

Securing Containers: Honeypots for Analysing Container Attacks

*A Seminar Report
Submitted by*

Arjun Sable
Roll No 22M1528

Under guidance of
Prof. Manjesh Kumar Hanawal



Indian Institute of Technology, Bombay
Powai – 400 076, Maharashtra, India

May 2023

Abstract

With the help of Docker, an application and all of its dependencies may be packaged into a standardised, self-contained unit, or "container," that can be used for software development and operate on any platform. Dockerfiles are declarative specifications of an environment designed to provide replicability in container builds. They're frequently located in source code repositories and allow the hosted software to operate in its intended environment. The Docker ecosystem has rapidly gained popularity and has become a crucial component in the software development and deployment process. Empirical analysis of the Docker ecosystem can provide valuable insights into its adoption, usage patterns, and potential challenges.

The use of Docker containers has become increasingly popular in recent years for various applications, including Operations Research (OR). OR is a field of study that uses mathematical models and algorithms to improve decision-making processes in complex systems. Docker containers provide a range of benefits that can enhance OR workflows, including improved reproducibility, portability, scalability, and simplification of software installation and setup.

Docker Containers are increasingly being used to develop, deploy and distribute software. However, they are also vulnerable to various attacks resulting in breaches and access to the host machines. To understand the various attacks on the containers and study them in a sandbox environment, we develop a honeypot. The honeypot systematically collects data logs of all the activities both inside and outside the container. We build on open source log gathering tool Osquery and Docker APIs to obtain 'evented' activity logs. Our honeypot provides granular information about the container activities which can be analysed to build detection rules and secure the containers.

Table of content

Abstract	i
1. Introduction	1
1.1 Introduction to Docker container ecosystem	1
1.1.1 Overview of Docker Container Ecosystem	1
1.1.2 Background	3
1.1.3 Advantages	3
1.1.4 Empirical Analysis	4
1.1.5 Current State of the Docker Ecosystem	5
1.1.6 Future Potential of the Docker Ecosystem	5
1.1.7 Challenges Facing the Docker Ecosystem	6
1.2 Introduction to Operations Research and its Computational Challenges .	6
1.3 Security challenges in container environments	7
2. Literature survey	10
2.1 Introduction of Honeypots	10
2.2 Working of Honeypot	10
2.3 Existing Tools and Limitations	10
2.3.1 Docker bench for security	11
2.3.2 Clair	12
2.3.3 Cilium	13
2.3.4 Anchore	14
2.3.5 OpenSCAP Workbench	13
2.3.6 Sysdig	13
3. Log collection for containers	15
3.1 Proposed solution	15
3.2 Deploying the Container Honeypot	15

3.3 Querying container for logs	15
3.4 Obtaining queried information	16
4. Working with Sysdig	18
4.1 Introduction to Sysdig	18
4.2 Using csysdig tool	19
4.3 Process monitoring	22
4.4 Network monitoring	24
4.5 Container monitoring	25
4.6 File accesses	26
5. Conclusion	27
Bibliography	28

List of Figures

1.1	Comparison of container versus virtual machine	2
1.2	Popularity of containers	5
2.1	Sysdig architecture	14
3.1	Container honeypot architecture	16
3.2	Docker events table	17
3.3	Process events inside docker container	17
4.1	Csysdig usage	22
4.2	Csysdig Processes view for host	23
4.3	Csysdig Connections view for host	25
4.4	Csysdig Containers view for host	25
4.5	Csysdig Files view for host	26

List of Tables

4.1	Views in csysdig	21
4.2	Processes view	22
4.3	Csysdig Connections view	24
4.4	Csysdig Connections view	25
4.5	Csysdig Files view	26

Chapter 1

Introduction

1.1 Introduction to Docker container ecosystem

1.1.1 Overview of Docker Container Ecosystem :

Docker is a popular platform for developing, deploying and running applications in containers. Containers provide a lightweight and isolated environment for applications to run, which makes it easier to manage and scale applications across different environments. The Docker ecosystem is a collection of tools, services, and technologies that work together to make it easier to build, deploy, and manage applications in containers. In this report, we will conduct an empirical analysis of the Docker ecosystem to understand its current state and its potential for future development.

Docker offers a few features that are helpful to administrators and developers. The operating system (OS) libraries and dependencies required to run the application source code in any context are included in containers, which are standardised, executable components. The open source platform Docker makes it possible for programmers to create, distribute, launch, operate, update, and manage containers. Containers facilitate the creation and distribution of distributed applications. As businesses switch to cloud-native development and hybrid multi cloud settings, containers have grown in popularity. It's possible for developers to create containers without Docker, by working directly with capabilities built into Linux and other operating systems. But Docker makes containerization faster, easier and safer.

Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. The capabilities such as control groups (Cgroups) for allocating resources among processes, and namespaces for restricting a processes access or visibility into other resources or areas of the system enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple virtual machines (VMs) to share the CPU, memory and other resources of a single hardware server.

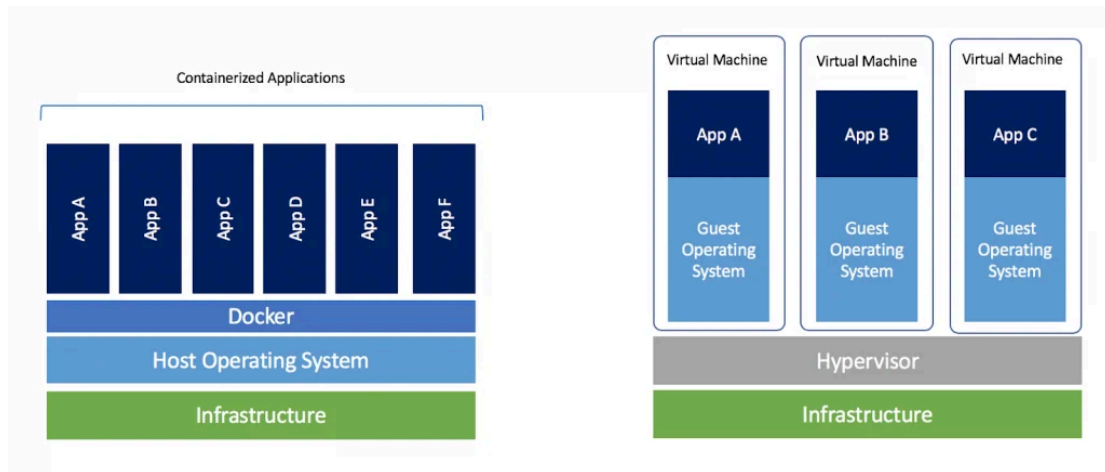


Fig 1.1 Comparison of container versus virtual machine

Docker containers are an emerging technology that allows applications to be packaged and run consistently across different environments. This technology has several benefits that make it well-suited for the needs of operations research, including easy deployment, reproducibility, and scalability. Docker containers can be used to simplify the setup and deployment of operations research models and algorithms. Docker containers can also be used in parallelizing operations research algorithms, including better scalability and improved performance.

Docker containers can be used to create a reproducible research environment, and can be used to parallelize an optimization algorithm to achieve faster runtimes. Docker can be used to streamline the development and deployment of operations research models and algorithms.

Operations research (OR) is an interdisciplinary field that involves the application of mathematical models, algorithms, and decision-making techniques to optimize complex systems. In recent years, OR has become increasingly reliant on computational techniques, with researchers developing sophisticated algorithms and models to solve complex problems.

One of the key challenges in OR is the development and deployment of these algorithms and models. Often, researchers use a variety of tools and software packages to develop and test their models, which can lead to compatibility issues and make it difficult to reproduce results. Furthermore, OR algorithms and models can be computationally intensive, requiring significant resources to run efficiently.

In this context, Docker containers have emerged as a promising technology for simplifying the development and deployment of OR models and algorithms. Docker containers provide a lightweight, portable way to package software and dependencies,

making it easier to deploy applications consistently across different environments. This technology has several benefits that make it well-suited for the needs of OR, including easy deployment, reproducibility, and scalability.

1.1.2 Background:

Docker was first introduced in 2013 and quickly gained popularity among developers due to its simplicity and ease of use. The Docker platform allows developers to build applications in containers and deploy them anywhere, making it easier to manage and scale applications across different environments. Over the years, the Docker ecosystem has grown and evolved, and it now includes a wide range of tools, services, and technologies that work together to make it easier to build, deploy, and manage applications in containers.

1.1.3 Advantages :

Application isolation, cost-effective scalability, and disposability are just a few of the features and advantages that container technology offers in addition to several other crucial advantages such as:

- **Lighter:** Unlike virtual machines (VMs), containers do not transport the whole OS instance and hypervisor with them. They only contain the OS dependencies and processes required for the code to run. Container sizes are measured in megabytes on other hand VMs sizes are in gigabytes.
- **Improved developer productivity:** Applications that use containers can be created once and executed anywhere. Additionally, containers are quicker to deploy, provision, and restart than virtual machines. They are better suited for development and are perfect for usage in CI/CD pipelines (continuous integration and delivery).

Due to Docker's broad usage, the terms "Docker" and "containers" are now frequently used interchangeably. The fundamental container technology, however, has been available for many years. Using simple commands and an application programming interface (API), Docker enables developers to automate these native containerization capabilities. Application deployment into containers can be automated with the help of Docker.

Docker adds an additional layer of deployment engine on top of the virtualized and performed programmes in a container environment. Docker is made to provide a speedy, lightweight environment where code may execute effectively, and it also offers the additional

capability of a professional work procedure to remove the code from the computer for testing prior to production.

Containers, Docker Images, Docker Registries, and Docker Client and Server are the four primary core parts of Docker.

1. **Docker client and server** :The docker server receives the request and executes it as necessary. Docker has the complete RESTful (Representational state transfer) API in addition to a binary command-line client. A Docker daemon/server and client can run on the same computer, or a local Docker client can connect to a remote server or daemon running on another machine.
2. **Docker image** : There are two ways to construct an image. The first step entails utilizing a read-only template to create an image. Every image has a base image as its basis. The operating system images build a container with the capacity to fully operate the OS. The second approach involves producing a docker file. When the "Docker build" command is executed from the bash terminal, it follows all the instructions in the docker file and creates an image. The docker file comprises a list of instructions.
3. **Docker registries** : Docker registries house Docker images. Similar to source code repositories, it allows for pushing and pulling of photos from a single source. Public and private registrations are the two different categories. The public registry known as Docker Hub allows users to push and pull pre-existing images without having to start from scratch.
4. **Docker container** : A docker container is created by a docker image. Containers store everything needed for a programme, allowing it to execute independently. For instance, if there is an image of the Ubuntu OS that includes SQL SERVER, running this image with the docker run command will result in the creation of a container that runs the Ubuntu OS and SQL SERVER.

1.1.4 Empirical Analysis:

To conduct our empirical analysis of the Docker ecosystem, we collected data from various sources, including online forums, news articles, and reports, as well as conducting surveys and interviews with developers and users of the Docker platform. Our analysis focused on three main areas: the current state of the Docker ecosystem, its future potential, and the challenges it faces.

1.1.5 Current State of the Docker Ecosystem:

The Docker ecosystem is currently experiencing significant growth and development. In recent years, the popularity of containers has increased, and this has driven the growth of the Docker ecosystem. The ecosystem now includes a wide range of tools and services for building, deploying, and managing containers, and various cloud-based services for deploying and managing containers.

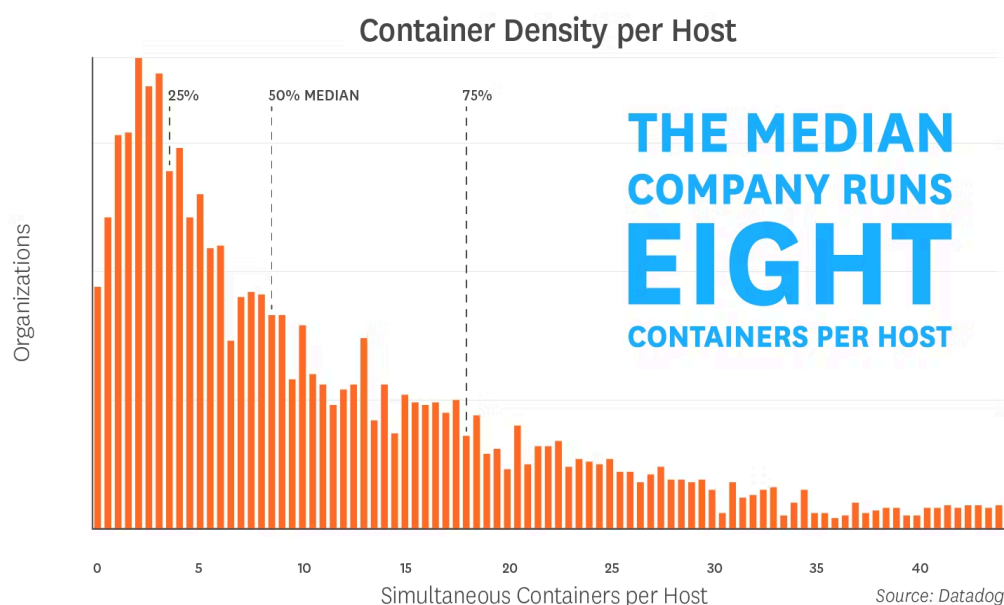


Fig 1.2 Popularity of containers

1.1.6 Future Potential of the Docker Ecosystem:

The future potential of the Docker ecosystem is very promising. As the popularity of containers continues to grow, the Docker ecosystem is poised for further growth and development. This growth will be driven by the continued development of new tools and services for building, deploying, and managing applications in containers, as well as the increasing adoption of containers by businesses and organisations. In addition, the rise of cloud computing and the growth of cloud-based services for deploying and managing containers will also play a significant role in the future growth of the Docker ecosystem.

1.1.7 Challenges Facing the Docker Ecosystem:

Despite its growth and potential, the Docker ecosystem faces several challenges that must be addressed in order for it to continue to thrive. One of the biggest challenges facing the Docker ecosystem is the lack of standardisation and compatibility between different tools and services. This can make it difficult for developers and organisations to integrate different tools and services into their workflows, and it can also make it difficult to ensure that applications run consistently across different environments.

Another challenge facing the Docker ecosystem is the lack of security features. Although containers offer many benefits for security and isolation, they also pose new security risks that must be addressed. For example, containers can be vulnerable to attacks, and there is a risk of data loss or theft if containers are not properly secured.

1.2 Introduction to Operations Research and its Computational Challenges

Operations Research (OR) is an interdisciplinary field that applies mathematical, statistical, and computational techniques to solve complex problems and optimize systems. OR is used in a wide range of applications, including manufacturing, logistics, transportation, healthcare, and finance.

OR involves developing mathematical models that represent real-world systems and using algorithms to find optimal solutions to these models. However, these models can be computationally intensive and require significant resources to run efficiently. Moreover, OR researchers often use a variety of tools and software packages to develop and test their models, which can lead to compatibility issues and make it difficult to reproduce results.

The computational challenges of OR are further complicated by the increasing complexity of modern systems, the volume and variety of data available, and the need to make decisions quickly in response to changing conditions. These challenges require the development of new algorithms and techniques that can handle large-scale, complex systems and provide fast, accurate solutions.

To address these challenges, OR researchers have turned to computational techniques such as optimization, simulation, and data analytics. These techniques involve the use of mathematical models, algorithms, and statistical analysis to find the best solution or make predictions about the performance of a system.

However, developing and deploying these algorithms and models can be challenging due to the need to integrate multiple software tools and packages, manage dependencies, and ensure reproducibility of results. This is where the Docker container ecosystem comes in, providing a streamlined way to package and deploy software and dependencies, ensuring consistency across different environments, and improving the reproducibility of research results.

1.3 Security challenges in container environments

While containerization brings numerous benefits, it also introduces specific security challenges that need to be addressed. Here are some common security challenges in container environments:

1. **Container Breakouts:** Container breakouts occur when an attacker gains unauthorized access to the underlying host system from within a container. This can happen due to vulnerabilities in the container runtime or misconfigurations. Once the attacker breaks out of a container, they can potentially compromise other containers or the host system. Proper isolation and hardening of the host system are crucial to mitigate this risk.
2. **Vulnerabilities in Container Images:** Containers are typically built from pre-existing images available from public repositories or custom images created by developers. These images may contain outdated or vulnerable software components. If an image has unpatched vulnerabilities, an attacker can exploit them to gain unauthorized access or perform malicious activities. Regular vulnerability scanning and updating of container images are essential to mitigate this risk.
3. **Insecure Configuration:** Inadequate or insecure configurations can leave containers exposed to attacks. For example, running containers with excessive privileges or using weak access controls can increase the risk of unauthorized access. It is important to follow security best practices and configure containers with the principle of least privilege to limit the potential attack surface.

4. **Orchestrator Security:** Container orchestration platforms like Kubernetes are commonly used to manage container deployments. However, the security of the orchestrator itself is critical. Insecure configurations, weak access controls, and insufficient network security within the orchestrator can lead to unauthorized access, data breaches, or disruption of services.
5. **Container Image Integrity:** Ensuring the integrity of container images is crucial. If an attacker modifies a container image before deployment, it could lead to the execution of malicious code or unauthorized access. Employing secure image registries, digital signatures, and image validation mechanisms can help maintain the integrity of container images.
6. **Insider Threats:** Container environments can be susceptible to insider threats, where authorized users intentionally or unintentionally misuse privileges or access confidential data. Implementing proper access controls, monitoring user activities, and conducting regular security audits can help mitigate the risks associated with insider threats.
7. **Data Protection:** Containers often process sensitive data, and securing this data is essential. Proper encryption of data at rest and in transit, using secure communication channels, and implementing strong access controls are vital for protecting sensitive information within containers.
8. **Container Lifecycle Management:** Containers have a shorter lifespan compared to traditional deployments. Managing container lifecycles, including proper disposal of containers, can be challenging. Forgotten or unused containers may contain vulnerabilities and become an attractive target for attackers. Implementing proper container lifecycle management practices, such as regularly removing unused containers and implementing automated processes for container disposal, can mitigate this risk.
9. **Regulatory Compliance:** Container environments must comply with various regulatory requirements, such as data privacy regulations (e.g., GDPR) or industry-specific standards (e.g., PCI-DSS). Ensuring that containers meet these

compliance requirements regarding data protection, access controls, and auditing is crucial.

10. Addressing these security challenges requires a multi-layered approach that includes secure configuration practices, regular vulnerability management, robust access controls, container image security, monitoring and logging, network security, and proper training and awareness for developers and operations teams working with containers.

Chapter 2

Literature survey

2.1 Introduction of Honeypots

A "honeypot" is a network-attached device that is used as a ruse to draw internet attackers. Honeypots are used as a security measure to study hacking attempts to gain unauthorised access to information systems. To entice attackers, this system has been deliberately made insecure. A honeypot's function is to pose online as a server or high-value item that might be targeted by attackers. Additionally, it gathers information and notifies defenders of any unauthorised users that attempt to use the honeypot. Large companies and organisations involved in cybersecurity research regularly employ honeypots to identify and defend against attacks from advanced persistent threat (APT) actors. Large businesses utilize honeypots as a crucial tool to establish an active defence against attackers or for cybersecurity researchers who are interested in learning more about the methods and instruments employed by attackers.

2.2 Working of Honeypot

A computer, software, and data used in a honeypot operation frequently resemble the behaviour of a real system that an attacker could find enticing, such as a financial system, internet of things (IoT) devices, a public utility, or a transportation network. It seems to be a part of a network despite being isolated and being closely watched. Any attempts to interact with a honeypot are viewed as hostile because it has no reason to be accessible by authorised users. Honeypots can also be positioned outside the external firewall, facing the internet, to collect information on efforts to access the internal network. The precise location of the honeypot varies depending on its complexity, the kind of traffic it seeks to draw, and its proximity to key corporate network resources. No matter where it is located, it will always be somewhat cut off from the production environment.

2.3 Existing Tools and Limitations

Users can log events from a Docker container using a tool that Docker makes available. This tool is a typical utility offered by Docker to keep track of events occurring in

Docker containers and analyse them to understand how they behave internally when moving. The docker events utility gives details on events involving containers, images, plugins, services, etc. The following events can be tracked with this utility:

- **Containers:** Container-related activities like attach commit, copy, create, and destroy
- **Images:** Image-related activities like delete, import, load, and pull
- **Plugins:** Plugins-related activities like enable and disable
- **Volumes:** Volume-related activities like destroy and mount
- **Networks:** Networks-related activities like connect, destroy, and disconnect
- **Daemons:** Daemons-related activities like reload
- **Services:** Services-related activities like create and remove
- **Nodes:** Nodes-related activities like remove and update
- **Secrets:** Docker Secrets-related activities like create and remove
- **Configs:** Configurations-related activities like create and update

Although several container monitoring tools are accessible to the public, they often lack detailed container information. For instance, Osquery is an example of such a tool that fails to offer in-depth insights into container events. Understanding and contextualizing logging of Process Events, File Events, and Socket events are vital for comprehensive threat analysis. Due to its limited container visibility, Osquery alone cannot ensure complete container security. Therefore, additional open-source tools for container security include:

2.3.1 Docker bench for security :

One notable open-source tool for container security is "Docker Bench for Security." Docker Bench for Security is a script provided by the Center for Internet Security (CIS) that automates the process of checking Docker containers and Docker hosts against best practices for security. It performs a series of tests based on the Docker CIS Benchmark, which is a consensus-developed industry-standard for securing Docker deployments.

By running Docker Bench for Security regularly, you can assess the security posture of your Docker environment and identify potential vulnerabilities or misconfigurations. It helps ensure that your Docker containers and hosts adhere to recognized security standards, reducing the risk of security breaches.

It's important to note that while Docker Bench for Security is a valuable tool, it should be used in conjunction with other security measures and best practices. Container security is a multifaceted discipline that requires a comprehensive approach, including secure configurations, vulnerability management, access controls, monitoring, and more

Docker Bench will scan your host platform for the following vulnerabilities:

1. General configuration
2. Linux host-specific configuration
3. The Docker daemon configuration
4. All Docker daemon configuration files
5. Container images and build files
6. Container runtime
7. Docker security operations
8. Docker swarm configuration
9. Docker Enterprise configuration
10. Docker trusted registry configuration

2.3.2 Clair :

Clair is an open-source container vulnerability scanner developed by CoreOS. It is designed to provide static analysis of container images and detect known vulnerabilities in their software packages and libraries. Clair is commonly used as part of the container security workflow to assess the risk associated with containerized applications.

Here's how Clair works:

- **Image Analysis:** Clair analyzes container images by extracting metadata and inspecting the software packages and libraries contained within them. It retrieves information such as package names, versions, and associated vulnerabilities.
- **Vulnerability Detection:** Using the extracted information, Clair compares the package versions against a database of known vulnerabilities, which is regularly updated with information from various sources such as the National Vulnerability Database (NVD). It identifies any known vulnerabilities that match the packages found in the container image.
- **Vulnerability Reporting:** Once the analysis is complete, Clair generates a report that provides details about the vulnerabilities found in the container image. The report

includes information such as the severity level of the vulnerabilities, recommended fixes or mitigation steps, and references to additional resources.

2.3.3 Cilium :

Cilium is an open-source networking and security solution for container environments, particularly designed for orchestrators like Kubernetes. It focuses on providing advanced network visibility, security, and load balancing capabilities for microservices-based applications.

2.3.4 Anchore :

Anchore is an open-source container image scanning and policy enforcement tool that focuses on providing deep analysis of container images for security vulnerabilities and policy compliance. It helps organizations ensure the integrity and security of their container images throughout the software supply chain.

2.3.5 OpenSCAP Workbench :

OpenSCAP Workbench is an open-source graphical user interface (GUI) tool that provides a user-friendly interface for the OpenSCAP framework. OpenSCAP is a standardized framework for evaluating and enforcing security compliance in IT environments, including servers, virtual machines, and containers.

The OpenSCAP Workbench simplifies the process of creating, editing, and executing security compliance scans and assessments using OpenSCAP. It offers a visual interface that allows users to define policies, scan target systems, view scan results, and generate reports.

2.3.6 Sysdig :

Sysdig is a container security platform that provides monitoring, visibility, and threat detection capabilities for containerized environments. It offers a comprehensive set of tools and features to ensure the security and compliance of container deployments.

Sysdig provides a unified platform for container security, combining monitoring, threat detection, vulnerability management, compliance monitoring, and network security capabilities. It helps organizations proactively secure their containerized applications and infrastructure, detect and respond to security incidents, and maintain compliance in dynamic container environments.

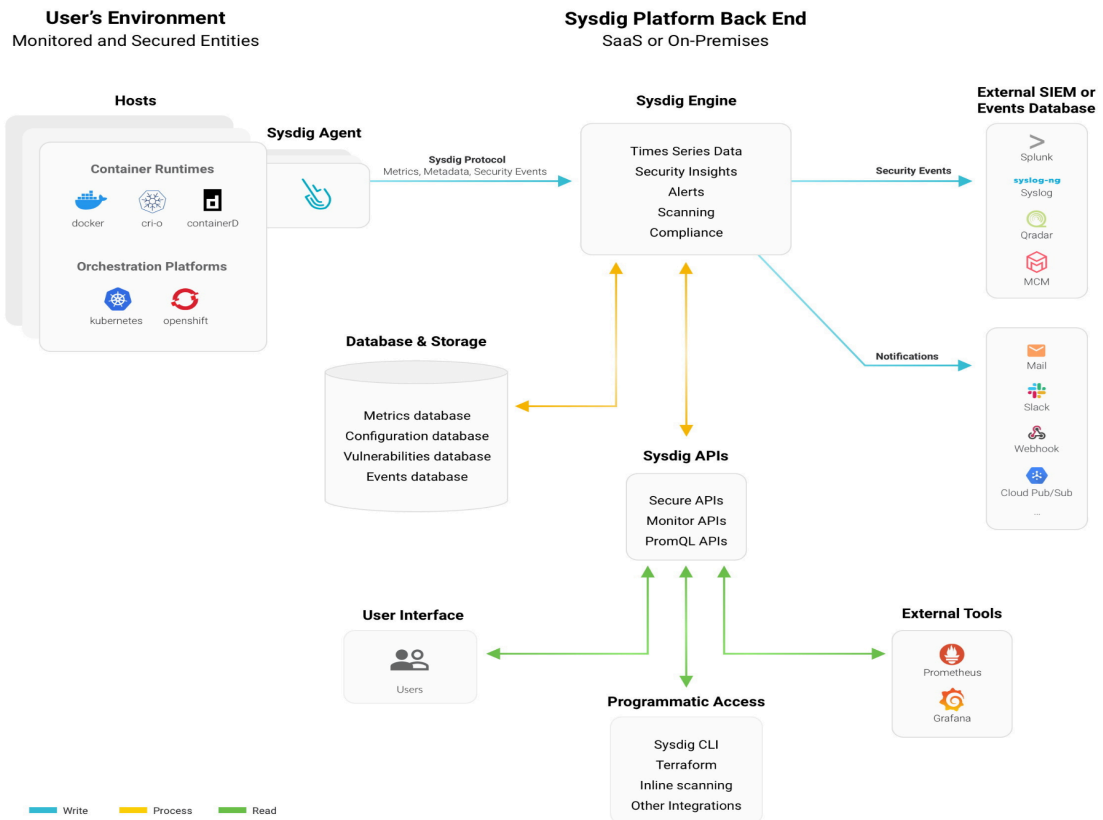


Fig 2.1 Sysdig architecture

Chapter 3

Log collection for containers

3.1 Proposed solution

To identify patterns in container security attacks, deploying a honeypot and capturing all events within the container environment is crucial. Merely relying on the basic functionality of Osquery is insufficient for a comprehensive investigation and analysis of attacks. To address this, we have enhanced Osquery by incorporating eBPF support, which allows for more detailed system data capture and improved container visibility. eBPF, a modern approach utilized by Osquery, gathers event data directly from the kernel instead of relying solely on the Linux audit subsystem. Enabling eBPF auditing features requires additional configuration for Osquery. On Linux and macOS systems, Osquery can be used to capture real-time process executions and network connections. While these auditing tools excel in capturing host activity, they may introduce higher CPU costs and an increased volume of Osquery log events. However, we can effectively manage these issues to obtain finer system information. In this setup, container logs are obtained, but a challenge arises in distinguishing between host and container events and accurately tracing events back to their respective containers. To overcome this, we employ a custom Osquery extension developed in Golang. This extension provides container-specific information, including unique container identities, creation times, and activity details. With this extension, we can differentiate between host and container events and map them accordingly.

3.2 Deploying the Container Honeypot

Our custom Honeypot is not merely a Honeypot created using containers; its main motive is to invite attacks on containers in order to provide a better understanding of the type of threats targeting containers. We deployed our honeypot on an external network and allowed threat actors looking for misconfigured, exposed containers to perform various attacks on it.

3.3 Querying container for logs

The honeypot is integrated with our tailored Golang code and a custom Osquery extension. Prompted by the Golang code, the Osquery extension queried the machine (in this

case, the Container Honeypot) for various system logs and data. The extension includes numerous Docker tables that obtained a wide range of operational data from the Container Honeypot.

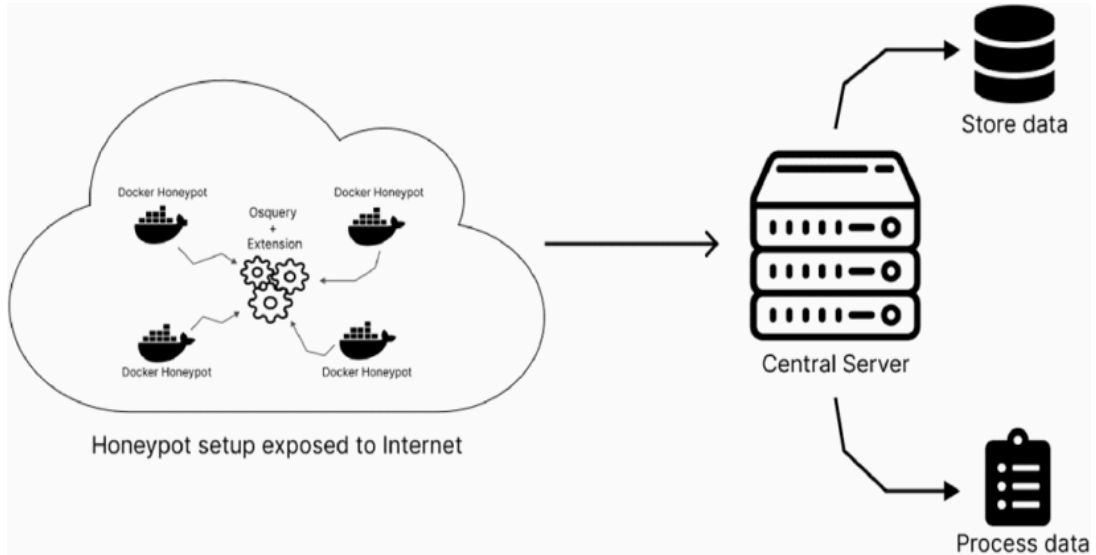


Fig 3.1 Container honeypot architecture

In the proposed architecture we run the Osquery agent on the host machine. On the same host machine we run the container or containers we want to monitor or log. In osquery, SQL tables, configuration retrieval, log handling, etc. are implemented via a robust plugin and extensions API. This project contains Go bindings for creating osquery extensions in Go. To create an extension, you must create an executable binary which instantiates an ExtensionManagerServer and registers the plugins that you would like to be added to osquery. You can then have osquery load the extension in your desired context (ie: in a long running instance of osqueryd or during an interactive query session with osqueryi).

When the Osquery core starts, it opens a Unix domain socket or TCP socket for communication with extensions. The extension needs to register with the core by specifying the socket path or address to establish the connection. We run the golang code that connects to the osquery unix socket to push data into the Osquery table.

3.4 Obtaining queried information

The custom Golang code returned the valuable data obtained by the Osquery extension and presented each data set as a column in a compact and legible log table. This data included the 'evented' activities which were buffered in the bpf_process_events and

socket_events tables. By writing our own additional Osquery tables, we were able to obtain more system information than what Osquery provides.

id	host_id	host_name	container_image	container_name	action	privileged	from_column	status	scope	time	type	path	cmd_line	pid	container_id
1	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	busy_box	die	false	{}	die	local	1679145451	container	{}	{}	0	786b2f698097e7eb2c44e5bc23a4da46a3187d3
2	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	busy_box	disconnect	false	{}	disconnect	local	1679145451	network	{}	{}	0	786b2f698097e7eb2c44e5bc23a4da46a3187d3
3	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	busy_box	destroy	false	{}	destroy	local	1679145451	container	{}	{}	0	786b2f698097e7eb2c44e5bc23a4da46a3187d3
4	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	busy_box	create	false	{}	create	local	1679145454	container	{}	{}	0	963b978e3bab43103af43c4e48af26ca381ce35
5	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	attach	false	{}	attach	local	1679145454	container	{}	{}	0	963b978e3bab43103af43c4e48af26ca381ce35
6	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	attach	false	{}	attach	local	1679145454	network	{}	{}	0	963b978e3bab43103af43c4e48af26ca381ce35
7	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	start	false	{}	start	local	1679145455	container	{}	{}	73991	963b978e3bab43103af43c4e48af26ca381ce35
8	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	resize	false	{}	resize	local	1679145455	container	{}	{}	73991	963b978e3bab43103af43c4e48af26ca381ce35
9	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	attach	false	{}	attach	local	1679145454	container	{}	{}	0	963b978e3bab43103af43c4e48af26ca381ce35
10	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	attach	false	{}	attach	local	1679145454	network	{}	{}	0	963b978e3bab43103af43c4e48af26ca381ce35
11	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	create	false	{}	create	local	1679145454	container	{}	{}	0	963b978e3bab43103af43c4e48af26ca381ce35
12	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	destroy	false	{}	destroy	local	1679145451	container	{}	{}	0	786b2f698097e7eb2c44e5bc23a4da46a3187d3
13	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	busy_box	die	false	{}	die	local	1679145451	container	{}	{}	0	786b2f698097e7eb2c44e5bc23a4da46a3187d3
14	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	busy_box	disconnect	false	{}	disconnect	local	1679145451	network	{}	{}	0	786b2f698097e7eb2c44e5bc23a4da46a3187d3
15	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	busy_box	resize	false	{}	resize	local	1679145455	container	{}	{}	73991	963b978e3bab43103af43c4e48af26ca381ce35
16	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	start	false	{}	start	local	1679145455	container	{}	{}	73991	963b978e3bab43103af43c4e48af26ca381ce35
17	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	lucid_brahmagupta	attach	false	{}	attach	local	1679145455	container	{}	{}	0	963b978e3bab43103af43c4e48af26ca381ce35
18	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	strange_ryabata	die	false	{}	die	local	1679145455	container	{}	{}	0	e481843abf86c0d1bf85222964836a0a0c5
19	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	strange_ryabata	destroy	false	{}	destroy	local	1679145455	network	{}	{}	0	e481843abf86c0d1bf85222964836a0a0c5
20	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	strange_ryabata	create	false	{}	create	local	1679145455	container	{}	{}	0	1b2b7f5d6cb5d88b022206078650873375ba2df
21	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	strange_ryabata	attach	false	{}	attach	local	1679145455	container	{}	{}	0	1b2b7f5d6cb5d88b022206078650873375ba2df
22	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	strange_ryabata	connect	false	{}	connect	local	1679145455	network	{}	{}	0	1b2b7f5d6cb5d88b022206078650873375ba2df
23	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	strange_ryabata	start	false	{}	start	local	1679145455	container	{}	{}	29552	1b2b7f5d6cb5d88b022206078650873375ba2df
24	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	ubuntu	strange_ryabata	resize	false	{}	resize	local	1679145455	container	{}	{}	29552	1b2b7f5d6cb5d88b022206078650873375ba2df

Fig 3.2 Docker events table

We can get visibility of process events and socket events happening inside containers using custom golang extension and osquery. Using the golang extension we get the information shown in fig 3.2. Docker events table gives overview of action, status, pid, containers image, containers id, name, privileged, added capabilities, dropped capabilities, port bindings and many more.

We connect docker events and bpf_process_events table from osquery to get insights of the process_events happening inside the container. This we can achieve by joining docker_events and bpf_process_events table.

honeyptdb=# select bpf_process_events.id,bpf_process_events.host_id,bpf_process_events.host_name,unixtime,tid,bpf_process_events.pid,parent,syscall,bpf_process_events.path,docker_events.type,docker_events.container_name,docker_events.action,docker_events.privileged from bpf_process_events join docker_events on cast(docker_events.pid as int)=bpf_process_events.parent limit 20;												
id	host_id	host_name	unixtime	tid	pid	parent	syscall	path	type	container_name	action	privileged
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	strange_ryabata	create	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	network	strange_ryabata	connect	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	strange_ryabata	attach	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	strange_ryabata	destroy	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	network	strange_ryabata	disconnect	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	mystifying_hoover	die	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	network	mystifying_hoover	connect	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	jolly_wilbur	die	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	jolly_wilbur	destroy	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	network	jolly_wilbur	connect	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	strange_ryabata	attach	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	strange_ryabata	create	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	network	strange_ryabata	disconnect	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	jolly_wilbur	die	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	network	jolly_wilbur	connect	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	xenodochial_turing	die	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	xenodochial_turing	destroy	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	container	jolly_wilbur	create	false
6508	20190727-Sc87-9c38-1fb2-5c879c381fb6	sablearjun	1679122613	34095	34095	0	exec	/usr/bin/dpkg	network	jolly_wilbur	connect	false

Fig 3.3 Process events inside docker container

Chapter 4

Working with Sysdig

4.1 Introduction to Sysdig

Sysdig is a silver sponsor of the Linux Foundation.. Sysdig is an open-source, container-native system monitoring and troubleshooting tool. It provides deep visibility into system-level activity in containerized and non-containerized environments. Sysdig captures, analyzes, and displays real-time and historical system data, allowing users to gain insights into system behavior, performance, and security.

Sysdig operates by leveraging eBPF (extended Berkeley Packet Filter) technology, which allows it to capture and analyze system events at a low level. It can monitor and capture data from various sources, including containers, hosts, and orchestrators. This comprehensive monitoring capability makes Sysdig well-suited for modern infrastructure environments, such as those built on containerization technologies like Docker and Kubernetes.

Sysdig enables businesses of all sizes to securely operate cloud applications by identifying, mitigating, and addressing security risks, ensuring compliance, and enhancing availability and performance. The company's foundation as an open-source organization remains integral to its identity and plays a pivotal role in driving widespread adoption among modern developers and enterprises. The platform builds upon the successful Falco and open-source sysdig projects, which have become prominent solutions for real-time detection and response in cloud and container environments. Additionally, Sysdig actively embraces other open-source initiatives such as Anchore Engine for container image scanning, Prometheus for metrics and monitoring, and Cloud Custodian for cloud configurations. These open-source technologies are integrated with proprietary workflow, orchestration, and forensics tools, creating a unified and comprehensive solution.

Here are some ways in which Sysdig provides container security:

- **Deep visibility:** Sysdig provides deep visibility into containerized environments, including container images, registry activity, network traffic, and system calls. This visibility enables security teams to quickly detect and respond to threats.
- **Runtime security:** Sysdig provides runtime security for containers by monitoring for suspicious activity, such as unauthorized file access, privilege escalation, and network

intrusions. Sysdig can also block malicious activity by enforcing policies and container isolation.

- **Compliance:** Sysdig helps ensure compliance with security and regulatory standards by providing auditing and reporting capabilities. Security teams can use Sysdig to monitor activity across containerized environments, detect vulnerabilities, and generate compliance reports.
- **Image scanning:** Sysdig provides image scanning capabilities that help identify vulnerabilities in container images. By scanning images before deployment, security teams can identify and mitigate potential risks.
- **Integration:** Sysdig can be integrated with other security tools and services, such as vulnerability scanners, SIEM systems, and orchestration platforms. This integration enables security teams to streamline their workflows and improve their overall security posture.

4.2 Using csysdig tool

"csysdig" is a command-line utility provided by Sysdig, which is used to interactively explore and monitor system activity. It is part of the Sysdig toolset and offers a convenient way to view real-time system data and perform troubleshooting tasks.

Here are some key features and capabilities of csysdig:

- **System Exploration:** csysdig provides an interactive terminal-based interface that allows users to explore system activity in real-time. It displays a comprehensive view of system calls, processes, network connections, file activity, and other system-level events.
- **Filtering and Search:** csysdig enables users to apply filters and search criteria to focus on specific system events or processes. This helps in narrowing down the displayed data and identifying relevant information.
- **Performance Analysis:** With csysdig, users can analyze system performance by monitoring CPU usage, memory utilization, disk I/O, network traffic, and other performance metrics. It provides a detailed view of resource consumption at the process and system level.
- **Troubleshooting and Debugging:** csysdig is a valuable tool for troubleshooting system issues. It allows users to inspect system calls, trace process activity, capture network

packets, and investigate file operations. This helps in diagnosing problems, identifying performance bottlenecks, and understanding system behavior.

- **Container Support:** csysdig is container-aware and can be used to monitor and analyze activity within containerized environments. It provides visibility into container resource usage, network connections, and file operations, facilitating troubleshooting and performance analysis in containerized environments.

Csysdig provides 35 options (views) to dive into the system to get detailed information about the operating system. Press the F2 key or click the Views option to enter the monitoring options list, as shown in the figure 4.1. The views and descriptions are as follows :

Sr	View	Description
1	Connections	Presents a network connections overview.
2	Containers	List of all containers running
3	Containers errors	System error counters for each containers
4	Directories	Directories accessed on the file system
5	Errors	Top system call errors
6	File opens list	List file name and process of every single file open
7	Files	Files accessed on the file system
8	I/O by type	I/O volume based on I/O type
9	K8s containers	Kubernetes controllers running on the machine
10	K8s deployments	List of Kubernetes deployments running on the machine
11	K8s namespaces	List of Kubernetes namespaces running on the machine
12	K8s pods	List of Kubernetes pods running on the machine
13	K8s replicaset	List of Kubernetes replicas running on the machine
14	K8s services	List of Kubernetes services running on the machine
15	Marathon apps	List of Marathon apps running on the machine
16	Marathon groups	List of Marathon apps running on the machine

17	Mesos frameworks	List of Mesos frameworks running on the machine
18	Mesos taska	List of Mesos tasks running on the machine
19	New connections	List newly established network connections
20	Page faults	Page faults counters for processes
21	Processes	Typical top/htop process list
22	Processes CPU	Total versus users versus CPU usage for every process
23	Processes errors	System error counters for processes
24	Processes fd usage	File descriptor usage for processes
25	Server ports	List of all server ports
26	Slow file I/O	All file read and write calls
27	Socket queues	Show queues utilization per process
28	Spectrogram-file	I/O latency spectrogram
29	Spy syslog	Entries written to syslog
30	Spy users	All commands that run interactively
31	System calls	Top system calls in the system
32	Threads	List of all threads running
33	Traces list	Detailed list of traces list executing
34	Traces spectrogram	Traces duration spectrogram
35	Traces summary	Summary of traces executing

Table 4.1 Views in csysdig

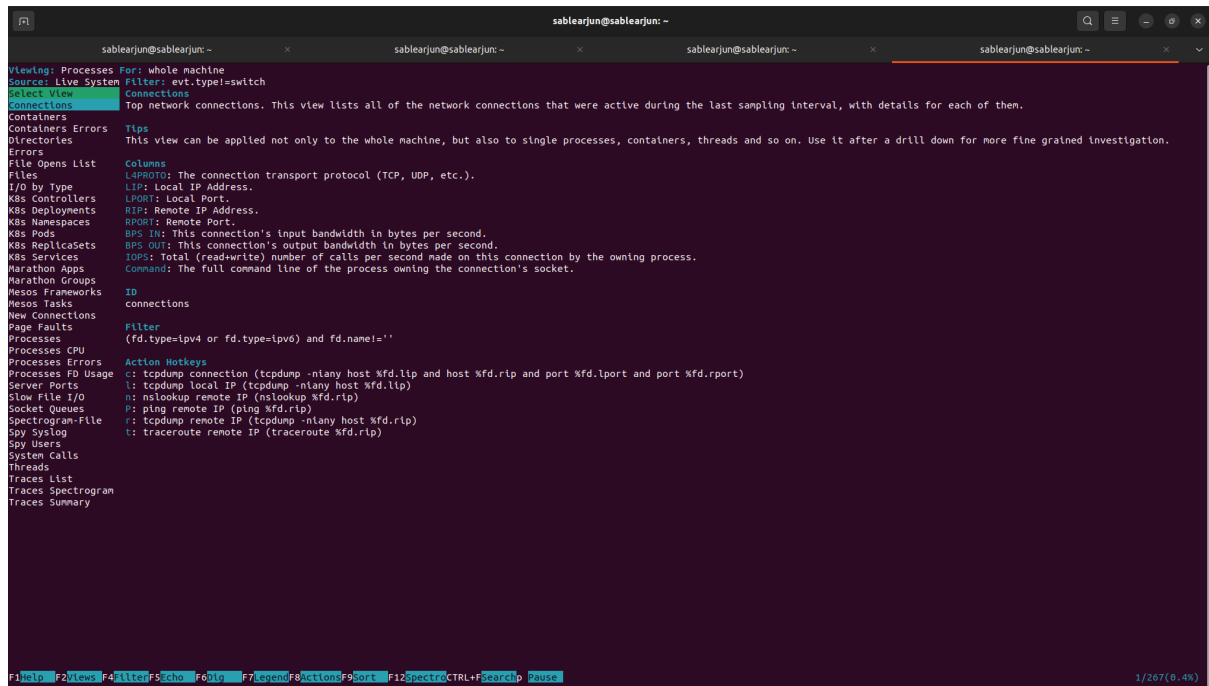


Fig 4.1 Csysdig usage

4.3 Process monitoring :

In csysdig we can see the list of processes running in the system. In the process view we can list the processes. This is a typical top/htop process list, showing usage of resources like CPU, memory, disk and network. Processes View provides an overview of active processes running on the system and their associated details. Here is a description of the Processes View in csysdig:

Sr	Column name	Description
1	Process ID (PID)	The unique identifier assigned to each running process.
2	Parent Process ID (PPID)	The Process ID of the parent process that spawned the current process.
3	Process Name	The name or executable file name of the process.
4	CPU Usage	The amount of CPU resources consumed by the process, usually represented as a percentage.

5	Memory Usage	The amount of memory utilized by the process, typically shown in kilobytes or megabytes.
6	Virtual Memory Size (VIRT)	The total virtual memory allocated for the process, including both used and unused memory.
7	Resident Set Size (RSS)	The portion of physical memory (RAM) occupied by the process.
8	Number of Threads	The count of threads spawned by the process.
9	User	The user account under which the process is running
10	State	The current state of the process, such as running, sleeping, or waiting.
11	Start Time	The timestamp indicating when the process started.
12	Priority	The priority level assigned to the process, indicating its scheduling preference.
13	File Descriptors	The number of file descriptors or open files associated with the process.

Table 4.2 Processes view

Viewing: Processes for: whole machine
Source: Live System Filter: evt.type=switch

PID	PPID	VPID	CPU	USER	TH	VIRT	RES	FILE	NET	CONTAINER	Command
332130	332130	332130	72.77	root	5	463M	156M	2G	0.00	host	/home/sablae/sun/vsrs-osquery/build/osquery/osqueryd -S --disable-events=false --enable-bpf-events=true
341508	341507	341508	15.08	root	1	2G	2G	0	0.00	host	csysd -d 100000 -pc
268749	29726	29188	5.52	sablae:run	17	3G	303M	0	53.82K	host	/opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=29713 --enable-crash-reporter=3c08e7e2
320736	281544	320736	3.23	root	20	465M	61M	1M	21.68	host	/opt/draios/bin/dragent --noipcs
29750	29721	29750	3.18	sablae:run	28	1G	142M	20	0.00	host	/opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=29713 --enable-crash-reporter=3c08e7e2
5483	4781	5483	2.19	sablae:run	28	1G	225M	0	0.00	host	/usr/bin/gnome-shell
29704	5483	29704	2.19	sablae:run	36	1G	576M	20M	13.88	host	/opt/google/chrome/chrome
5019	5016	5019	1.79	sablae:run	17	2G	59M	240K	0.00	host	/usr/lib/xorg/xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -nolisten tcp -background none
341536	29726	1	1.47	sablae:run	14	3G	143M	0	58.41K	host	/opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=29713 --enable-crash-reporter=3c08e7e2
900820	4781	900820	0.50	sablae:run	5	628M	51M	63K	0.00	host	/usr/libexec/gnome-terminal-server
320736	281544	320736	0.33	root	1	53M	4M	649K	0.00	host	/opt/draios/bin/dragent --noipcs
29751	29704	29751	0.25	sablae:run	13	488M	84M	768K	543.66	host	/opt/google/chrome/chrome --type=utility --utility-sub-type=network.nojon.NetworkService --lang=en-GB -
4781	1	4781	0.25	sablae:run	1	19M	9M	0M	0.00	host	/lib/systemd/systemd --user
341558	4781	341558	0.18	sablae:run	6	519M	28M	811K	0.00	host	/usr/libexec/tracker-extract-3
320	1	320	0.17	root	1	181M	85M	774K	0.00	host	/lib/systemd/systemd-journald
610	1	610	0.17	systemd-oom	1	16M	5M	236K	0.00	host	/lib/systemd/systemd-oomd
29809	29726	21	0.17	sablae:run	14	3G	97M	29K	412.88	host	/opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=29713 --enable-crash-reporter=3c08e7e2
2736	1	2736	0.15	root	22	2G	8M	2K	0.72	host	/opt/teamviewer/tv_bin/teamviewerd -d
29809	29726	79	0.08	sablae:run	14	3G	130M	0	0.00	host	/opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=29713 --enable-crash-reporter=3c08e7e2
4871	4781	4871	0.06	sablae:run	8	685M	32M	3M	0.00	host	/usr/libexec/tracker-miner-fs-3
469048	1	469048	0.06	root	3	255M	13M	0	4.00	host	/usr/sbin/NetworkManager --no-daemon
181261	29726	26992	0.06	sablae:run	14	3G	106M	1M	0.00	host	/opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=29713 --enable-crash-reporter=3c08e7e2
870	1	870	0.06	syslog	4	217M	5M	18K	0.00	host	/usr/sbin/rsyslogd -n -lNONE
250744	5483	250744	0.06	sablae:run	12	4G	52M	0	0.00	host	935 /usr/share/gnome-shell/extensions/ding@rastersoft.com/ding.js -E -P /usr/share/gnome-shell/extension
1153	1	1153	0.06	root	13	2G	11M	0	0.00	host	/usr/bin/containerd
5693	4781	5693	0.04	sablae:run	3	158M	5M	0	0.00	host	/usr/libexec/at-spi2-registryd --use-gnome-session
1	0	1	0.04	root	3	166M	32M	1K	0.00	host	/sbin/init splash
891	1	891	0.04	root	4	270M	8M	2K	0.00	host	/usr/sbin/thermald --system --dbus-enable --adaptive
856	1	856	0.04	messagebus	1	12M	6M	440	0.00	host	@dbus-daemon --system --address=systemd: --nofork --noptfile --systemd-activation --syslog-only
355	1	355	0.04	root	1	27M	6M	48	0.00	host	/lib/systemd/systemd-udev
320736	281544	320736	0.02	root	13	2G	59M	371K	0.00	host	colinterface --log.file /opt/draios/logs/k8s_klog.log --log.file_max_size 10 -v 0 --feature-gates K8sCom
5857	4781	5857	0.02	sablae:run	3	232M	6M	0	0.00	host	/usr/libexec/ibus-panel
314305	29726	33270	0.02	sablae:run	14	3G	318M	0	0.00	host	/opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=29713 --enable-crash-reporter=3c08e7e2
5748	4781	5748	0.02	sablae:run	4	455M	7M	0	0.00	host	/usr/libexec/gsd-sharing
267277	29726	29113	0.02	sablae:run	14	3G	89M	6K	0.00	host	/opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=29713 --enable-crash-reporter=3c08e7e2

Fig 4.2 Csysdig Processes view for host

4.4 Network monitoring

Connections View provides an overview of network connections made by processes on the system. It allows you to monitor network activity and gain insights into various aspects of the connections. Here is a description of the Connections View in csysdig:

Sr	Column name	Description
1	Local Address	The local IP address and port associated with the connection.
2	Remote Address	The remote IP address and port to which the connection is established.
3	Protocol	The network protocol used by the connection, such as TCP or UDP.
4	State	The current state of the connection, indicating whether it is established, listening, or closed.
5	Process ID (PID)	The unique identifier of the process that initiated the connection.
6	Process Name	The name or executable file name of the process associated with the connection.
7	Sent Bytes	The number of bytes sent over the connection.
8	Received Bytes	The number of bytes received over the connection.
9	Process User	The user account under which the process is running.
10	Time Opened	The timestamp indicating when the connection was established.

Table 4.3 Csysdig Connections view

Viewing: Connections For: whole machine
Source: Live System Filter: (fd.type=ipv4 or fd.type=ipv6) and fd.name!=''

Protocol	IP	IPNet	IP	IPNet	BPS OUT	BPS IN	IOPS	Container	Command
tcp	a60:837:8efa:b74	47602	8efa:b74c:a2ba:b	443	50.45	504.17	1.23	host	/opt/google/chrome/chrome --type=utility
tcp	a60:837:a77:212	40020	a77:2125:c9e1:a60	22	40.01	45.23	1.22	host	ssh osquery@10.119.2.10
tcp	a60:837:8efa:c02	60182	8efa:c024:16eb:b	443	44.49	5.91	0.21	host	/opt/google/chrome/chrome --type=utility
tcp	a60:837:8efa:d20	42956	8efa:d20e:cca7:b	443	10.76	320.51	0.46	host	/opt/google/chrome/chrome --type=utility
udp	ac11:1:e900:fb:e	5353	e000:fb:e914:e91	5353	5.64	0.00	0.16	host	/opt/google/chrome/chrome
udp	7f00:1:e000:fb:e	5353	e000:fb:e914:e91	5353	5.64	0.00	0.12	host	/opt/google/chrome/chrome
tcp	a60:837:8efa:c78	46452	8efa:c783:74b5:b	443	5.22	24.01	0.07	host	/opt/google/chrome/chrome --type=utility
tcp	a60:837:a77:212	35592	a77:212:88b:1600	22	4.00	2.76	0.28	host	ssh osquery@10.119.2.10
udp	a60:837:ac8:10b	35354	ac8:10b:1a8a:350	53	3.02	0.47	0.10	host	/lib/systemd/systemd-resolved
tcp	a60:837:23ae:7f1	45324	23ae:7f1f:cb1:bb	443	2.69	2.48	0.07	host	/opt/google/chrome/chrome --type=utility
tcp	a60:837:227a:792	60922	227a:7920:faed:5	80	2.69	0.88	0.08	host	/usr/sbin/NetworkManager --no-daemon
udp	7f00:1:f000:35:c	59240	7f00:35:0be7:350	53	2.06	1.16	0.20	host	/opt/draios/bin/dragent --noipcs
tcp	a60:837:8efa:c00	44224	8efa:c00e:c0ac:b	443	1.74	7.88	0.31	host	/opt/google/chrome/chrome --type=utility
tcp	a60:837:2275:c9a	42226	2275:c9aa:f2a4:b	443	1.76	1.95	0.27	host	/opt/google/chrome/chrome --type=utility
tcp	a60:837:9df0:ed3	41322	9df0:ed3c:0aa1:b	443	1.44	1.40	0.16	host	/opt/google/chrome/chrome --type=utility
udp	ac11:1:e914:e914	5353	:ac11:1:e914:e9	5353	1.41	0.00	0.04	host	avahi-daemon: running [sablearn.local]
udp	7f00:1:e914:e914	5353	:7f00:1:e914:e9	5353	1.41	0.00	0.01	host	avahi-daemon: running [sablearn.local]
udp	a60:837:ac8:10b	55751	ac8:10b:c7d9:350	53	0.78	0.39	0.10	host	/lib/systemd/systemd-resolved
udp	7f00:1:f000:35:c	50705	7f00:35:81dd:350	53	0.56	0.00	0.03	host	/lib/systemd/systemd-resolved
tcp	a60:837:d992:b09	40312	d992:b09:esb4:bb	443	0.24	0.45	0.08	host	/opt/teamsviewer/tv_bin/teamsviewer -d
tcp	a60:837:2291:7bf	41878	2291:7bfd:96a3:2	6443	0.00	0.00	0.06	host	/opt/draios/bin/dragent --noipcs
udp	a60:837:e000:fb:	5353	e000:fb:e914:e91	5353	0.00	2.52	0.06	host	avahi-daemon: running [sablearn.local]
tcp	a60:837:2291:7bf	48448	2291:7bfd:40bd:2	6443	0.00	0.00	0.01	host	/opt/draios/bin/dragent --noipcs

Fig 4.3 Csysdig Connections view for host

4.5 Container monitoring

Containers View provides an overview of containers running on the host. Here is a description of the Containers View in csysdig:

Sr	Column name	Description
1	CPU	Amount of CPU used.
2	Procs	Number of processes currently running
3	Threads	Number of threads running
4	Virt	Total virtual memory for process
5	Res	Resident no-swap memory for the processes.
6	File	Total file I/O bandwidth generated
7	Net	Total network bandwidth generated
8	Engine	Container type
9	Image	Container image name
10	ID	Container ID
11	Name	Name of the container.

Table 4.4 Csysdig Containers view

Viewing: Containers For: whole machine
Source: Live System Filter: container.name != host

CPU	PROCS	THREADS	VIRT	RES	FILE	NET	ENGINE	IMAGE	ID	NAME
0.00	1.00	1.00	5M	2M	0	0.00	docker	ubuntu	05d232571687	mystifying wing

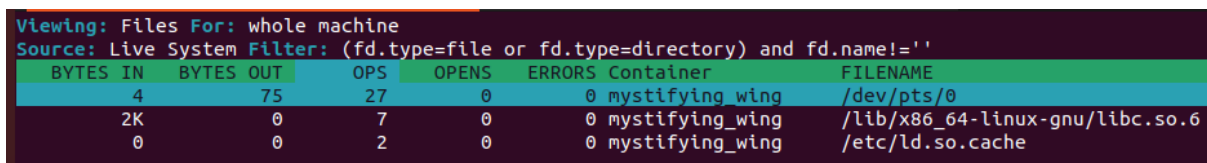
Fig 4.4 Csysdig Containers view for host

4.6 File accesses

Files View provides insights into file-related activities on the system. The Files View allows you to monitor and analyze various file operations performed by processes. Here is a description of the Files View in csysdig:

Sr	Column name	Description
1	Bytes in	Amount of bytes read from file
2	Bytes out	Amount of bytes write to file
3	Ops	Number of I/O operations done to the file.
4	Opens	Number of times file opened.
5	Errors	Number of I/O errors that happened
6	Filename	Filename including its full path

Table 4.5 Csysdig Files view



Viewing: Files For: whole machine						
Source: Live System Filter: (fd.type=file or fd.type=directory) and fd.name!=""						
BYTES IN	BYTES OUT	OPS	OPENS	ERRORS	Container	FILENAME
4	75	27	0	0	mystifying_wing	/dev/pts/0
2K	0	7	0	0	mystifying_wing	/lib/x86_64-linux-gnu/libc.so.6
0	0	2	0	0	mystifying_wing	/etc/ld.so.cache

Fig 4.5 Csysdig Files view for host

Chapter 5

Conclusion

We demonstrated a honeypot to monitor container attacks. The honeypot is built on open-source tools Osquery and Docker API to gather ‘evented’ data logs of the container activities. Our honeypot setup will be very useful for collecting granular data and can be used for training AI/ML algorithms. These trained models can be used to predict threats in production environments and take necessary remediation actions.

We also tried comparing logs between Osquery and an Open source tool Sysdig.

In conclusion, both Osquery and Sysdig are powerful tools used for different aspects of container security and monitoring.

Osquery, with its SQL-like interface, focuses on system visibility and querying capabilities. It allows administrators to gather detailed information about the system, monitor processes, files, and other system-level activities. Osquery's extensibility through custom extensions, including eBPF integration, enhances its container visibility and enables better threat analysis. It is a versatile tool that provides insights into system state and behavior.

On the other hand, Sysdig specializes in container and system-level monitoring, detection, and response. It offers comprehensive visibility into containerized environments, including network connections, file activities, and system calls. Sysdig's strength lies in its ability to capture and analyze real-time system activity, providing a holistic view of containerized applications. It also integrates with other open-source projects like Falco, Anchore Engine, Prometheus, and Cloud Custodian, further enhancing its capabilities.

Bibliography

1. Osquery [online]
<https://osquery.io/>
2. Docker Engine API [online]
<https://docs.docker.com/engine/api/v1.42/>
3. Osquery Go client [online]
<https://pkg.go.dev/github.com/docker/docker/client>
4. Go lang, [online]
<https://go.dev/>
5. Sysdig monitor [online]
<https://docs.sysdig.com/en/docs/installation/sysdig-monitor/>
6. The Relevance of Container Monitoring Towards Container Intelligence [online]
<https://ieeexplore.ieee.org/document/8493766>
7. Securing Containers: Honeypots for Analysing Container Attacks
<https://ieeexplore.ieee.org/document/10041276>