International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org Original Research Paper

A Hybrid Model for Real-Time Docker Container Threat Detection and Vulnerability Analysis

Vipin Jain[1], Baldev Singh[2], Nilam Choudhary[3], Praveen Kumar Yadav[4]

Submitted:13/02/2023 **Revised:**16/04/2023 **Accepted:**10/05/2023

Abstract: Computer attacks are becoming more sophisticated and difficult to detect, the large volume of information in security records generated by network devices, servers and applications does not allow an adequate analysis due to the complexity of the data produced, and this leads to not taking measures timely in the event of gaps or security incidents. In spite of the fact that containers are widely accepted as a standardized way of deploying micro services and play a vital role in emerging areas of cloud computing such as service meshes. A survey has revealed that container security is the common concern and barrier to adoption for most organizations. This paper aims to conduct a review of the existing literature on container security and solutions. The time that elapses between the occurrences of an incident until its discovery should be as short as possible in order to determine the scope of the incident and compromised systems in a timely and efficient manner. The main objective of this paper is to deploy a solution using open source tools in a Docker container based environment to allow analysis of security anomalies. Therefore, relevant information must be collected, processed, and presented through dashboard displays in order to identify or detect security incidents in real time in a timely manner.

Keywords*: Docker; Container; Threat; Cloud; Vulnerability*

1 Introduction

According to data provided by the 2019 M-Trends report of the Fire Eye company on breaches and cyber-attacks [1] , the global average time from having evidence of an intrusion to detection is 78 days, which means that a cybercriminal has more than two months to operate freely on a vulnerable system without being detected. Although this time is reduced each year, said report indicates that 31% of incidents are detected within 30 days or less. This time is still high considering the damage capacity that a computer attack can cause to an organization and the economic losses that it can cause. On the other hand, according to the report prepared by the EU Agency for Network and Information Security (ENISA)[2]. In which an analysis of cyber-attacks is carried out, the biggest cyber threats for the years 2017 and 2018 continue to be Malware, attacks on web services, phishing and spam. A new threat Cryptojacking is identified in year 2021.

[1]*Research Scholar,Department of CSE, Vivekananda Global University, Jaipur-303012 (INDIA)*
Associate Professor,Department of IT, Swami Keshvanand Institute of Technology, Management & Gramothan,Jaipur-302017 (INDIA)
[2]*Professor, Department of CSE, Vivekananda Global University, Jaipur-303012 (INDIA)*
[3]*Associate Professor, Department of CSE, Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur-302017 (INDIA)*
[4]*Assistant Professor, Department of IT, Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur-302017 (INDIA)*
Author's Email: ervipin.skit@gmail.com[1], baldev.vit@gmail.com[2], neelamvit@gmail.com[3],praveenyadavskit@gmail.com[4]

This attack is based on the malicious use of computers for cryptocurrency mining. Within the field of cybersecurity, it is also important to know the main industries affected by a security incident. According to IBM's annual report on the threat index [4], the main organizations attacked in 2021. Due to the various threats that organizations face, it is essential to monitor and analyze the records or events generated by the devices, systems or applications, as well as the events produced by the operating systems themselves related to the records of Audit that allows you to recognize logins, file modifications and erroneous configurations. These mechanisms facilitate the identification of malicious activities or violations in the security policy and allow taking preventive and corrective measures [4], this in order to avoid a security incident that affects IT assets. On the other hand, many of the organizations have several simultaneous methods that allow anomaly detection: intrusion detection and prevention systems, antivirus, authentication servers, firewalls, etc.

1.1 Current Situation

Many of the organizations carry out the compilation of events or logs as essential elements to know the behavior and state of the network. The existence of a great variety and quantity of devices means that it is necessary to deploy a centralized log management system, which allows the administrator to determine in a timely manner any type of incident of some kind in the element of communications infrastructure [5].

1.2 Virtual machine V/s Container

When it comes to computer security, virtual machines (VMs) [6] top the list hands down. But even though virtual machines (VMs) provide better security isolation, the number of virtual machines running on a server is limited by the need for each VM to have its own copy of the OS, libraries, dedicated resources, and applications [6]. As a result, both system performance (such as slow startup times) and available storage space are negatively affected. There is a pressing need for a faster alternative than virtual machines (VMs) since operating each microservice on a separate VM is wasteful because of the long startup time and increased resource usage that happens. Traditional virtualization has been replaced by container-based virtualization. Unlike virtual machines, which each have their own operating system kernel, several containers can share a single kernel. This greatly reduces the startup time and the amount of resources required for each image. While a virtual machine (VM) may boot in as little as 30-40 milliseconds, a container can do so in as little as 50 milliseconds. To ensure this article was ready to be published, Kashif Saleem helped as an assistant editor.

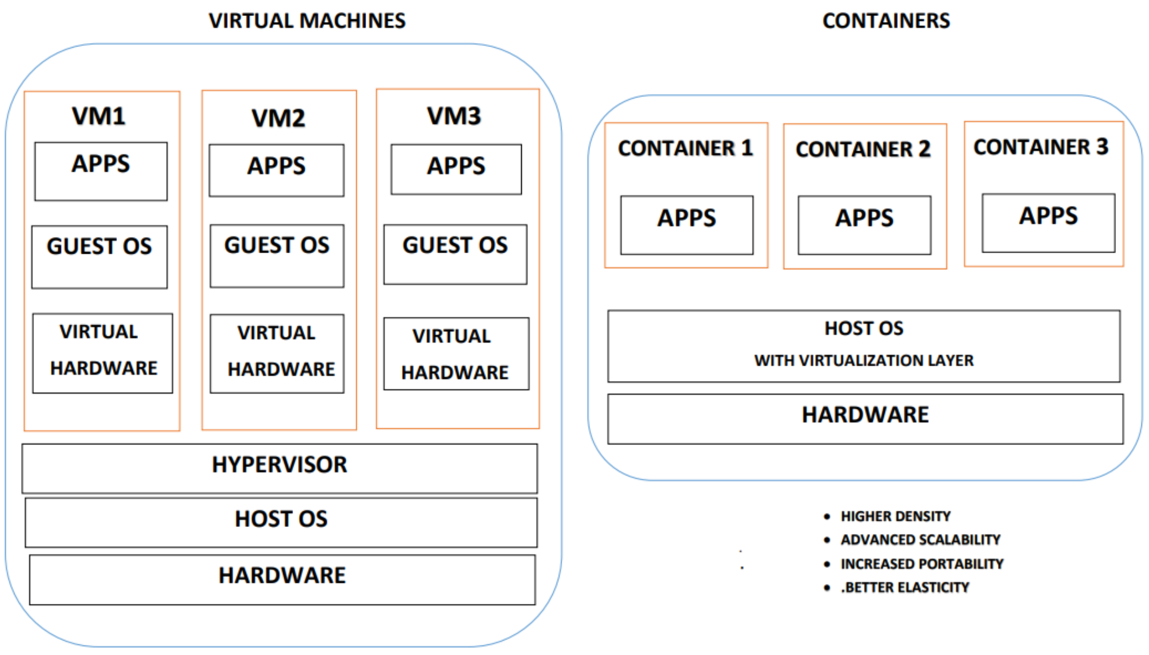

Fig 1: Comparison between Virtual machine and Docker

[7] begins the process. LXC, OpenVZ, and Linux-Vserver are all container technologies, but Docker is the most popular. It shows the fundamental container architecture, whereas the design of a simple virtual machine. For micro services, containers provide a number of advantages over virtual machines (VMs), such as being lightweight, rapid, and simple to deploy; they also allow for improved resource efficiency and version control, making them a more feasible option. Internet of Things (IoT), smart cars, fog computing, service meshes, and other related initiatives are all benefiting from containers. Because Linux containers are virtualized at the system call level, programmed running within them share the same underlying Linux kernel as those running outside of containers. Container-based cloud services provide a number of advantages over virtual machines as a result, including the following [8]:

1. When running in a virtual machine, it is not reasonable to expect a virtual machine application to perform as well as a virtual machine program running in a container when running in a virtual machine. In part, this is due to the fact that containers don't emulate hardware but instead directly access system resources.
2. Containers may be deployed at a higher density than virtual machines since they have a shorter startup time when compared to virtual machines.
3. Because containers only run a few apps at a time, they consume resources such as memory more effectively than virtual machines, allowing them to be deployed at a higher density than virtual machines.

1.3 Container Related Technology

Linux containers use virtualization at the system call layer to provide isolation. Virtualization is implemented at the system call layer to achieve isolation. In addition to Docker and LXC, there are some of the Linux container technologies that are currently available. The most significant difference between container technologies is how they make use of the namespace [9] support that is supplied by the container technology. Namespaces are utilised in a number of ways by different technologies in a range of contexts. On the management stack level, separate command line tools and APIs are offered, as well as a diverse ecosystem with a variety of functionality that is supported by the management stack. The performance of all containers, on the other hand, is expected to be identical. Furthermore, para virtualization and full virtualization are two of the most widely used virtualization technologies today. It is necessary to implement hypervisor calls in order to transition between the operating system kernel and the hypervisor's virtualization layer, which is referred to as paravirtualization in this context. As an example, consider the hypervisor in the Xen operating system, as well as the Linux Xen support for paravirtualized kernels [10]. Paravirtualization, as compared to full virtualization, provides a considerable performance advantage over the latter technology. The ability to boot paravirtualized computers without the usage of firmware is available in a variety of configurations and environments.

2 Literature Review

Container use initiatives have been built in numerous fields since Docker's inception in 2013. Docker has the advantage [11], but does not have protection assurances for any established security bugs in Docker image structures, that it can easily exchange application building environments with developers by container technologies. As Docker images are shared without a security diagnosis, polluted Docker images can be distributed to make it easy to collapse the Docker based application environment. In this article, we are addressing a secure Docker system with a Docker Image Vulnerability Diagnostic System (DIVDS). Once Docker files are posted or downloaded from a Docker file store, the DIVDS suggested diagnoses Docker images.

With the exponential growth of test results, the fusion computing pattern is focused on the centralized database or file system [12]. A cluster is essential for such storage systems, but its set-up and delivery of resources are difficult. It is a complex task to install, configure the operating system and software and manage all dependencies. Beyond this, different configuration protocols are important, owing to the variation of server equipment and the disparity between testing and the production environment.

The J-TEXT cloud database (JCDB) is an experimental nuclear fusion data storage and management system to satisfy the needs of future MongoDB, Cassandra cluster and web applications push-on long distance experiment [13]. The configuration or size of JCDB from scratch on these servers is not easy. Docker is a lightweight containerization software platform for the system, to construct cloud databases and internet apps, ensuring all servers are consistent and automating deployment and scale out tasks, reducing time required. JCDB sent out two files and the files were removed to the Docker server. You can also deploy it to cloud services like Amazon Web Services and Microsoft Azure even without any physical servers available.

The efficiency advantages of a lightweight container architecture can be reaped with rapid I / O response using quick back-end storage [14]. It can, however, have a considerable impact on the performance of several instances of the same or distinct applications running at the same time. Due to the variety of I/O (read and write), I/O (random and sequential), and I/O access patterns that SSDs may support, performance can vary widely even within the same application. As a first step, this study aims to examine and analyse the performance characteristics of containerized IO-intensive applications with high performance NVMe SSDs, as well as new guidelines for the design of both homogenous and heterogeneous mixtures in order to achieve an optimal and fair operation. We're working on a new docker controller to programme workload containers for diverse applications using these development principles.

In order to minimize total execution time and maximize resource use, our controller decides the optimum batches of simultaneously working containers [15]. In addition, our controller tries to ensure that all apps running simultaneously are performing at a consistent level. This new docker controller was developed by addressing an optimization challenge using five distinct optimization strategies. We used a multi-docker configuration with three NVMe discs for our tests. Using a variety of different I/O applications and I/O behaviours, we test our controller and compare it to containers without a controller in parallel. The results of our tests show how well our new Docker workload controller handles a large number of apps running simultaneously on SSDs.

Increased services with unknown demand pattern in the edge network, driven by the growing popularity of the microservice architecture [16]. It's unlikely that pre-deployed services, which are impractical commercially, would be mostly silent. In addition, our restricted storage capacity restricts the amount of instances that we can provide. Instead, we have an on-demand depletion strategy thanks to the Docker platform. In Docker, each layer of a unit image has a separate piece of functionality. To prevent humiliating the storage, different services are allowed to repeat layers. Users may link to a server and download all layers from various places, and then run the required application instance as quickly as possible using our layer placement method. Because of storage and latency limits, we seek for the ideal layer placement to optimise fulfilled demand. Through the subdivision of the global issue into smaller subproblems, a less exhaustive iterative optimization was also constructed. Our simulation results suggest that our heuristic solves the problem of less machine capital. Finally, we give a few examples of how this method might be put to use in the real world.

Virtualisation Network Feature (NFV) is one of the main facilitators in mobile network networks of next-generation (5G)[17].There are a broad range of virtualization systems (e.g. virtual servers or containers) that may be used to construct virtual networks (NF) in a cloud environment. With its great efficiency and adaptability, it is an excellent choice. As a result, the 5G core network's NFs may be loaded as software on the platform's common hardware. JSON and HTTP/2.0 are used to create the Network Depository (NRF) function in the Docker-based platform for NFV service discovery. This includes the usage of hardware platforms for performance testing such as NFReg, NFRupdate and NFRegDeregister The testing data shows that the NRF performs well in the 5G NFV architecture and that the NFV platform is more adaptable than traditional communication networks from Docker..

Complex CNC system functions and a wide range of product types are required to meet the needs of the present industrial age and to create a modular CNC framework that can be customised and expanded. In this paper[18], With the high-performance multi-core processor, we propose an integrated parallel CNC system that can operate multiple CNC machine tools based on the original architecture between CNC system and machine tool and to increase integration level of Industrial Control System. For example, Docker containers have their own interpreters and interpolators because of container virtualization technologies.

The Program Scheduling Software schedules over usable modules and caches the data that each software produces in order to optimize the quality of processing and the utilization of resources [19]. The colored Petri networks are used to model the system architecture for CNC models and task planning. To decide the goals of the system threads, two approaches for task preparation dependent upon the task dependencies and the flow of data are provided. The developed platform analyzes the real time environment of the virtual system, the real-time system environment and the integrated CNC communication module. In addition, the results of the tests show that synchronized control of two equipment tools is possible with the proposed integrated CNC parallel system.

Docker offers the chance to further enhance the efficiency of DCs. However, existing models [20] and schemes for the allocation of container-based resources by Docker are not efficient. We are designed to minimise infrastructure costs for DC installations and enable Automated Scaling as the operating load of cloud applications varies, using a new infrastructure-focused Docker resource management architecture (AODC). The AODC resource allocation issue will then be modelled taking into consideration Docker characteristics, various applications requirements and available resources in cloud data centres, and a scalable DC algorithm with a variety of dynamic applications and massive physical resources.

Desktop applications across edge networks need to transition services smoothly [21]. Edge computing systems have to help these transitions of resources seamlessly and keep up with network device movements. Live migration of offload services in the wide area network, even in the edge computing environment, is a major challenge. In this paper we suggest the architecture of the edge computing platform to facilitate smooth file transfer, while still retaining the device user in contact with the nearest edge server. In the state-of-the-art container migration method, we define a crucial issue. Based on our systemic analysis of the container storage system in Docker, we propose to leverage the storage system's layered nature so that overhead synchrography is reduced without reliance on the distributed file system. Our system cuts handoff speed by 80 per cent (56 percent) under 5 Mbps (20 Mbps) network bandwidth conditions in comparison to the state-of-the-art service handoff approach in the edge setting.

Docker is used in hosting facilities of data centres. A hierarchic storage architecture is implemented in the docker model [21], which means that this image consists of a file system consisting of layers. Only the top layer is read in writing in the process of creating images, while all lower layers are read-only. In the process of image creation, however, temporary files are also used. However, if an unattended developer inserts and extracts a temporary file in various layers then a file becomes redundant. This scent leads to big images that significantly restrict the reliability of image processing, reducing the quality of operation in the face of unexpected heavy loads. This has of- ten been dubbed "Temporary File Scent" issue. In order to resolve this problem, we conduct a case study on DockerHub for the real world Dockerfiles. We summarize four different smell patterns and propose a state-dependent static analysis method for the detection of such smells. Based on this case study, we also provide selective options to remove the temporary file with three possible fixing methods.

2.1 Docker Hub’s

When it comes to container images, Docker Hub has the largest database available. Docker Hub official repositories are divided into two types, official and community, depending on their contents. Meta-information such as repository descriptions, repository histories, Docker files, the number of stars and pulls for each repository, and developer information is also included alongside each image in a Docker Hub repository. Conducting a risk assessment is the first step in analysing Docker Hub's security. Using the primary factors that affect a Docker image's behaviour, we pinpoint potential dangers [22].

2.2 A command and a sensitive variable

Users can launch a Docker container with the command run-cmd. Running a container relies on the image and parameters specified in the run command. "docker run vipin-opencv2-privileged p 1234:11 vipin/opencv/x86" is an example of how to launch a container on Docker Hub. Suggested run-commands can be helpful to those who have never used the image previously. For customers, the instructions provided by anonymous programmers are not explicit enough to tell them how much trust they should put in them. Consequently, these files are not filtered out by Docker Hub in any way. Running a command with a variety of arguments can also affect how the container behaves [23]. When it comes to the isolation between a host computer and its docker container or other containers when it comes to network and storage, these elements are crucial. Running an image with a privileged parameter is one method of gaining root privileges. Neither the host nor the container will be spared the consequences of a badly handled run command that contains sensitive parameters.

2.3 Programs Execution in Container

Many Docker images already include duplicate software, according to previous research [24]. Docker images should be secure if we focus on the applications running on them. According to our empirical research of Docker images, an entry-file (an executable file in images, determined by a configuration or execute commands) is always the first software to be activated when a container is launched. It is possible that other files will be automatically started when the entry-file is run. Safety depends on the programmes that are carried out on a container (the entrance file and subsequent triggered files). As a result, consumers are less likely to use software that is distinct from what they have previously used. Why do we need to look at the programmes that have been executed in order to see whether any hazardous images have been found?

2.4 Containment Program Vulnerabilities

Docker images are built from many different types of programmes, each of which has unique vulnerabilities that might pose a serious threat to the system. Exploiting security holes might lead to data leaks and other problems. Thus, Docker has no incentive to fix bugs in software that has been reused from other systems [25]. Docker’s security concerns are increased since it takes longer to fix bugs in Docker images.

3 Proposed Work

To better understand the security of Docker Hub, we need to know what images are available on Docker Hub. Since much of the data has never been gathered, the analysis is incoherent. A custom web crawler that uses the API provided in [26] is used to acquire the Docker images and their related meta-information from [Docker Hub]. The Docker Hub data we collect is openly accessible to everyone, and it is permissible to do research on this data set.

Namespaces in Linux are the primary method of container separation. Using names-paces gives each container a unique view of the system. Additionally, namespaces may be used for network connectivity, mount points, process IDs and user IDs, inter-process communication as well as many other functions. Namespaces can be considered as containers in a collection [27]. Resource visibility is controlled through namespaces, which may be used to restrict access to resources in a container. Specifically, container processes have access to a variety of network interfaces, each with a unique IP address compared to other network interfaces on the host or other containers. Some domains, such as security subsystems, need more namespace support in order for containers to function independently. Namespace-less system are often inaccessible to containers.

Many system calls are implemented in Linux and only the root user of the host computer can utilise some of them. A good example of this is changing the system clock. As long as the containers are running on distinct hosts, each can have its own root user and execute privileged syscalls [28]. It is vital to limit access to the syscall interface to prevent container users from modifying the system clock, as namespaces do not isolate the system clock. An example of a technology that supports this is the Linux Capabilities. Linux includes 37 features that can be used to prohibit access to certain system calls or system calls with certain parameters, at this time. Deprives container apps of 24 of the 37 capabilities that Docker automatically removes for processes that it starts in a container. Applications running on virtual machines cannot adjust the system clock and do other tasks that need privileged access, such as activating or deactivating swap memory.

3.1 Threat Model

According to Figure 1, Docker Hub is vulnerable to two types of attacks. In Vulnerable Docker Image inclusion of meta information in Docker Hub images may result in the introduction of security flaws. Hackers can gain access to your personal information if you download and run unsafe images. Developers may also distribute commands that contain sensitive arguments, such as allowing containers root access to the host, which may present extra security vulnerabilities.

Sometimes, malicious Docker Hub images are uploaded with harmful run-commands included in the description. Docker Hub's security mechanisms allow malicious images and data to slip through the holes [29]. If a person runs a malicious image they've obtained, they may be exposed to cryptomining attacks. Host file leaks, on the other hand, can be caused by malicious run-commands. It is reasonable to assume that the tools we use to analyse Docker images in this study are not known by potential attackers. The conclusion is that the harmful code may be able to evade the scrutiny as a result.

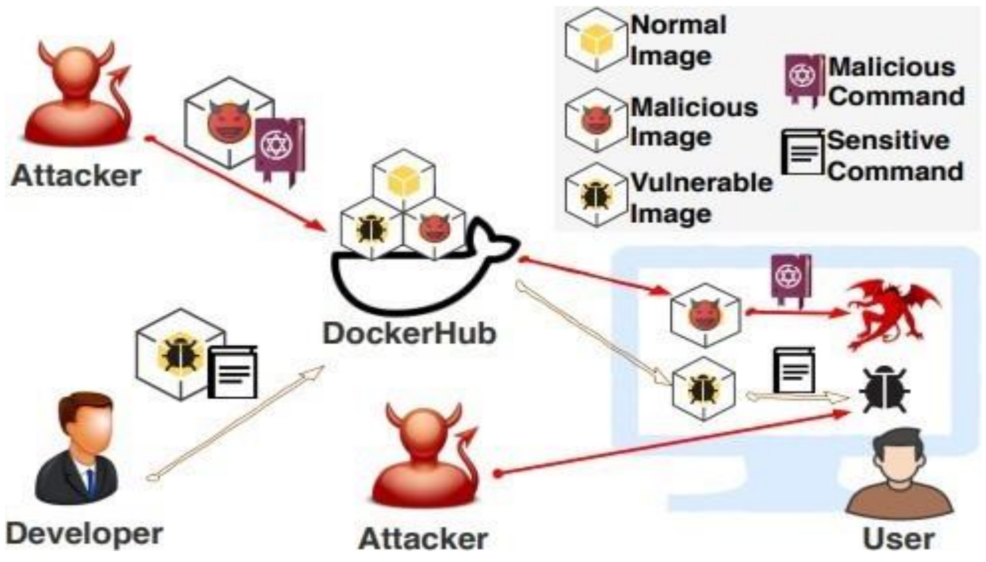

Fig 2: Threat Model Docker Security Analysis

Table 1: Data Set Collected from Docker hub

	Images	Developers	Repositories
Community	71	369860	985710
Official	13	1	151
Total	84	369861	985869

3.1.1 Analysis of proposed Framework

Security information may be found in Docker images on Docker Hub. The study is nonsensical since a large portion of the data has never been collected. The API supplied by [Docker Hub] is utilised by a custom web crawler to obtain the Docker images and their related meta-information. The data-set of Docker Hub is publicly available to everyone, and it is legal to do research on this data.

3.1.2 Collection of Data from Docker Hub

Table 1 shows that our data set contains directories containing more than 985,869 repositories. Each dataset in each archive contains images and associated metadata. The raw data also contains the following information and code that we extract for research purposes.

The run-commands and any sensitive parameters; should be carefully considered, as they can greatly affect the way containers behave. When a repository's description begins with "docker run," sensitive parameters are extracted from run-commands using string matching so that security analysis can be performed.

Collecting executed program: In order to protect a container, the security of the container's run programme must be considered. We create an automatic parser tool in order to locate and extract the running application. The entry-file for each image is identified by a Dockerfile or manifest. As soon as the parser encounters an entry-file, it begins searching for other files that are activated by that particular entry-file. Once again, the image's programmes are extracted, and the process is repeated. In order to assess ELF files and shell scripts, we’ve implemented a custom script interpreter that uses strings and a special interpreter for ELF files [29].

We can get a complete list of all the programmes that have been run on the images examined by using the parser. JAR, ELF, Shell Script, etc. are all file kinds that may be found in the extracted executable applications. Analyzing several different kinds of software at once is difficult since developing and verifying malware fingerprints takes a long time. Because of this, we resort to internet virus analysis programmes for assistance. Antivirus (AV) platforms such as Kaspersky, Symantec, and BitDefender use VirusTotal [29] are used for malware detection with different signature-based and anomaly-based detection methodologies.

3.1.3 Track the malicious code and program

Malware of all types, such as Trojans, Backdoor Trojans, and BitcoinMiners may be detected with this tool. As a result, we use Virus Total to do a preliminary scan. Previous research has demonstrated that VirusTotal may incorrectly classify harmless software as harmful [14, 22, 29]. When a sufficient number of antivirus vendors identify a programme as potentially harmful, earlier studies used that information to migrate false positives. In reality, there is no consensus on a threshold value for a certain disease. A threshold of two, four, or five has been used in previous detection methods [?]. At least five of the leading antivirus (AV) vendors deem software harmful in this article, and we define it as such. As a result of this method, the evaluated programmes are (nearly) appropriately categorised as either harmless or harmful. Aside than reporting the type of each antivirus company’s malware, the findings of the primary screening are all that are available. First stage screening findings are difficult to demonstrate, much alone the behaviour of each malware kind. As a result, once the main screening yields a list of potentially harmful files, a follow-up screening is required to verify the detections and examine the behaviour of the files in question. Additionally, we gather logs of system calls, network traffic, etc. to highlight security breaches caused by potentially dangerous files that are dynamically launched in a container.

A framework is then implemented to complete the aforesaid workflow. To begin, our framework makes use of parser to discover and extract running apps from Docker images. VirusTotal API is used to identify potentially harmful files. As a last step, we create a container that includes strace and tcpdump tools for security investigation. For example, we use this container for automatically launching and tracking potentially harmful files so that we may collect useful system logs. In order to decrease the number of system logs created, main screening filters out most non-malicious images. Automated detection of malicious images is substantially facilitated by the use of this framework.

3.1.4 Malicious Image Dissemination in Docker

There are no dangerous applications running in the newest images from 147 authoritative repositories, according to our first investigation. Extracting the following subsets from the dataset will allow us to conduct in-depth studies of community images. A list of the most popular images in the top 10,000 repositories is identified and randomly selected 100 most recent images from each category on basis of popularity rankings of the remaining community repositories.

4 Results

A single image or file is processed by our parser in Section 3.2 in 5 and 0.15 seconds on average.

The parser identifies 693,757 running processes in the images evaluated. After duplication, we are left with 36,584 distinct programmed that have been run, of which our framework has detected 13 as dangerous. In all, there are 17 images of the 13 harmful applications. Furthermore, we observe that all of the malicious programmed in these malicious images are entry files. This suggests that malicious images frequently use a single entry-file to carry out assaults, rather than relying on a series of triggered files.

Docker Hub is a good place to look for malicious images created by the same person. The harmful images that are associated should be checked. In the repositories, we notice two types of linked malicious images; Newest 10 images and most popular images.

A total of 48 and 84 linked images are identified; among them 24 images are detected that have the same malicious file. There are 186 new programmes that have been executed since the framework was built, with 20 of those applications being malicious. Using heuristic techniques like the ones presented here to analyses related images can be an excellent way to uncover harmful images and programmes, as this enlightening discovery shows. Each of the blocks that make up the pipeline must be defined in a file configuration inside the Logstash directory.

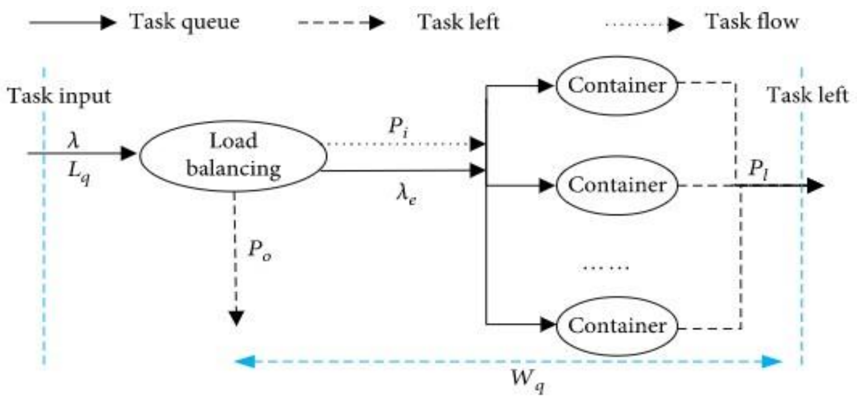

Fig 3: Docker model for task execution

Att_q is defined as the attack task queue length: the mathematical anticipation of the number of Attack which launched in the docker and handled in the proposed model. According to the Equations (1) and (3) restrictions their relationship identified and they exist:

$$Att_q = \sum_{m-1}^{L} (m-t) q_m$$

$$= \frac{q_p \rho^t \rho_t}{t!\,(1-\rho_t)^2} [1 - \rho_t^{L-t+1} - (1-\rho_t)(L-t+1)\rho_t^{L-t}]$$

Where $\rho = \lambda/\mu, \rho_s = \lambda/t\mu$ expressing the level of service intensity, which is a reflection of the system's performance.

$$P_p = \left(\sum_{m=0}^{t-1} \frac{\rho^m}{m!} + \frac{\rho^t (1-\rho_t^{L-t+1})}{t!\,(1-\rho_t)} \right)^{-1}$$

$$q_m = \frac{\rho^m}{t!\, t^{m-t}} q_0$$

From these eEquations W_r can be obtained.

$$W_r = \frac{M_r}{\lambda_f}$$

$\lambda_e = \lambda\ (1- p_k)$ reflects the pace at which jobs are completed. An issue with the cluster service caused the task request's effective arrival rate to be higher than expected. The reason for this is that jobs with probability Q_o can't be handled adequately by cluster service, but tasks with probability Q_i can. As a result, we may arrive to the arrangement Equation in a straightforward manner.

$$W_q = \frac{r_q}{\lambda_e} = \frac{\sum_{m=1}^{K} (m-s) p_m}{\lambda P_i} = \frac{\sum_{m=1}^{K} (m-t) p_m}{\lambda (1-P_o)}$$

It represent the existence of QoS constraints, so it must satisfy the $W_q < W_{q\max}$, the relationship to execute the docker program.

$$W_r = \frac{R_r}{\lambda_e} = \frac{\sum_{n=1}^{K} (m-t) p_m}{\lambda P_i} = \frac{{}_{m=1}^{K} (m-t) p_m}{\lambda (1-P_o)} < W_{r\max},$$

$$E[W_r | t = t] < W_{rmax} \text{ and } E[W_r | t = t-1] W_{rmax},$$

$$U_{down} < E\left[\sum_{i=1}^{t} \frac{U_i}{t} | t = t \right] < U_{up}$$

$$U_{down} > E\left[{}_{i=1}^{K} \frac{U_i}{t} | t = t+1 \right] \text{ or } EE\left[{}_{i=1}^{K} \frac{U_i}{t} | t = s - 1 \right] > U_{up}$$

here Equation show that attack W_r is related with resource in cluster t, K, λ, and μ, and since λ, μ, and K independent if the u share the same Network, t Qot constraints are a collection of viable solutions, and the solution space is the set of all feasible solutions. As a result, finding the best cluster index is a challenging task.

Relevant information of the analyzed traffic in logs at the location / usr / local / bro / logs / current, these logs are written in ASCII format, organized in columns and separated by tabs. The possible records or logs generated by Zeek from the analyzed traffic are indicated in the table3.

Table 2: Log file Location to analysis

Log file	Depiction
files.log	File analysis
dns.log	DNS requests
http.log	HTTP requests and responses
conn.log	TCP, UDC and ICMP connections

ssh.log	SSH connections
notice.log	Detection notifications
intel.log	Intelligence Framework data matches

The logs that will allow the identification of anomalies are following:

intel.log is one of the most remarkable feature of Zeek is the possibility of working jar with its Intelligence Framework, which allows consuming data from different sources with relevant and updated information on potential threats. The main element of intelligence data is the indicator, which can be an IP address, email address, hash of a file or certificate, subnet, domain, etc. In addition, the indicator contains a set of metadata to provide more information about the security event. Intelligence data is usually generated by incident response processes and is available for free use in many cases. For this project, intelligence data will be produced through integration with a threat management system called CIF (Collective Intelligence Framework), which allows combining information from different feeds (datasets) and consuming them in order to identify and detect malicious activity.

notice.log contains security events related to port scanning, brute force attacks, SQL injection attacks, invalid SSL certificates, among others are sent to this registry.

conn.log contains session data between two nodes is security related information. The conn.log record stores all connections established through TCP / UDP and ICMP network protocols. The storage of the logs can be done in a JSON format facilitating the subsequent ingestion of these logs in the collection tool, this can be done by using a script and loading it into the /local.bro file. For ease, this format modification will be carried out directly in the tool for collecting all the data from different sources.

What's more, on the file /local.bro se define the script that has to run, Directives necessary for the operation of intelligence data were added to the default configuration. The content of this file is indicated below setting.

4.0.1 Tickets generation in Docker

They allow to establish the source of the logs sent, in this case it is necessary to define This information is sent from the Filebeat agent through the listening TCP port 5044, the port published in the container for establishing the connection between Filebeat and Logstash.

4.0.2 Filters

Filters perform event processing in order to transform the data into a defined format or normalize it. Basically, the complexity in the implementation of the pipeline lies in the filter block. For example, the following Steps are performed at this stage for the conn.log records:

(1) Elimination of comments located at the top of the log files. Regular expressions are used to remove all lines that start with.

(2) Elimination of the tabs using the CSV filters to carry out the transformation of the data. Although the separation of the columns is not carried out

(3) Transformation of the time stamp (timestamp) that Zeek establishes from UNIX type to date type, this is done using a filter that Logstash has.

A 1560212576.046810 -> Jun 11, 2019 0: 22: 56.046

Finally, it is necessary to carry out two tasks: modify the names of the fields. It is necessary to use the filter called Mutate to modify the names of fields that contain points, because Logstash has certain disadvantages in processing these characters. For example, the following change will be made to the names of the fields for the logs.

id. resp_p => id_resp_port id. orig_h => id_orig_host id. resp_h => id_resp_host id. orig_p => id_orig_port

At the same time, it supports data types in its scripting language:

id. resp_p => integer duration => float orig_bytes => integer missed_bytes => integer id. orig_p => integer resp_bytes => integer

Data normalisation is achieved by the creation of filters for each of the log files that Logstash processes. An image's release and update dates and times must be determined exactly as part of the Docker life cycle in order to fully grasp the chronology. We can't reliably predict the exact timing because of these challenges. Several vulnerabilities have been addressed numerous times. As fresh data is discovered, it is feasible that different manufacturers will announce different timetables. Due on earlier research, these times are proposed to be defined in the following manner: There has been a software vulnerability discovered and deemed dangerous to the general population on this day. For vulnerability, patch-time refers to when a workaround or patch is made available by the software vendor or the product's originator to guard against its exploitation. As an alternative to reporting a vulnerability that has been fixed by software update, we instead publish the date of such update.

4.1 Leakage of Host File detection

For detection of anomalies we use the following Data set as input and output.

Input: the type of entry and the routes where the records of interest are stored are defined. The following table indicates the defined paths for the records generated.

Table 3: Generated Log Paths got Anomalies Detection

Item	Log Route	Description
intel.log	/usr/local/bro/logs/current/intel.log	Intelligence information
notice.log	/usr/local/bro/logs/current/notice.log	Detected anomalies
conn.log	/usr/local/bro/logs/current/conn.log	Session data

Output: The IP address of the server and the default listening port, 5044, must be entered in the output. We demonstrate how to leak user files from the host with the critical attributes shown in Figure 3. In order to mount a volume from the host to the container, use the -volume or option. A -volume on the host can be made available to the container using the -v src: dest parameter. Attackers can use this method to upload user data from the host disc to a remote internet repository. Docker images include critical characteristics, thus we looked at how they were distributed. Many Docker repositories contained descriptions that included the proposed run instructions, according to our research. The total number of sensitive parameters in these commands is 81,294 since each command has an average of one sensitive parameter (see Table 1). In light of how widely they are used and the significant impact they have on security, we must do everything we can to improve user knowledge of the security threats posed by sensitive parameters and to provide approaches that can effectively identify these hazards.

5 Conclusions

Runtime security can be provided by node-based solutions in a number of different ways, such as at the orchestration layer (Kubernetes admission controller) or the application layer (Node level agent, privileged container). While effective, these consequences are either overly restrictive (such as requiring complex deployment checks via the Kubernetes admission controller) or ineffective (such as requiring simple deployment checks. Although security hooks are being incorporated into container-as-a-service environments, these efforts are still in their infancy and may not be fully implemented across your entire container infrastructure. The inability to install an agent on the host or node makes it challenging to migrate from traditional node-based approaches (which have their own benefits) to modern Container as a Service environments (such as AWS Fargate, Azure Container Instances, and Google CloudRun).

References

[1] Jiang, Y., Liu, W., Shi, X., Qiang, W. (2021). Optimizing the copy-on-write mechanism of Docker by dynamic prefetching. *Tsinghua Science and Technology*, *26*(*3*), 266–274.

[2] Kwon, S., Lee, J.-H. (2020). Divds: Docker image vulnerability diagnostic system. *IEEE Access*, *8*, 42666–42673.

[3] Zhao, N., Tarasov, V., Albahar, H., Anwar, A., Rupprecht, L., et al. (2021). Large-scale analysis of docker images and performance implications for container storage systems. *IEEE Transactions on Parallel and Distributed Systems*, *32*(*4*), 918–930.

[4] Xie, Y., Jin, M., Zou, Z., Xu, G., Feng, D., et al. (2020). Real-time prediction of docker container resource load based on a hybrid model of arima and triple exponential smoothing. *IEEE Transactions on Cloud Computing*, 1–1.

[5] Jin, H., Wang, Y., Wang, Q., Liu, J., Wang, S., et al. (2019). Architecture modelling and task scheduling of an integrated parallel cnc system in docker containers based on colored petri nets. *IEEE Access*, *7*, 47535–47549.

[6] Divya, V., Sri, R. L. (2021). Docker-based intelligent fall detection using edge-fog cloud infrastructure. *IEEE Internet of Things Journal*, *8*(*10*), 8133–8144.

[7] Melo, L., Wiese, I., d.Amorim, M. (2021). Using docker to assist q amp;a forum users. *IEEE Transactions on Software Engineering*, *47*(*11*), 2563–2574.

[8] Sollfrank, M., Loch, F., Denteneer, S., Vogel-Heuser, B. (2021). Evaluating docker for lightweight virtualization of distributed and time-sensitive applications in industrial automation. *IEEE Transactions on Industrial Informatics*, *17*(*5*), 3566–3576.

[9] Ma, L., Yi, S., Carter, N., Li, Q. (2019). Efficient live migration of edge services leveraging container layered storage. *IEEE Transactions on Mobile Computing*, *18*(*9*), 2020–2033.

[10] Zou, Z., Xie, Y., Huang, K., Xu, G., Feng, D., et al. (2019). A docker container anomaly monitoring system based on optimized isolation forest. *IEEE Transactions on Cloud Computing*, 1–1.

[11] Lu, Z., Xu, J., Wu, Y., Wang, T., Huang, T. (2019). An empirical case study on the temporary file smell in dockerfiles. *IEEE Access*, *7*, 63650–63659.

[12] Sami, H., Mourad, A., El-Hajj, W. (2020). Vehicular-obus-as-on-demand-fogs: Resource and context aware deployment of containerized micro-services. *IEEE/ACM Transactions on Networking*, *28*(*2*), 778–790.

[13] Diekmann, C., Naab, J., Korsten, A., Carle, G. (2019). Agile network access control in the container age. *IEEE Transactions on Network and Service Management*, *16*(*1*), 41–55.

[14] Sairam, R., Bhunia, S. S., Thangavelu, V., Gurusamy, M. (2019). Netra: Enhancing iot security using nfv-based edge traffic analysis. *IEEE Sensors Journal*, *19*(*12*), 4660–4671.

[15] Wu, Y., Zhang, Y., Wang, T., Wang, H. (2020). Characterizing the occurrence of dockerfile smells in open-source software: An empirical study. *IEEE Access*, *8*, 34127–34139.

[16] Bellavista, P., Corradi, A., Foschini, L., Scotece, D. (2019). Differentiated service/data migration for edge services leveraging container characteristics. *IEEE Access*, *7*, 139746–139758.

[17] Mahmud, R., Toosi, A. N. (2021). Con-pi: A distributed container-based edge and fog computing framework. *IEEE Internet of Things Journal*, 1–1.

[18] Kim, T., Al-Tarazi, M., Lin, J.-W., Choi, W. (2021). Optimal container migration for mobile edge computing: Algorithm, system design and implementation. *IEEE Access*, 1–1.

[19] Cai, L., Qi, Y., Wei, W., Li, J. (2019). Improving resource usages of containers through auto-tuning container resource parameters. *IEEE Access*, *7*, 108530–108541.

[20] Tsung, C.-K., Hsieh, H.-Y., Yang, C.-T. (2019). An implementation of scalable high throughput data platform for logging semiconductor testing results. *IEEE Access*, *7*, 26497–26506.

[21] Cinque, M., Della Corte, R., Pecchia, A. (2019). Microservices monitoring with event logs and black box execution tracing. *IEEE Transactions on Services Computing*, 1–1.

[22] Jimenez, L. L., Schelen, O. (2020). Hydra: Decentralized location-aware orchestration of containerized applications. *IEEE Transactions on Cloud Computing*, 1–1.

[23] Karn, R. R., Kudva, P., Huang, H., Suneja, S., Elfadel, I. M. (2021). Cryptomining detection in container clouds using system calls and explainable machine learning. *IEEE Transactions on Parallel and Distributed Systems*, *32*(*3*), 674–691.

[24] Nakata, R., Otsuka, A. (2021). Cyexec*: A high-performance container-based cyber range with scenario randomization. *IEEE Access*, *9*, 109095–109114.

[25] Ramanathan, S., Kondepu, K., Razo, M., Tacca, M., Valcarenghi, L., et al. (2021). Live migration of virtual machine and container based mobile core network components: A comprehensive study. *IEEE Access*, *9*, 105082–105100.

[26] Karn, R. R., Kudva, P., Elfadel, I. A. M. (2019). Dynamic autoselection and autotuning of machine learning models for cloud network analytics. *IEEE Transactions on Parallel and Distributed Systems*, *30*(*5*), 1052–1064.

[27] Epiphaniou, G., Pillai, P., Bottarelli, M., Al-Khateeb, H., Hammoudesh, M., et al. (2020). Electronic regulation of data sharing and processing using smart ledger technologies for supply-chain security. *IEEE Transactions on Engineering Management*, *67*(*4*), 1059–1073.

[28] Sultan, S., Ahmad, I., Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead. *IEEE Access*, *7*, 52976–52996.

[29] Xu, R., Jin, W., Kim, D. (2021). Enhanced service framework based on microservice management and client support provider for efficient user experiment in edge computing environment. *IEEE Access*, *9*, 110683–110694.

[30] Souquet, L., Talbi, E. G., Nakib, A. (2020). Fractal decomposition approach for continuous multiobjective optimization problems. *IEEE Access*, *8*, 167604–167619.

[31] Sami, H., Mourad, A. (2020). Dynamic on-demand fog formation offering on-the-fly iot service deployment. *IEEE Transactions on Network and Service Management*, *17*(*2*), 1026–1039.

[32] Cui, H., Zhou, Y., Wang, C., Wang, X., Du, Y., et al. (2021). Ppsb: An open and flexible platform for privacy-preserving safe browsing. *IEEE Transactions on Dependable and Secure Computing*, *18*(*4*), 1762–1778.

[33] Kafle, V. P., Muktadir, A. H. A. (2020). Intelligent and agile control of edge resources for latencysensitive iot services. *IEEE Access*, *8*, 207991–208002.

[34] Kim, A., Park, M., Lee, D. H. (2020). Ai-ids: Application of deep learning to real-time web intrusion detection. *IEEE Access*, *8*, 70245–70261.

[35] Cai, B., Li, K., Laiping, Z., Zhang, R. (2020). Less provisioning: A hybrid resource scaling engine for long-running services with tail latency guarantees. *IEEE Transactions on Cloud Computing*, 1–1.

[36] Vedant Bhatt, Harvinder Singh Diwan, S. K. A., Yashika Saini (2021)**,** Empowering ML Work-Flow with DevOps within Micro Service Architecture and Deploying A Hybrid-Multi Cloud, Maintaining CI/CD Pipeline: An Open Shift Orchestration of ML-OPS. International Journal of Contemporary Architecture The New ARCH, e-ISSN: 2198-7688**,** Vol. 8 No. 2 (2021): Vol. 8 No. 2.

[37] Ashutosh Kumar & S. K. A. (2018)**.** Implementation of new Cryptographic Encryption Approach for Trust as a Service (TAAS) in Cloud Environment. International Journal of Computer Application (2250-1797) Issue 8 Volume 4, https://dx.doi.org/10.26808/rs.ca.i8v4.03