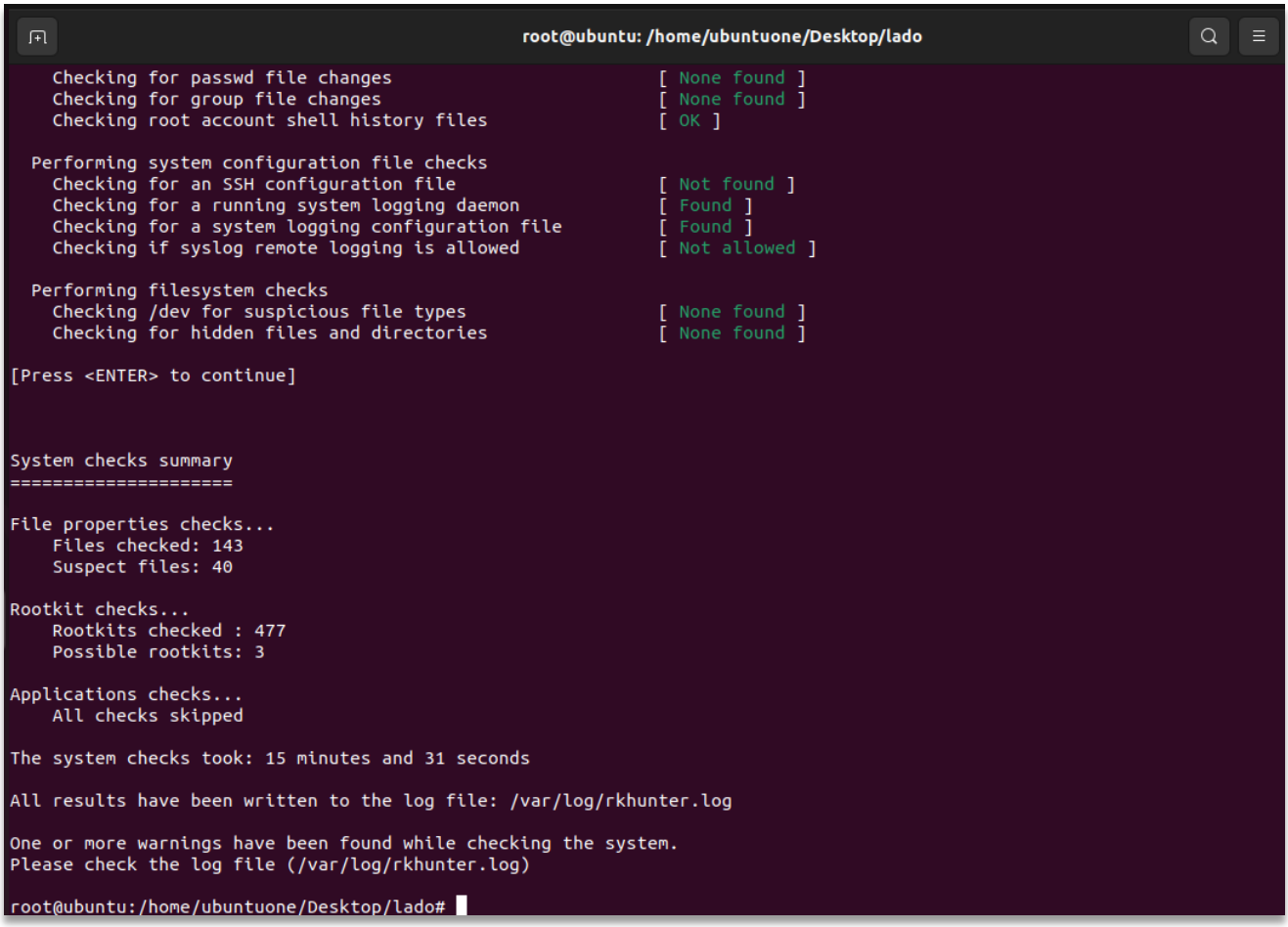

Figure 46 RKhunter Scan

- Virtual Machine2: Installing Rootkithunter

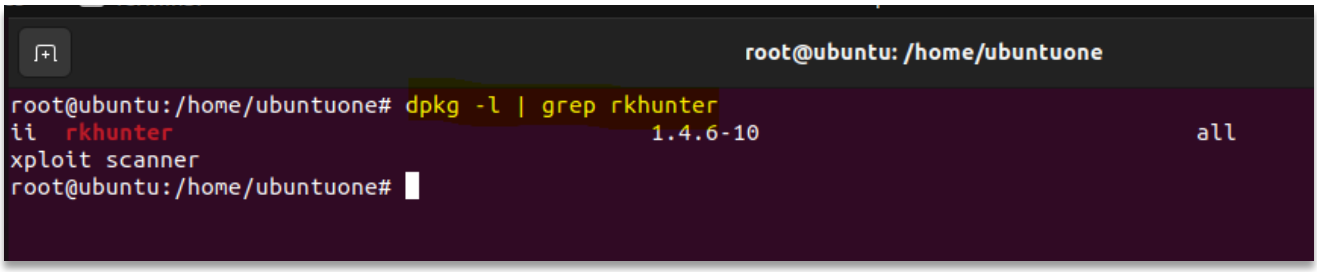

Figure 47 RKhunter Scan

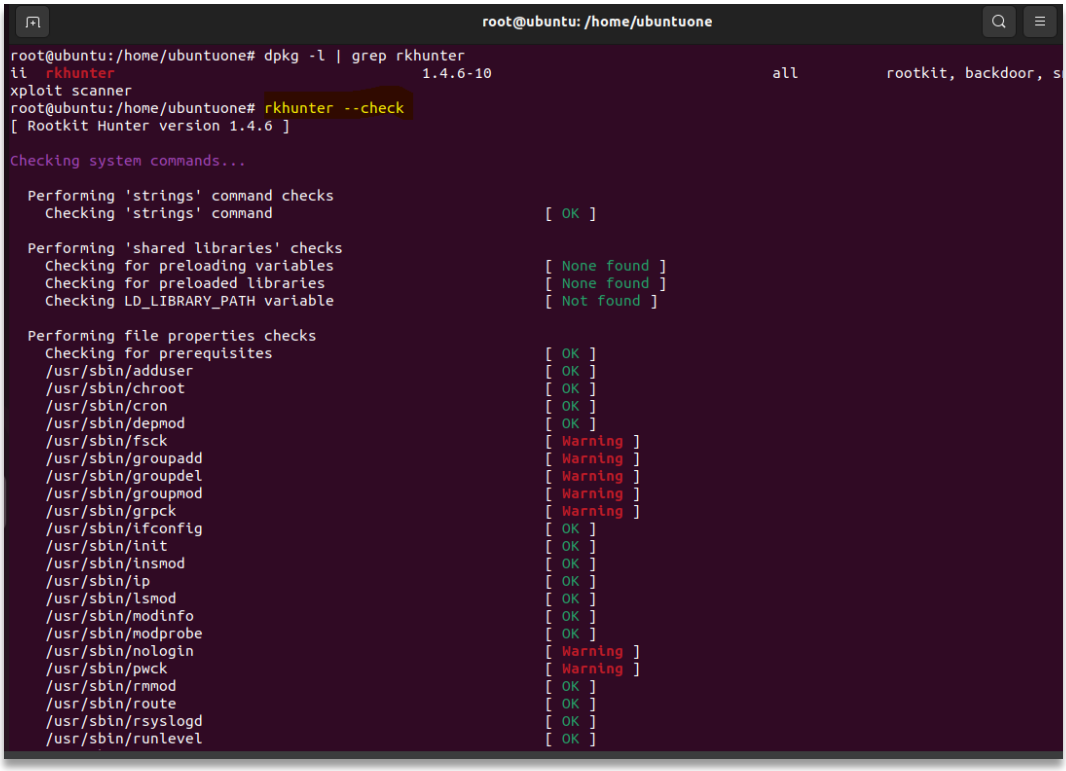

Figure 48. RKhunter Scan Docker

Figure 49 RKhunter Scan Docker

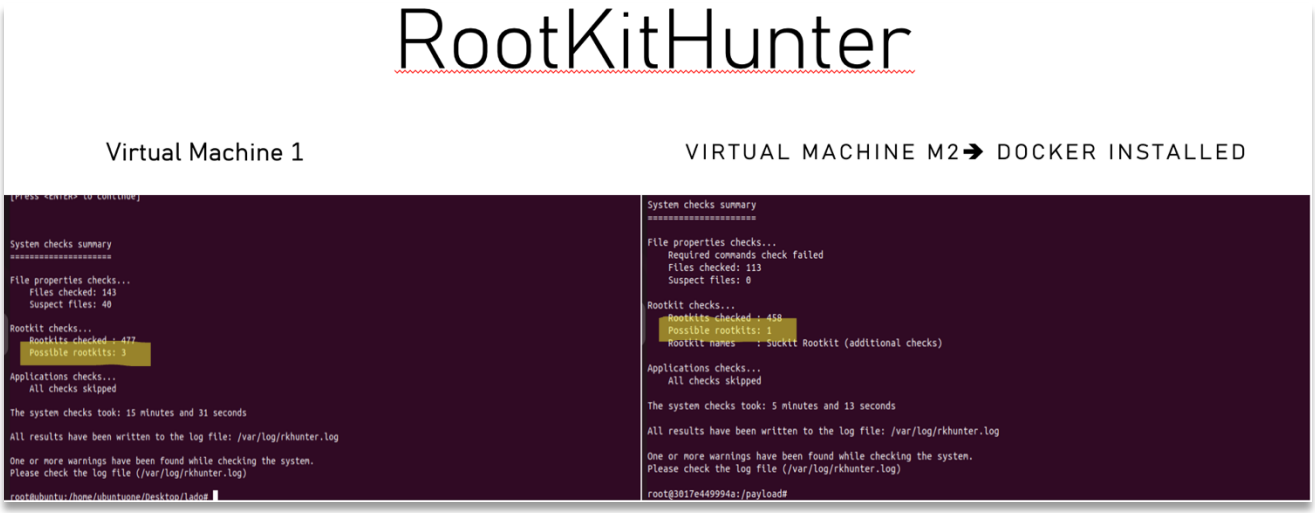

Figure 50 RKhunter Scan Docker result

Result: Rootkit hunter was able to warn us about the infected file

Isolation Tree Model

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler

dataset = pd.read_csv("data.csv", delimiter='|')

dataset[['popularity', 'star_count', 'pull_count']] = dataset[['popularity', 'star_count', 'pull_count']]
    .apply(pd.to_numeric, errors='coerce')
dataset[['popularity', 'star_count', 'pull_count']] = dataset[['popularity', 'star_count', 'pull_count']]
    .fillna(dataset[['popularity', 'star_count', 'pull_count']].median())

dataset['categories'] = dataset['categories'].fillna('')
dataset['operating_systems'] = dataset['operating_systems'].fillna('')
```

Figure 51 Isolation Tree Code 1

- To process and analyze data, this Python code makes use of libraries like scikit-learn, Matplotlib, and Pandas. Initially, a CSV file called "data.csv" is read and then saved in a Pandas Data Frame called "dataset`." The code then changes some columns ('popularity,"star_count,"pull_count') to numeric types and uses the median of the corresponding column to fill in any missing values. In the 'categories' and 'operating_systems' columns, it also inserts empty strings in place of missing values. For

tasks involving machine learning and visualization in data analysis, this preprocessing is necessary.

```
scaler = StandardScaler()
features_scaled = scaler.fit_transform(dataset[['popularity', 'star_count', 'pull_count']])

model = IsolationForest(contamination=0.05)
model.fit(features_scaled)
```

```
IsolationForest(contamination=0.05)
```

Figure 52 Isolation Tree Code 2

- This part of the code focuses on using scikit-learn to preprocess data to detect anomalies. To normalize the numerical features ('popularity,"star_count,'pull_count,') a StandardScaler object is first instantiated. To make sure that every feature contributes equally to the anomaly detection process, this normalization is essential. The Isolation Forest algorithm is then used to find anomalies. The expected percentage of outliers in the dataset is indicated by the contamination parameter, which is set to 0.05 when creating an Isolation Forest model. After that, the model is trained using the scaled features, allowing it to distinguish anomalies from deviations from the data's normalized distribution. The overall goal of this procedure is to efficiently identify and separate anomalous patterns or outliers from the dataset.

```
anomaly_scores = model.decision_function(features_scaled)
anomalies = dataset[model.predict(features_scaled) == -1]
```

```
plt.scatter(dataset['popularity'],
            dataset['star_count'], s=100*(anomaly_scores - min(anomaly_scores)) / (max(anomaly_scores) - min(anomaly_scores)),
                c=anomaly_scores, cmap='coolwarm', label='Normal', alpha=0.5)
plt.scatter(anomalies['popularity'],
            anomalies['star_count'], s=200*(anomaly_scores[model.predict(features_scaled) == -1] - min(anomaly_scores))
                / (max(anomaly_scores) - min(anomaly_scores)), c='red', label='Anomaly', alpha=0.8)
plt.colorbar(label='Anomaly Score')
plt.xlabel('Popularity')
plt.ylabel('Star Count')
plt.title('Anomaly Detection')
plt.legend()
```

Figure 53 Isolation Tree Code 3

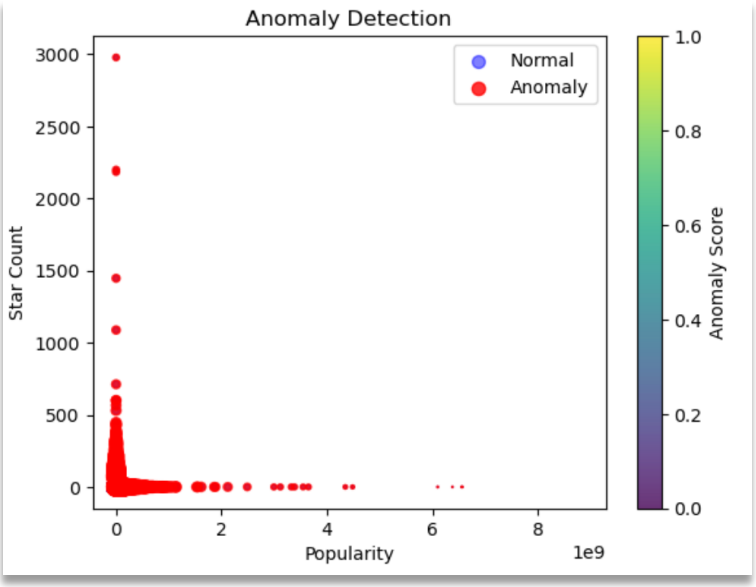

Figure 54 Isolation Tree Code 4

- The purpose of this code segment is to display the abnormalities that the Isolation Forest model has found. Using the model's learned patterns, the `decision_function} method is first used to calculate anomaly scores for each data point in the dataset. Then, by selecting the data points that the model predicts to be anomalies, the dataset is filtered to identify anomalies. Next, a scatter plot is used to create the visualization, with the 'popularity' of each data point being plotted against its'star_count'. The 'coolwarm' colormap exhibits a range of shades for normal data points, while anomalies are highlighted in red. The size and colour of the markers are determined by the anomaly.

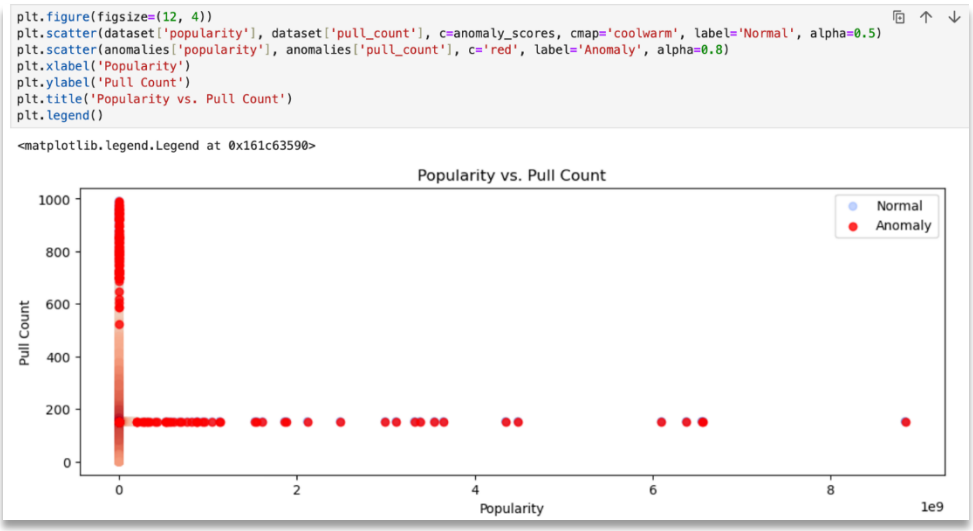

Figure 55 Isolation Tree Code 5

- This section of the code highlights anomalies found by the Isolation Forest model by creating a visual comparison between the 'popularity' and 'pull_count' features. To ensure a good aspect ratio for the plot, the figure size is first set to 12 inches in width and 4 inches in height. Next, two scatter plots are produced: the first plots "popularity" against "pull_count" and uses the "coolwarm" colormap to indicate anomaly scores. This scatter plot displays normal data points. Anomalies are overlaid in the second scatter plot, which clearly distinguishes them by highlighting them in red. The titles 'Popularity vs. Pull Count' concisely communicates the plot's content, and axes labels are assigned for 'popularity' and 'pull_count' to aid in interpretation.

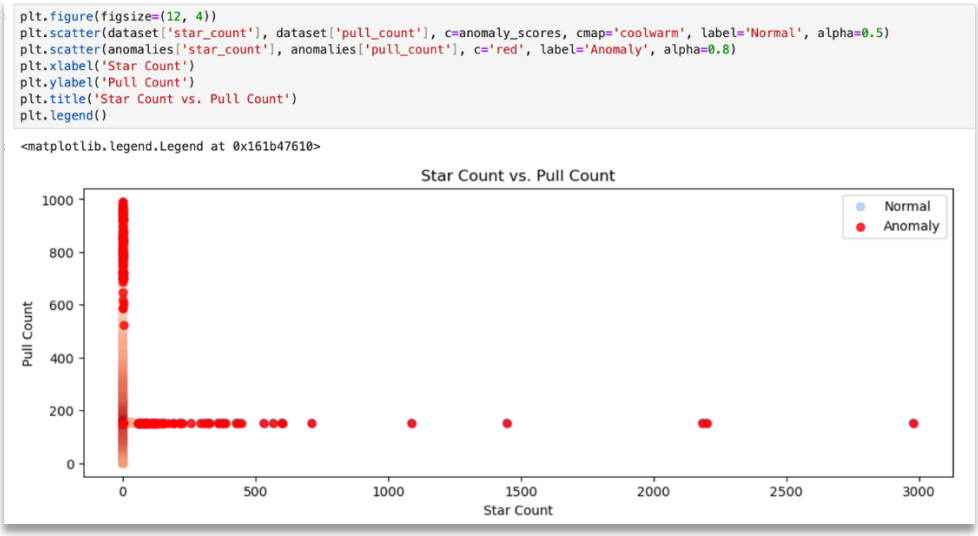

Figure 56 Isolation Tree Code 6

- To investigate the correlation between the'star_count' and 'pull_count' features, this section of the code creates a graphic representation that highlights anomalies found by the Isolation Forest model. The figure size is optimized for visualization with a width of 12 inches and a height of 4 inches. A pair of scatter plots are generated: the first one shows the normal data points by charting 'star_count' against 'pull_count' and utilizing the 'coolwarm' colormap to represent anomaly scores with marker colors. Anomalies are overlaid in the second plot and are indicated by red markers. The title 'Star Count vs. Pull Count' provides a concise summary of the plot's content, and axes labels for 'star_count' and 'pull_count' aid in interpretation. To further improve clarity and interpretation, a legend is added to distinguish between normal data points and anomalies.

Recommendations and Further Analysis

Extensive Examination of Antivirus Software Performance: Conduct an exhaustive examination of antivirus software performance on a variety of Linux distributions and containerization platforms. A wide range of metrics, such as resource usage, false positive rates, and detection rates, should be included in this investigation. Detailed information about the effectiveness of antivirus programs in containerized environments would be provided by such an analysis, which would help stakeholders make well-informed decisions about cybersecurity tactics.

Advanced Machine Learning Techniques for Anomaly Detection: Go beyond the Isolation Forest approach and investigate more sophisticated machine learning algorithms to delve deeper into the field of anomaly detection in Docker environments. To improve detection accuracy and resilience against advanced malware strains, think about utilizing sophisticated techniques like ensemble learning approaches or deep learning architectures. This research avenue holds promise in pushing the boundaries of anomaly detection capabilities within containerized infrastructures.

Container Security and Regulatory Compliance Framework Alignment: Examine how container security practices fit into frameworks for regulatory compliance such as PCI DSS, GDPR, and HIPAA. Examine how regulatory requirements affect containerized environments and evaluate how well security controls fulfil compliance requirements. Researchers can help organizations achieve a harmonious balance between security posture and regulatory adherence by clarifying the relationship between container security and compliance.

Security Automation and DevSecOps Integration: Examine how security automation procedures can be incorporated into the pipelines used by DevOps and DevSecOps for containerized applications. Examine how security-as-code concepts, automated vulnerability scanning, and continuous security testing can improve the security resilience of containerized deployments. This line of research encourages the development of a proactive security culture in container-embracing organizations by bringing security into the software development lifecycle.

Conclusion

The research project titled "Antivirus detection of containerized malware in Linux distributions" demonstrates how important antivirus software is in protecting containerized environments. The study provides important insights into detection capabilities through a methodical assessment of well-known antivirus software and the creative application of machine learning techniques such as the Isolation Forest algorithm. Moreover, the incorporation of pragmatic components, such as employing VirtualBox instances to replicate actual conditions, augments the dependability and relevance of the results. The research emphasizes the significance of developing detection techniques and closely integrating with container orchestration platforms, despite the difficulties presented by the dynamic nature of containers and the changing threat landscape. The Isolation Forest algorithm effectively detects anomalies within datasets, providing valuable insights for improving security measures. Furthermore, the generated visualizations provide a clear representation of anomalous patterns, facilitating further analysis and decision-making processes. Finally, exporting detected anomalies to a CSV file simplifies subsequent examination and action. To preserve security and integrity in Linux distributions as containerization continues to reshape digital ecosystems, strong antivirus detection systems are essential.

References

[1] datascientest, "datascientest," 23 02 2023. [Online]. Available: https://datascientest.com/en/linux-the-preferred-os-for-developers.

[2] J. Wallen, "Zdner," 19 02 2024. [Online]. Available: https://www.zdnet.com/article/do-you-need-antivirus-on-linux/.

[3] E. Stevenson, "antivirus Detection of containerized malware in Linux distributions," SANS, 2023.

[4] scikit, "scikit-learn," 10 04 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html.

[5] A. Mavuduru, "Medium," 21 11 2021. [Online]. Available: https://towardsdatascience.com/how-to-perform-anomaly-detection-with-the-isolation-forest-algorithm-e8c8372520bc.

[6] K. Pareek, "Geeks For Geeks," Geeks for Geeks, 25 April 2023. [Online]. Available: https://www.geeksforgeeks.org/architecture-of-docker/. [Accessed 01 04 2024].