D4.2 Data Inspection Component v1

WP4 – Sphinx Toolkits

Version: 1.00

SPHINX

A Universal Cyber Security Toolkit for Health-Care Industry

Disclaimer

Copyright message

Document information

Grant Agreement Number	826183	Acronym	SPHINX		
Full Title	A Universal Cyber Security Toolkit for Health-Care Industry				
Topic	SU-TDS-02-2018 Toolkit for assessing and reducing cyber risks in hospitals and care centres to protect privacy/data/infrastructures				
Funding scheme	RIA - Research and Innovation action				
Start Date	1st January 2019	**Duration**	36 months		
Project URL	http://sphinx-project.eu/				
EU Project Officer	Reza RAZAVI (CNECT/H/03)				
Project Coordinator	National Technical University of Athens - NTUA				
Deliverable	D4.2 Data Inspection Component v1				
Work Package	WP4 – Sphinx Toolkits				
Date of Delivery	**Contractual**	M20	**Actual**	M20	
Nature	R - Report	**Dissemination Level**	P - Public		
Lead Beneficiary	PDMFC				
Responsible Author	Stylianos Karagiannis	**Email**	stylianos.karagiannis@pdmfc.com		
		Phone			
Reviewer(s):	TECNALIA, TEC				
Keywords	Sandboxing, security testing, untrusted components				

Document History

Version	Issue Date	Stage	Changes	Contributor
0.10	21/05/2020	Draft	ToC	Stylianos Karagiannis (PDMFC)
0.20	21/06/2020	Draft	Initial Steps	Stylianos Karagiannis (PDMFC)
0.30	21/07/2020	Draft	Discussion of ToC and content	Stylianos Karagiannis (PDMFC), Luis Laneiro (PDMFC)
0.40	25/08/2020	Draft	Main content added	Stylianos Karagiannis (PDMFC)
0.50	26/08/2020	Draft	Internal Review 1	Santiago de Diego (TECNALIA),
0.60	26/08/2020	Draft	Internal Review 2	Waqar Asif (TEC)
0.70	28/08/2020	Pre – Final	Applied corrections	Stylianos Karagiannis (PDMFC)
0.80	28/08/2020	Pre - Final	Quality Control	George Doukas (NTUA) , Michael Kontoulis (NTUA)
1.00	28/08/2020	Final	Final	Christos Ntanos (NTUA)

Executive Summary

The SPHINX Data Inspection Component includes the Sandbox (SB) and enables a solution for creating a safe and isolated environment for security testing and continuous component validation. Using existing technologies such as containerization and virtualization, this component aspires to provide the important infrastructure and deployment services which will be executed in an isolated and safe environment. These technologies include for example: the exploitation of Docker containers and Kernel Virtual Machine (KVM), among others.

This document presents the detailed design for the SPHINX SB component, following the component's introduction in the SPHINX architecture deliverable (D2.6 - SPHINX Architecture v2). It extends the details providing information related to the virtualization and deployment of third-party components in an isolated environment for conducting the data inspection.

The next iteration of this deliverable D4.2: Data Inspection Component *(R&DEM, PU&CO, M20 & M32)*, will incorporate refinements and updates of the SB component, integration efforts and case examples for demonstrating the process of the component.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Table of Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Table of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Table of Abbreviations

OS	Operating System
API	Application Programming Interface
VM	Virtual Machine
KVM	Kernel Virtual Machine
SDLC	Software Development Lifecycle
SB	Sandbox
DOC	Document
EXE	Executable
PDF	Portable Document Format
SCSI	Small Computer System Interface
IDS	Intrusion Detection Systems
RAM	Random Access Memory
CPU	Control Process Unit
SSH	Secure Shell
CLI	Command Line Interface
SA	Situational Awareness
ACC	Automated Cybersecurity Certification

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

1 Introduction

1.1 Purpose & Scope

This document reports on the data inspection component and the sandbox developments for having a service which handles the deployment of multiple services and external components in a sandboxed mode for conducting the data inspection. This approach is important for deploying easy and flexible external services in order to continue with the security testing or data inspection. Some of the technologies used by this solution include Docker containers, virtual machines and micro-VMs. The purpose of this component is to provide an efficient, flexible and low-overhead solution for executing the digital environment. Nowadays, the security aspects are mostly focused on the cloud perspective, meaning that the cybersecurity aspects advance to the topics of network security, network services and cloud infrastructure in general. Therefore, we deploy and analyze most of the modern components in terms of micro-services.

1.2 Structure of the deliverable

This document is structured as follows. Section 1 and its subsections present the purpose and scope of the SPHINX Data Inspection Component, as well as its relation to other tasks. In Section 2, it is introduced an overview of the SPHINX Data Inspection Component, emphasizing on design principles relevant to the aspects of virtualization, hypervisor technologies, integration and network isolation. In Section 3, virtualization technologies are further analyzed and described. In Section 4, the integration capabilities are investigated. In Section 5, the aspects regarding the network infrastructure, topology are addressed. Finally, Section 6 concludes this document, presenting the outcomes of this component's developments and future steps.

1.3 Relation to other WPs & Tasks

This document is tightly related to the tasks that partake in the deployment of the required services for extracting descriptful data from the sandbox or for deploying SPHINX components inside the sandbox for extending further the research impact and demonstrate or extend further their capabilities. Within the scope of the SPHINX project the tasks which relate to this task are T3.5 – D3.5: SPHINX Automated Cybersecurity Certification, T3.3 - D3.3: Vulnerability Assessment as a Service, T4.4 – D4.4 :SPHINX AI Honeypot integration to T4.5 – D4.5: SPHINX Embedded SIEM, T5.3 – D5.3: Security Incident/Attack Simulator. The component of sandbox and data inspection component was introduced to the SPHINX architecture (T2.3 - D2.3: Use Cases definition and requirements document).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

2 Overview of Data Inspection Component

2.1 Scope of Data Inspection Component

The main goal for the SPHINX sandbox and of the data inspection component is to provide a shared sandbox environment for conducting security testing. Therefore, a digital environment would be provided for separating running programs and services which might include vulnerabilities. Finally, the sandbox provides a restricted and tightly controlled set of resources for guest programs or services to run.

2.2 Design Principles

Taking into consideration the design and software development lifecycle (SDLC) principles narrated in deliverable D6.1 the sandbox is being developed having in mind the research scope of the project to identify the applicability and the deployment options for the sandbox.

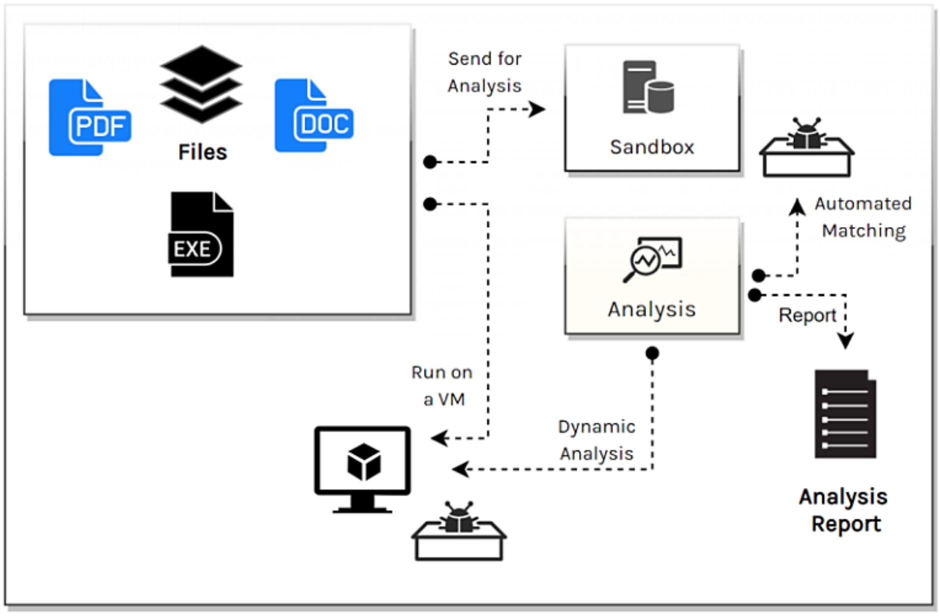


Figure 1. Existing sandbox approaches

In Figure 1, an example of the existing approaches is presented. Most of these approaches reply on submission of specific files or software, which are then exploited using software, which as a result provide evidence as to if the tested component is secure and safe to work on. Therefore, not only the signatures from the submitted files are matched to the existing taxonomies (e.g. VirusTotal[1]) but it includes various processing modules for extracting reports [1]. Consequently, such approaches are a combination of sandboxing and security testing for discovering for example zero-day attacks and suspicious software components or digital assets that might hide malicious payload. Some popular examples include Cuckoo Sandbox[2], Sandboxie[3] or commercial solutions such as FortiSandbox[4] [2, 3, 4].

According to deliverable D2.6 - SPHINX Architecture v2 (WP2 – Conceptualization, Use Cases and System Architecture), the sandbox and data inspection component was described the main component for conducting

[1] https://www.virustotal.com

[2] https://cuckoosandbox.org/

[3] https://www.sandboxie.com/

[4] https://www.fortinet.com/products/sandbox

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

security tests and to host the cybersecurity certification. There are Nine (9) basic functional requirements SB-F-010 to SB-F-060, SB-F-090, SB-F-120 and SB-F-140 (D2.6 - SPHINX Architecture v2). The table below illustrates the functional requirements identified for the Sandbox by the stakeholders (**Table 1**).

Technical Specification ID	Stakeholder Requirement ID	Observations
SB-F-010	***STA-F-160*** ***STA-F-570***	*Verification toolkit easy to integrate* *Isolated sandboxed environment*
SB-F-020	***STA-F-150*** ***STA-F-180***	*Automated zero touch device and service verification* *Automated certification (including API)*
SB-F-030	***STA-F-570***	*Isolated sandboxed environment (replication of IT infrastructure for tests)*
SB-F-040	***STA-F-160*** ***STA-F-570***	*Verification toolkit easy to integrate* *Isolated sandboxed environment*
SB-F-050	***STA-F-200***	*Monitoring network traffic and suspicious network packets*
SB-F-090	***STA-F-220***	*Data analysis and visualisation*
SB-F-120	***STA-F-200***	*Monitoring discovered unsupervised processes*

Table 1 Functional requirement traceability (SPHINX Project. D2.6 - SPHINX Architecture v2)

Malware analysis usually includes an API to upload potential malicious files that are sent for malware analysis to a sandbox which initiates a virtual machine to exexute or open the file. After the execution of the file from the sandbox, screenshots are generated accordingly, and the system shuts down in case of a malware infection. While the procedure is dynamic, the results and reports are static, only focusing on the potential infected file. Therefore, such approaches do not include vulnerability assessments in cases where a vulnerable service is deployed that might not be malicious, even so the deployed service could intentionally open specific vulnerabilities in the system (e.g. deploying an outdated apache server). Our intention is to further broaden the potential of dynamic analysis, using sandboxing to conduct security and auditing tests, including procedures such as file integrity monitoring, vulnerability detection, regulatory compliance, among others.

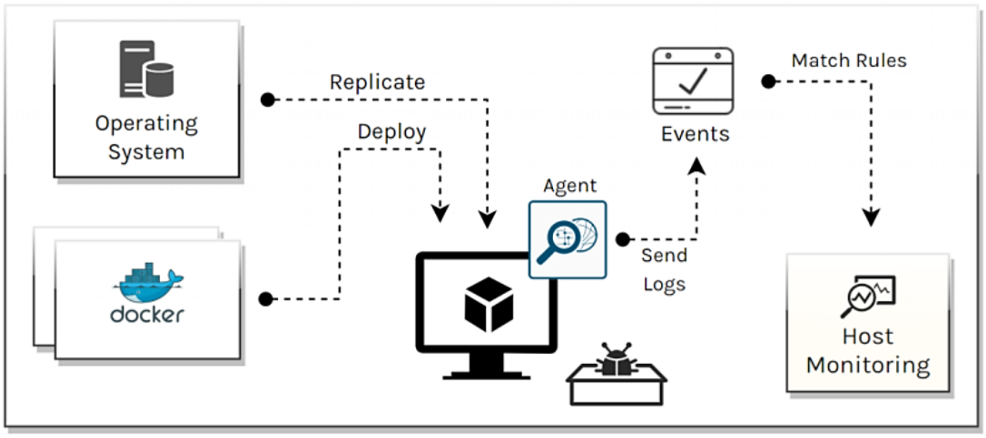
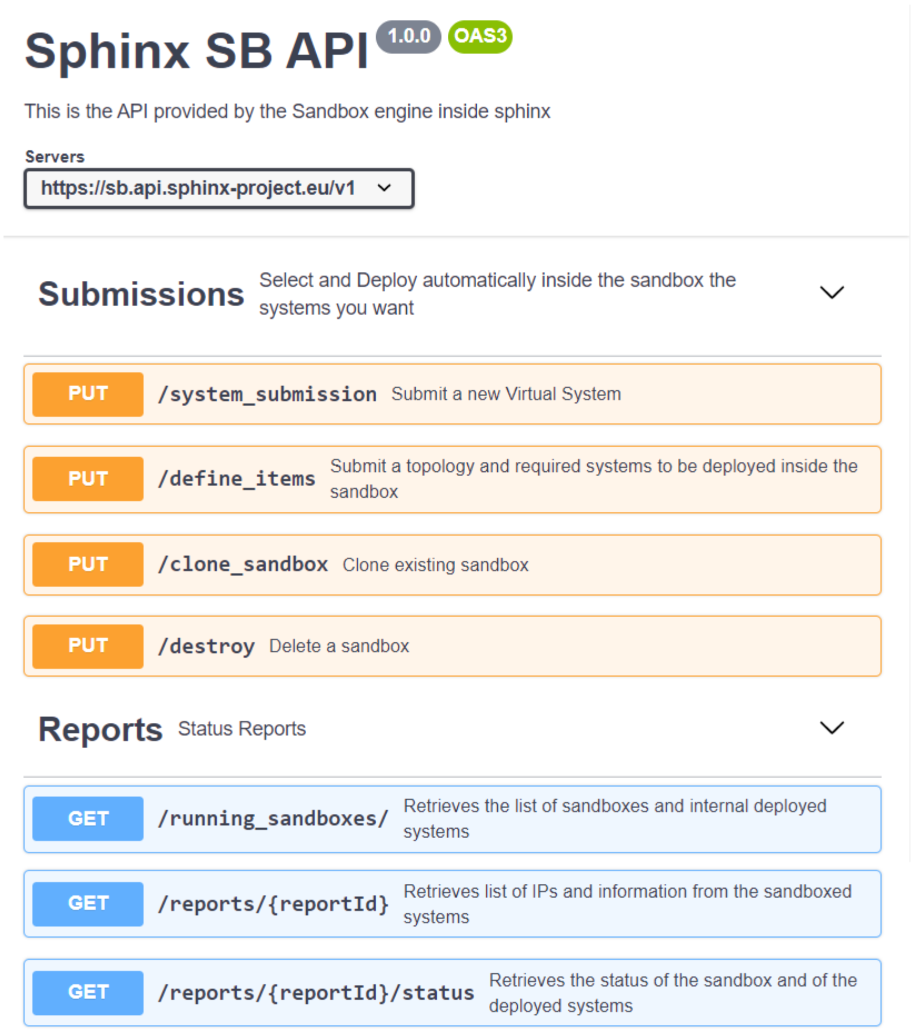


Figure 2. Dynamic and continuous system auditing using sandboxing

As presented in Figure 2, the goal for us is to deploy systems and services in a way we can monitor and conduct security tests. For enabling such aspects we use virtualization technologies which are more secure than containerization technologies such as Docker containers. The main concern that enforced us to include strictly virtualization technologies include the strong isolation capabilities that virtualization provides in comparison to containerization.

2.3 Swagger Specification

The API endpoints include the submission of system(s) providing a qcow2 or virtual machine image and of Docker images to be deployed inside the sandbox. The items could be either defined or chosen using a list of assets to be deployed. Further actions include the cloning of a sandbox or the deletion of a sandbox (Figure 3).


Sphinx SB API 1.0.0 OAS3

This is the API provided by the Sandbox engine inside sphinx

Servers
https://sb.api.sphinx-project.eu/v1

Submissions Select and Deploy automatically inside the sandbox the systems you want

PUT /system_submission Submit a new Virtual System

PUT /define_items Submit a topology and required systems to be deployed inside the sandbox

PUT /clone_sandbox Clone existing sandbox

PUT /destroy Delete a sandbox

Reports Status Reports

GET /running_sandboxes/ Retrieves the list of sandboxes and internal deployed systems

GET /reports/{reportId} Retrieves list of IPs and information from the sandboxed systems

GET /reports/{reportId}/status Retrieves the status of the sandbox and of the deployed systems

Figure 3. Swagger API for Sandbox and Data Inspection Component

The Swagger API might be extended as other processes might be seem important for the SPHINX. Furthermore, we intend to upgrade our approach and provide automation options for the end user to deploy easier sandboxes and to interact better with the deployed systems.

2.4 Untrusted Sources

The developed sandbox is an important asset/toolbox for actually executing unknown and untrusted sources to understand their behaviour and highlight any potential impacts in security. Using the sandbox it is possible to extract information regarding the total behaviour of such components and to execute test cases to understand the total impact and potential issues that such components might trigger. For example, it is common to download, install and execute various applications or to open/execute files which could include malicious or unwanted payload (e.g. ransomware, trojans). More importantly specifically deployed services might not contain malicious payloads; however, such services could open applications which include vulnerabilities or could malfunction or negatively affect other network components/systems.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

2.5 Security Tests

For defining the above issues, it is important to include security tests when a sandbox is deployed. Not only strictly security tests are important, but to understand the total behavior of the component as well. This means that it is important to collect information regarding the network behavior, registry changes, filesystem changes and authorization processes, among others. By monitoring such information, we are able to retrieve data from the sandbox and upgrade the sandboxing process to not only deploy components in an isolated environment but to retrieve insightful data as well.

Furthermore, security testing is an interactive process meaning that in order to retrieve and collect auditing results and discover security flaws it is important to deploy a realistic environment. For example, security testing nowadays is strongly matched with red team assessments and to understand the actions of blue teams that use defensive techniques to mitigate threats and to execute incident response actively. Finally, blue teams could test their tools and rulesets in a realistic environment, using the sandbox to train their models further or to include existing adversaries and software components to extend further the existing approaches.

2.6 Background

The background of the work described in this deliverable report to deliverable D3.2 where it is addressed the Situational Awareness (SA) in the healthcare cybersecurity domain. The SA is based on three main sequential phases: “Perception,” “Comprehension,” and “Projection.” The Projection phase is the last one and, during this phase, the system and its interventions must demonstrate the capabilities. However, in our case the sandbox could include the whole lifecycle of the sequential phases, meaning a recursive process that includes perception and comprehension in order to improve further. In a healthcare environment, it is during the Perception phase that the elements of an IT department collects the information from all the electronic equipment connected to the network; however it is necessary to test the tools and the electronic equipment in a realistic environments, provided by the sandbox. During the Comprehension phase, it is important to understand the potential weaknesses of the network equipment regarding cybersecurity aspects. It is at this stage that a cybersecurity toolkit can play an important role, helping to identify potential cybersecurity gaps.

2.7 Data Inspection Model in Sphinx

Most of the current approaches for sandbox are focused in malware analysis providing a way to analyze files automatically and to provide the interactions between the files that are under analysis and the system [5, 6]. An important aspect is that sandboxing must be realistic in order for the malwares to have their normal behavior. Towards this direction, current approaches have been developed to extract and monitor suspicious and malicious files (e.g. Cuckoo Sandbox). However, there are cases when a specific software component or service do not include a direct malfunction or security issue to the system. Therefore, the interaction between systems, components or services must be carefully monitored and tested. The data inspection model in SPHINX provides this opportunity, meaning the deployment of services or software components in a way to interact each other and to extract data regarding their behavior accordingly.

2.7.1 Virtualization Technologies

Virtualization technologies are important to use for deploying a sandbox. Docker containers are also frequently used as a similar technology for virtualization; however, we must declare the key-differences between them. By design, containerization technologies such as that of Docker are designed to execute micro-services and not operating systems. Not only this, but the services running on a Docker container are kernelless meaning that

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

all of them share the same Operating System kernel. Virtualization technologies provide numerous capabilities including the following [8]:

1. **Server consolidation:** To distribute the workloads from complex systems to multiple micro-services or VMs in order to consolidate the workload effort and to manage better in terms of administration and scale better in terms of performance and system resources.
2. **Application consolidation:** Meet the application's requirements in terms of hardware or software by virtualizing the hardware or by meeting any of the software requirements independently.
3. **Sandboxing**: Provide secure and isolated environments for executing unknown sources and conducting security tests and malware analysis.
4. **Multiple execution environments:** Create multiple execution environments, increasing the scope for including quality tests.
5. **Virtual hardware:** Virtualize hardware resources such as SCSI drives and network interfaces, among others.
6. **Multiple simultaneously OS:** Execute multiple operating systems that interact with each other.
7. **Debugging:** Execute software in their full potential by letting the user interact with the software.
8. **Software migration and Appliances:** Provides flexibility, compatibility and enhance portability providing the capabilities to package a whole digital environment into an appliance.
9. **Test scenarios:** Helps produce test scenarios that are hard to reproduce in reality and therefore enhances the capabilities for conducting security test scenarios.

Popular virtualization and hypervisor technologies include Oracle Virtualbox, VMware, KVM, Hyper-V, Xen and OpenVZ among others. Micro Virtual Machines (Micro VMs) are also a modern approach and popular approaches include AWS (Amazon Web Services) Firecracker (using KVM or Ignite Firecracker) and RancherVM. Other popular approaches for maintaining virtualization technologies is Vagrant for building and managing virtual machine environments included in a single workflow. Each of the mentioned technologies include benefits and drawbacks and the analysis and outcomes are also provided in this deliverable.

2.7.2 Integration Capabilities

Integration capabilities are important to include the sandbox component to easily interact and integrate with other software components. Not only the automated deployment, but the integration with the other SPHINX components is important. Towards this direction, we implement the APIs and the automated processes for initiating systems-on-a-test and to provide an easy way for the end user to interact with the sandbox and to collect insights for running an executable digital environment. The integration is processed using WEB APIs and Web interface for controlling the components inside the sandbox.

2.7.3 Networking, subnetworks and System Isolation

Nowadays, it is important to consider the network communication as one of the most important aspects in the modern digital infrastructures and systems. This means that every component currently includes network connection and continuously interacts with other network components. In the developed sandbox there are 3 different aspects of networking. The first one is the physical network interfaces/adapters, secondly are the virtualized network adapters created by the hypervisor and third are virtualized network interfaces created for the Docker containers. As a result, it is possible to combine or revise any of the above options to meet our own goals, accordingly.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

3 Virtualization Technologies

Virtualization technologies include a lot of different implementations and existing deployments. We purpose to identify the key-benefits from such technologies and to include them in our development. The benefits of each one are described and the differences from using Docker containers are addressed. Therefore, the capabilities and the drawbacks or restrictions for each of the existing approaches have been tested and analyzed.

3.1 Overview

There are two types of virtualization methods and techniques including native or hosted virtualization. These two types are called native-bear metal and hosted approach (Type 1 and Type 2) [7, 8]. Using a type 1 hypervisor means that the operating system and the services which are executed are directly relevant to the hypervisor technology avoiding any extra overhead (Figure 4). Managing a type 1 hypervisor includes performance benefits; however, extra services are not included.

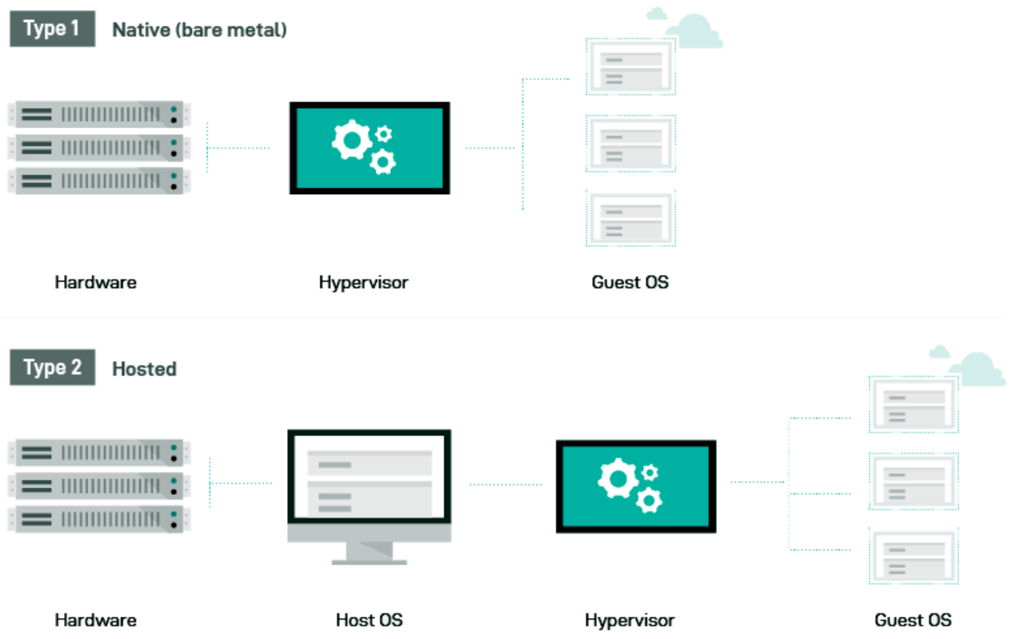


Figure 4. Virtualization - Type 1 and Type 2 hypervisors

Except these two types another relevant service and deployment approach is the containerization. Using containers, it is possible to run services or even operating systems; however, this approach is not strictly related to virtualization technologies, since the same operating system's kernel is used. The difference between the technologies and deployment approaches are presented in Figure 5. All the mentioned approaches include benefits and drawbacks. For example, containerization might include security risks which have to be managed, while virtualization include higher overhead and higher demands related to the system resources.

Type 1 virtualization approach (bare metal) is very good option when hosting multiple virtual systems and do not have any other services running, meaning that the resources are entirely dedicated to the virtual systems. Providing isolation and high system resources, type 1 is the recommended solution. Advantages of bare metal hypervisors are the following:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

- Resources dedicated to a single customer
- Greater processing power and input/output operations per second (IOPS)
- More consistent disk and network I/O performance
- Quality of Service (QoS) that guarantees elimination of the noisy neighbour problem in a multitenant environment.

Type 2 and hosted hypervisors might be similar to the bare metal hypervisors, but they could maintain and manage more services as well.

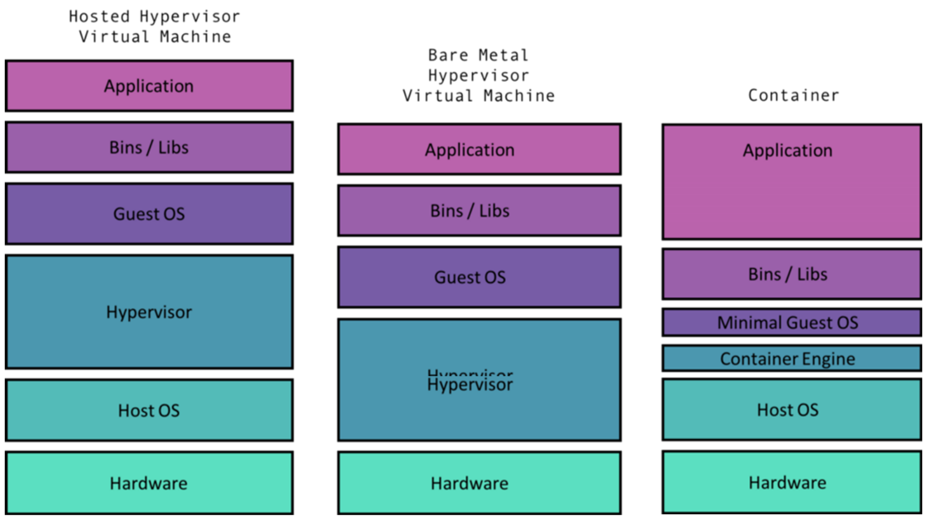


Figure 5. Differences between virtualization approaches and containers

Therefore, in the case of type-2 hypervisors, the system resources might be shared with the managed services. Finally, containerization technologies maintain a lot of benefits; however, there are multiple security risks due to the lack of a hypervisor and all of the deployed containers are executed in the same kernel, sharing the same filesystem and services of the hosting system. In that perspective an analysis of the benefits and drawbacks is important and it is provide in the section below.

3.2 Virtualization vs Containerization

As presented in the above section virtualization and containerization maintain some major differences. Containerization is a virtualization. One of the main aspects that require analysis when using containerization technologies from the security perspective (e.g. using Docker) includes properties and capabilities that containers do not include (process, filesystem, device, network isolation and the incapability for limiting the resources) [1]. Although mitigation actions exist (e.g. chroot jail creating an isolated directory for running processes) for enhancing the security posture of containers the attack surface is always bigger than virtual machines.

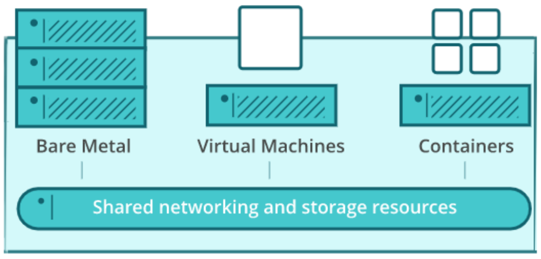


Figure 6. Virtualization approaches and containers

In some cases, the sandbox could include both the virtualization and containerization technologies. Despite the limitations, containers have been deployed in a variety of use cases. They are popular for hyperscale deployments, lightweight sandboxing, and, despite concerns about their security, as process isolation environments. Significant benefits from using containers include the following:

- Run stand-alone services and applications consistently across multiple environments
- Create isolated instances to run tests
- Build and test complex applications and architectures on a local host
- Provide lightweight stand-alone sandbox environment for developing, testing, and teaching technologies
- High performance

Regarding containerization, performance research has been done which evaluates the response times from HTTP requests when using containerization in comparison to virtualization [2]. The differences between the deployment of virtual machines compared to containers are presented in Figure 7.

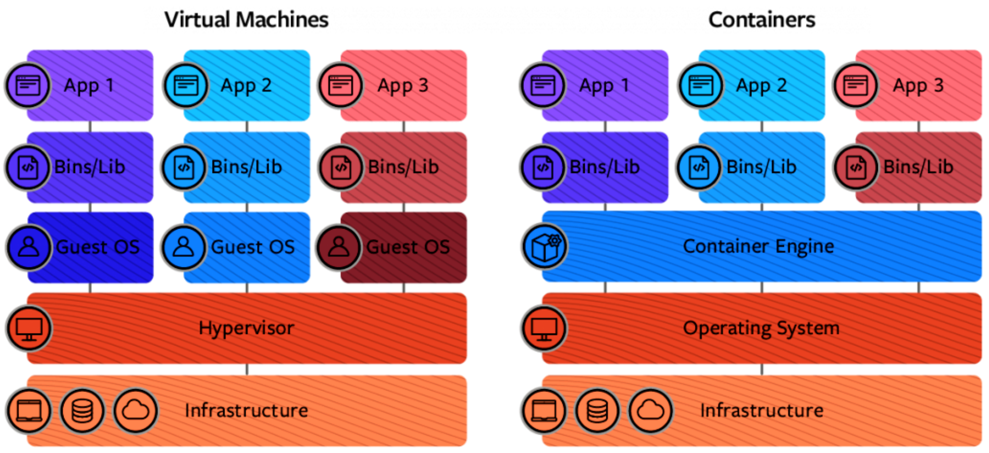
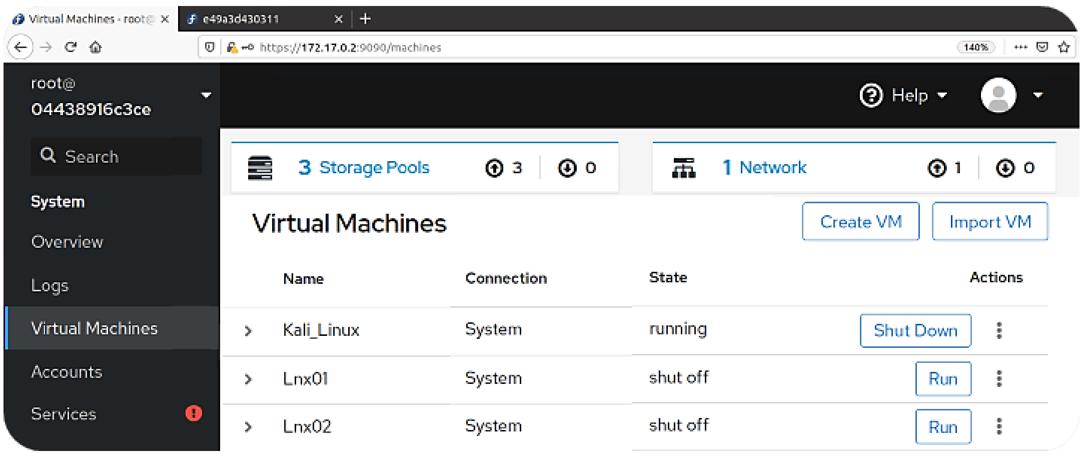


Figure 7. Difference between Virtual Machines and Containers on the usage of system resources

In our demo deployment we execute a sandbox using a Docker container which initiates multiple virtual machines which operate in the same subnet. As presented in *Figure. 8* the container is using the IP address 172.17.0.2 and the container includes virtual images (e.g. Linux, Windows 10). It is possible to easily destroy and create another sandbox (e.g. 172.17.0.3) which will initiate the virtual machines again in a different subnet and inside another Docker container.



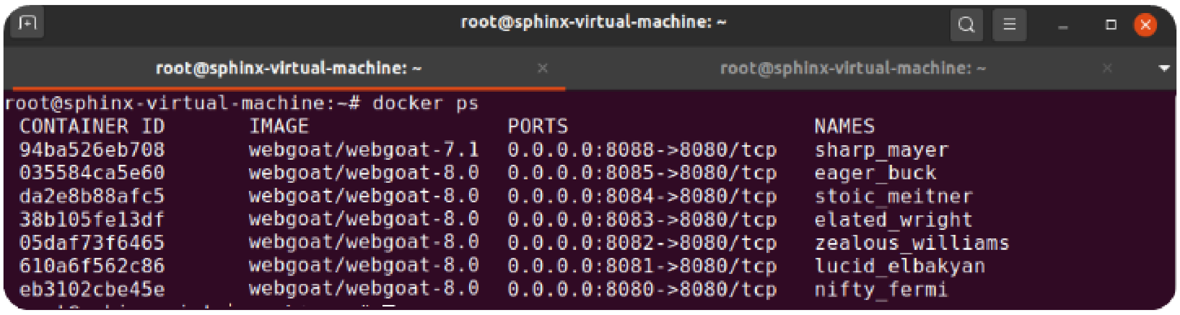
Virtual Machines

Name	Connection	State	Actions
Kali_Linux	System	running	Shut Down
Lnx01	System	shut off	Run
Lnx02	System	shut off	Run

Figure. 8. KVM running in two different Docker containers

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

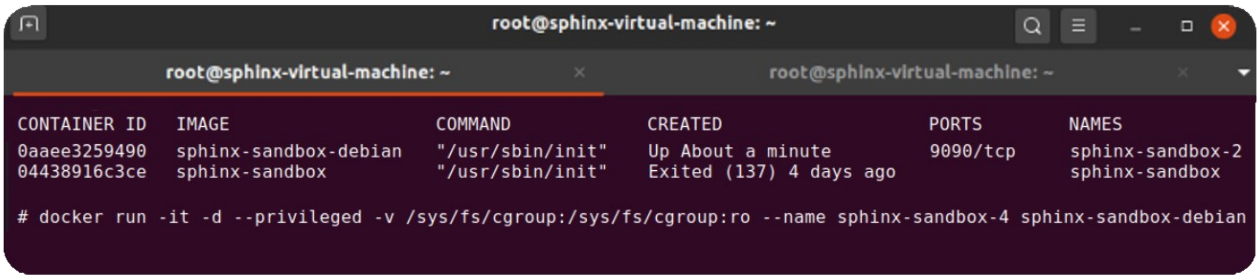
It is possible to include Docker containers as a virtualization system to conduct security tests on the deployed applications; however, this could not apply when the resources are unknown and could include security risks which include the Docker container take-over from potential malicious services. This is the main reason for not using containerization but virtualization technology when analyzing malwares for example. In ***Figure. 9*** the different services are presented (e.g. webgoat vulnerable machines running as a Docker). Using containers instead of virtual systems is a more flexible solution and provides higher performance and lower total overhead, reducing the required system resources. Furthermore, the total deployment effort and required deployment time is reduced as well.


root@sphinx-virtual-machine: ~
root@sphinx-virtual-machine: ~
root@sphinx-virtual-machine: ~

```
root@sphinx-virtual-machine:~# docker ps
 CONTAINER ID        IMAGE               PORTS                    NAMES
 94ba526eb708        webgoat/webgoat-7.1 0.0.0.0:8088->8080/tcp   sharp_mayer
 035584ca5e60        webgoat/webgoat-8.0 0.0.0.0:8085->8080/tcp   eager_buck
 da2e8b88afc5        webgoat/webgoat-8.0 0.0.0.0:8084->8080/tcp   stoic_meitner
 38b105fe13df        webgoat/webgoat-8.0 0.0.0.0:8083->8080/tcp   elated_wright
 05daf73f6465        webgoat/webgoat-8.0 0.0.0.0:8082->8080/tcp   zealous_williams
 610a6f562c86        webgoat/webgoat-8.0 0.0.0.0:8081->8080/tcp   lucid_elbakyan
 eb3102cbe45e        webgoat/webgoat-8.0 0.0.0.0:8080->8080/tcp   nifty_fermi
```

Figure. 9. The running Docker container that include KVM and Docker in a Docker capabilities

Similarly, it is possible to execute and maintain different docker containers which handle the hypervisor (KVM) accordingly and it is easy to rebuild Figure 10. Using this aspect, the potential malicious services are enclosed inside a virtual machine and have to escape the virtual machine which is difficult and then take-over tha Docker container in order to infect the main system. It is possible to include another border of isolation, executing the Docker containers inside a Virtual Machine.


root@sphinx-virtual-machine: ~
root@sphinx-virtual-machine: ~
root@sphinx-virtual-machine: ~

```
CONTAINER ID   IMAGE                  COMMAND            CREATED              PORTS      NAMES
0aaee3259490   sphinx-sandbox-debian  "/usr/sbin/init"   Up About a minute    9090/tcp   sphinx-sandbox-2
04438916c3ce   sphinx-sandbox         "/usr/sbin/init"   Exited (137) 4 days ago         sphinx-sandbox

# docker run -it -d --privileged -v /sys/fs/cgroup:/sys/fs/cgroup:ro --name sphinx-sandbox-4 sphinx-sandbox-debian
```

Figure 10. Sandboxing running KVM and provided by separate Docker containers

In Figure 10, the execution of the Docker run command will create another sandbox with the same topology as the others, running as separate virtual machines in a different subnet requiring 1 second for the deployment. Therefore, it is easy to deploy multiple sandboxes easy and fast. A comparison between standard and lightweight virtualization-containerization (Table 2) [3].

Parameter	Virtual Machines	Containers
Guest OS	*Each VM runs on a virtual hardware and the kernel is loaded into its own virtual memory*	*All the guests share the same kernel loaded in the physical memory*
Isolation	*Libraries and files are completely isolated*	*Directories can be mounted and can be shared between the containers and the physical machine*
Performance	*All instructions need to be translated between VMs and the physical machine, which incurs a performance decrease*	*Near native performance as compared to the physical machine*
Communication	*Virtual Ethernet devices*	*IPC mechanisms such as signals, sockets, etc*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Storage	*Need a large amount of disk space as each VM needs to store the whole OS and associated applications*	*Monitoring network traffic and suspicious network packets*

Table 2. Comparison between standard and lightweight virtualization

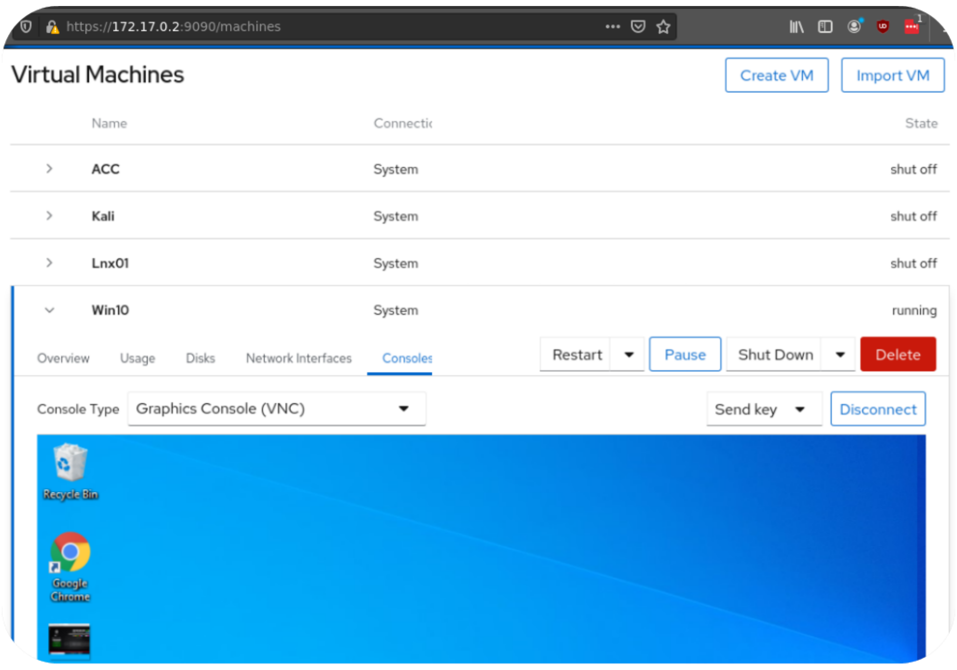
In terms of performance, containerization is more applicable than virtualization; however, security risks are included and are further discussed in the following section.

3.2.1 Security Aspects

The security aspects were previously described in summary and mostly include risks regarding the privileges and actions related to escaping a container. As for running KVM the deployed Docker containers require privileged mode, such security issues are increased. To mitigate such risks, we implement nested virtualization as proposed from other solutions (e.g. Cuckoo Sandbox). Therefore, any executed containers are running inside a VM. However, there are other modern approaches which are under development that require analysis and to be mentioned such as Kata containers[5] and Firecracker MicroVMs[6]. Other significant approaches include Rancher Harvester[7] (previously known as RancherVM) allowing to create VMs that run inside a Kubernetes cluster, called VM pods. However, such approaches are still under development, but for research purposes we deployed such solutions to test their impact and if appropriate for us to use them for the sandbox.

3.3 Operating systems in a Docker Container

In our approach we included the approach of running VMs inside the Docker containers. This approach gives us the possibility to easily deploy a sandbox environment that might include the required security hardness attributes (e.g. nested VMs, execution of operating systems inside a KVM).


https://172.17.0.2:9090/machines

Virtual Machines

Name	Connectio	State
ACC	System	shut off
Kali	System	shut off
Lnx01	System	shut off
Win10	System	running

Console Type: Graphics Console (VNC)

Figure 11. Virtualization of multiple systems running in a Docker container and using KVM

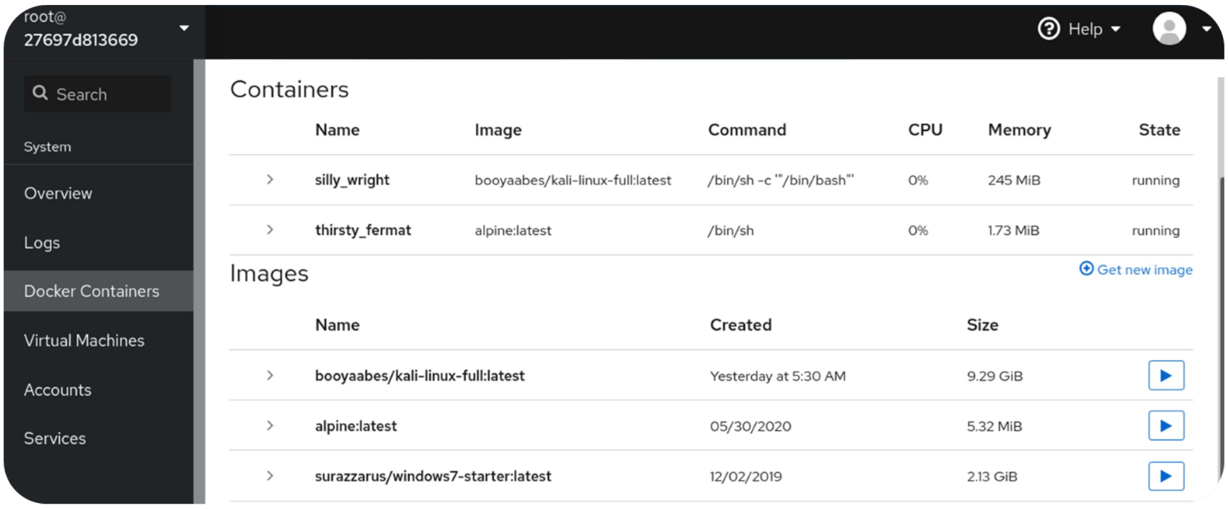
[5] https://katacontainers.io/

[6] https://firecracker-microvm.github.io/

[7] https://github.com/rancher/harvester

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

The virtual machines are by default deployed and starting the Docker container (running the Docker image) initiates the topology and the already deployed Virtual Machines. Therefore, it is easy to maintain complex topologies that include multiple Virtual Machines that could interact with each other Figure 11. Not only this but it is possible to maintain all the services provided from an operating system inside a Docker container. For example, there official Linux images exist as Docker images, published on Dockerhub[8].



Containers

	Name	Image	Command	CPU	Memory	State
›	silly_wright	booyaabes/kali-linux-full:latest	/bin/sh -c "/bin/bash"	0%	245 MiB	running
›	thirsty_fermat	alpine:latest	/bin/sh	0%	1.73 MiB	running

Images

	Name	Created	Size
›	booyaabes/kali-linux-full:latest	Yesterday at 5:30 AM	9.29 GiB
›	alpine:latest	05/30/2020	5.32 MiB
›	surazzarus/windows7-starter:latest	12/02/2019	2.13 GiB

Figure 12. Execution of Kali-linux and Alpine as Docker container

Example cases of running Linux distributions are presented in Figure 12 and most of the common and well-known distributions are official, maintaining continuous updates. For example, Fedora initiated version 23 in 2017 and Ubuntu from 2015 with the distribution version 12.04 Precise Pangolin, while CentOS initiated official version of Centos5 in 2017. Therefore, Docker containers are more frequently used lately and it is consequently a technology which is currently advancing and is broadly used. Security issues still exist for the containers; however, research is currently being conducted in how to mitigate these issues. Some of the mitigation actions include the creation of namespaces and for managing the privileges accordingly.

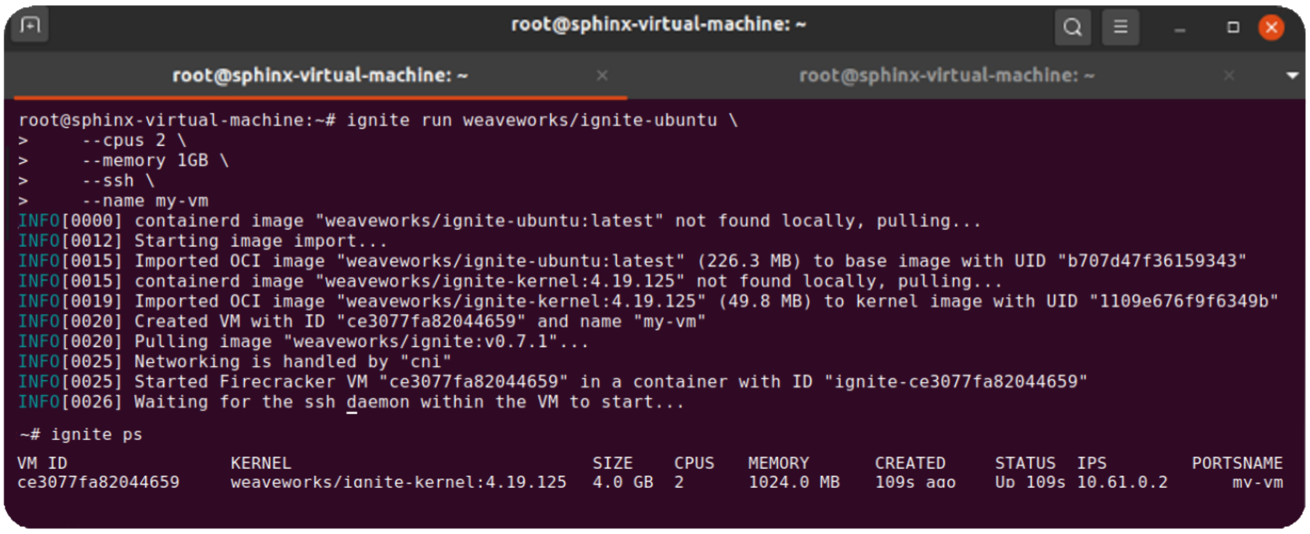
3.4 MicroVMs

MicroVMs as a context applies mostly to our deployments. The goal of a micro VM is to provide an isolated environment increasing cybersecurity and enhancing resilience through virtualization. The main benefits except the enhanced isolation is that micro VMs prevent latency and bottlenecks because they have been designed to only access a minimal set of resources. A new promising virtualization technology for maintaining micro VMS is called Firecracker by Amazon Web Services and the main goal is to enable large deployment workloads to run in lightweight virtual machines, providing enhanced security and workload isolation. This technology uses KVM to create and manage the MicroVMs.

Firecracker and more specifically Ignite Firecracker (an implementation that uses MicroVms), has been deployed for testing purposes (Figure 13). From the deployment it is concluded that even if currently only few Linux distributions are supported (e.g. Ubuntu) the deployment time is reduced and the performance is promising. Indeed, Ignite Firecracker combines the benefits of using containers but by initiating a kernel and all the isolation actions which a virtual machine might have. The total overhead is less and for analyzing the performance we evaluated the approaches of using MicroVMS instead of KVM and Docker containers.

[8] https://hub.docker.com/r/kalilinux/kali-rolling

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.


root@sphinx-virtual-machine: ~
root@sphinx-virtual-machine: ~
root@sphinx-virtual-machine: ~

```
root@sphinx-virtual-machine:~# ignite run weaveworks/ignite-ubuntu \
>     --cpus 2 \
>     --memory 1GB \
>     --ssh \
>     --name my-vm
INFO[0000] containerd image "weaveworks/ignite-ubuntu:latest" not found locally, pulling...
INFO[0012] Starting image import...
INFO[0015] Imported OCI image "weaveworks/ignite-ubuntu:latest" (226.3 MB) to base image with UID "b707d47f36159343"
INFO[0015] containerd image "weaveworks/ignite-kernel:4.19.125" not found locally, pulling...
INFO[0019] Imported OCI image "weaveworks/ignite-kernel:4.19.125" (49.8 MB) to kernel image with UID "1109e676f9f6349b"
INFO[0020] Created VM with ID "ce3077fa82044659" and name "my-vm"
INFO[0020] Pulling image "weaveworks/ignite:v0.7.1"...
INFO[0025] Networking is handled by "cni"
INFO[0025] Started Firecracker VM "ce3077fa82044659" in a container with ID "ignite-ce3077fa82044659"
INFO[0026] Waiting for the ssh daemon within the VM to start...

~# ignite ps

VM ID               KERNEL                            SIZE   CPUS   MEMORY      CREATED    STATUS   IPS         PORTSNAME
ce3077fa82044659    weaveworks/ignite-kernel:4.19.125 4.0 GB 2      1024.0 MB   109s ago   Up 109s  10.61.0.2        my-vm
```

Figure 13. Ignite Firecracker and deployment of a MicroVM

Using Ignite we were able to deploy an Ubuntu Linux distribution as a virtual machine and by executing the command we are able to set the amount of CPUs, diskspace size and the reserved memory (RAM) that will be used for each of the MicroVms. An issue with Ignite Firecracker is the incapability for currently running Windows operating systems or other than the Linux distributions. However, similar technologies in the past (RancherVM) managed to run Windows 7 as Micro VM. We tested the solution of RancherVM; however, only Windows 7 operating systems are supported.

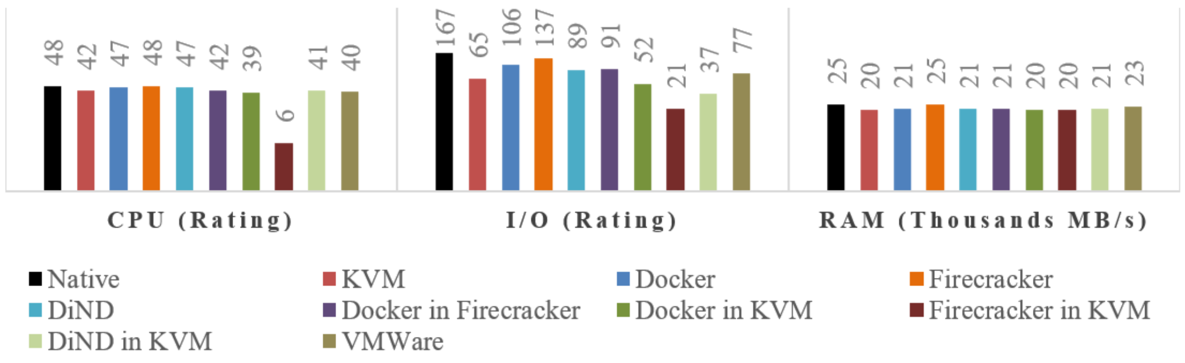


Figure 14. Comparison of the performance between the approaches

In all our tests we ensured that all the other applications were closed, and no additional overhead was added except for the main system services. The evaluation metrics of course depend on the main system resources and our purpose was to compare the differences between the used technologies. The results from the performance evaluation and benchmarks are presented in Figure 14.

Taking the above into consideration, the results from the performance evaluation present that docker containers maintain low overhead, mainly in terms of I/O – disk cache writing and reading speeds. Furthermore, we have also investigated the total overhead in terms of both the used hard disk space and memory size for deploying the vulnerable systems or services. For the evaluation tests of the Linux hosts/services we used Sysbench for the memory tests and Stress-ng for testing the Control Process Unit (CPU) and collecting disk cache input/output (I/O) benchmarks. For the Windows system hosts we used Novabench and the considered metrics are as follows:

1. **CPU:** CPU performance tests using the Stress-ng for each different technology. Rating is considered as the number of iterations of the CPU stressor run for 20 seconds.
2. **I/O – Hard disk:** Performance test using Stress-ng related to the disk's cache measuring the input/output operations per second. Rating is considered as the number of iterations of the disk cache stressor during the run for 20 seconds.
3. **RAM memory:** Effective RAM performance by calculating the writing speed (Mega Bytes per second – MB/s).

Summarizing, MicroVMs seem to be a very good option in terms of performance; however, incompatibilities exist. Therefore, MicroVMs is the best option if the services or systems which are on the test include services running exclusively on Linux (e.g. Docker Containers).

3.5 Networking Capabilities

Virtualization can virtualize hardware devices and this is a main benefit. Similarly, containerization simulates the network device creating a virtual ethernet connection for each container (Figure 15). Every container is running separately using the same hypervisor for the KVM and using different ethernet connections.

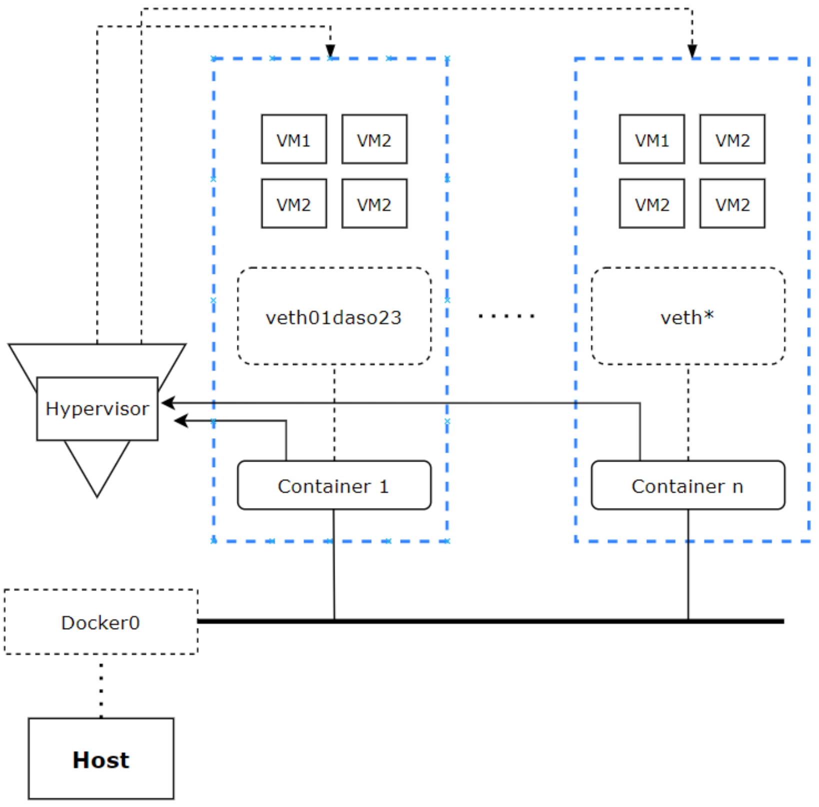
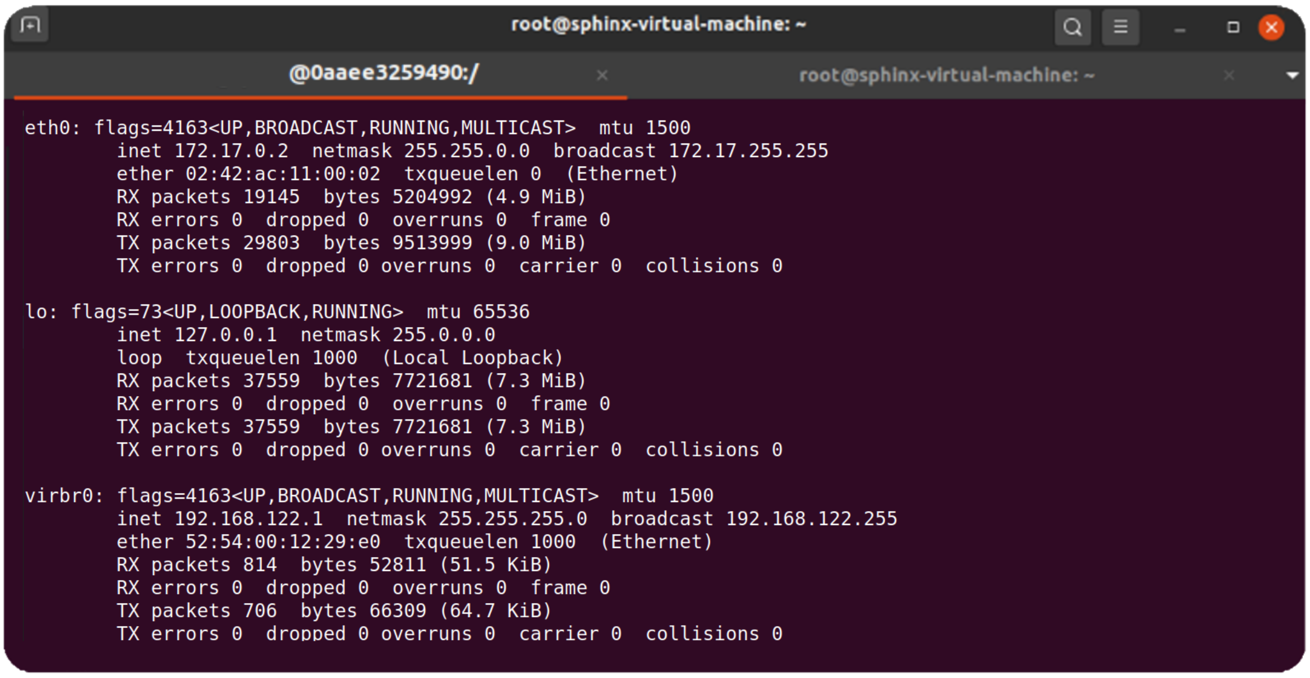


Figure 15. Docker containers running KVM

Every virtual machine is connected using the virtual network interface called virbr0. Every container has its own virbr0 isolated, meaning that VM1 for example cannot directly communicate to another VM from another container. Even the connection to the DHCP from virbr0 of each container do not accept any network traffic from other VM that is running on a different container.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.


root@sphinx-virtual-machine: ~
@0aaee3259490:/
root@sphinx-virtual-machine: ~

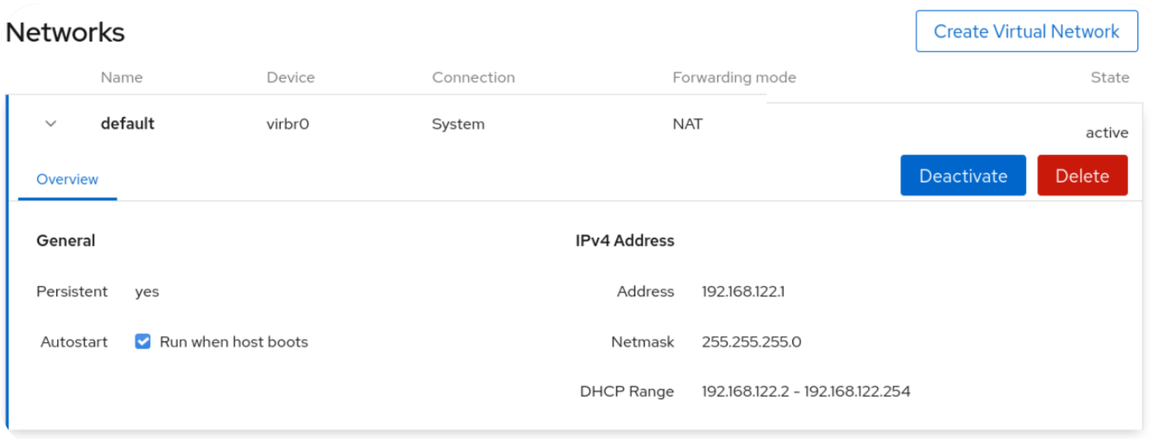
```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 19145  bytes 5204992 (4.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 29803  bytes 9513999 (9.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 37559  bytes 7721681 (7.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 37559  bytes 7721681 (7.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:12:29:e0  txqueuelen 1000  (Ethernet)
        RX packets 814  bytes 52811 (51.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 706  bytes 66309 (64.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 16. Network interfaces inside the container

As presented in Figure 16, eth0 is the main interface of the Docker container. Every Docker container has its own IP address using veht interfaces provided by Docker. Following, virbr0 is the virtual interface created by KVM inside the Docker container. Therefore, every VM which is inside the same container can send and receive network traffic from other VMs that are inside the container. More options are provided if we need to change the interface that KVM uses and the DHCP range, among other settings (Figure 17).



Name	Device	Connection	Forwarding mode	State
default	virbr0	System	NAT	active

General

Persistent	yes
Autostart	✓ Run when host boots

IPv4 Address

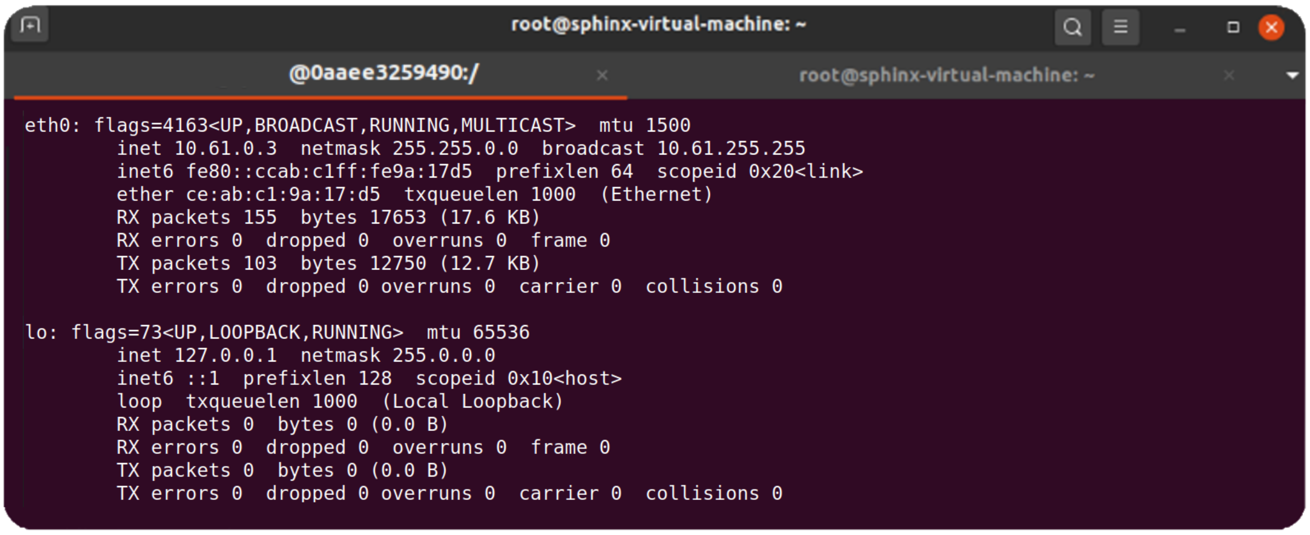
Address	192.168.122.1
Netmask	255.255.255.0
DHCP Range	192.168.122.2 - 192.168.122.254

Figure 17. Creation and editing of Virtual Networks

Taking into consideration the above, we conclude that we might have options that we did not yet consider well and that we have to test more the approaches in order to analyze and discover the possibilities. For example, we can also create subnets using other virtual devices for the VMs to actually use other virtual network interfaces (e.g. virbr0, virbr1, virbr2 etc.). Creating different virtual network interfaces, we can isolate the VMs more into subnets or create even much complex network topologies. For example, we can run various virtual machines using different MAC addres ses and a different virtual network interface (compatibility extensions include virtio, e1000e etc.). More comprehensive tests will be described in the next version of this deliverable as well as the evaluation results regarding the isolation properties of the sandbox.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Regarding Ignite Firecracker and the network capabilities, similar approach is followed by virtualizing the network interfaces. In this case every MicroVM has its own virtual ethernet (vethxxxx).


root@sphinx-virtual-machine: ~
@0aaee3259490:/
root@sphinx-virtual-machine: ~

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.61.0.3  netmask 255.255.0.0  broadcast 10.61.255.255
        inet6 fe80::ccab:c1ff:fe9a:17d5  prefixlen 64  scopeid 0x20<link>
        ether ce:ab:c1:9a:17:d5  txqueuelen 1000  (Ethernet)
        RX packets 155  bytes 17653 (17.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 103  bytes 12750 (12.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 18. Network settings for each Virtual Machine

The DHCP range for these interface ranges from 10.61.0.2 – 10.61.254.254 by default; however, we can configure this setting at the time we are creating the MicroVMs. The restrictions and potential benefits of using the network virtualization have not been tested, but we assume that since Ignite Firecracker uses KVM, similar attributes to the other deployment will be present.

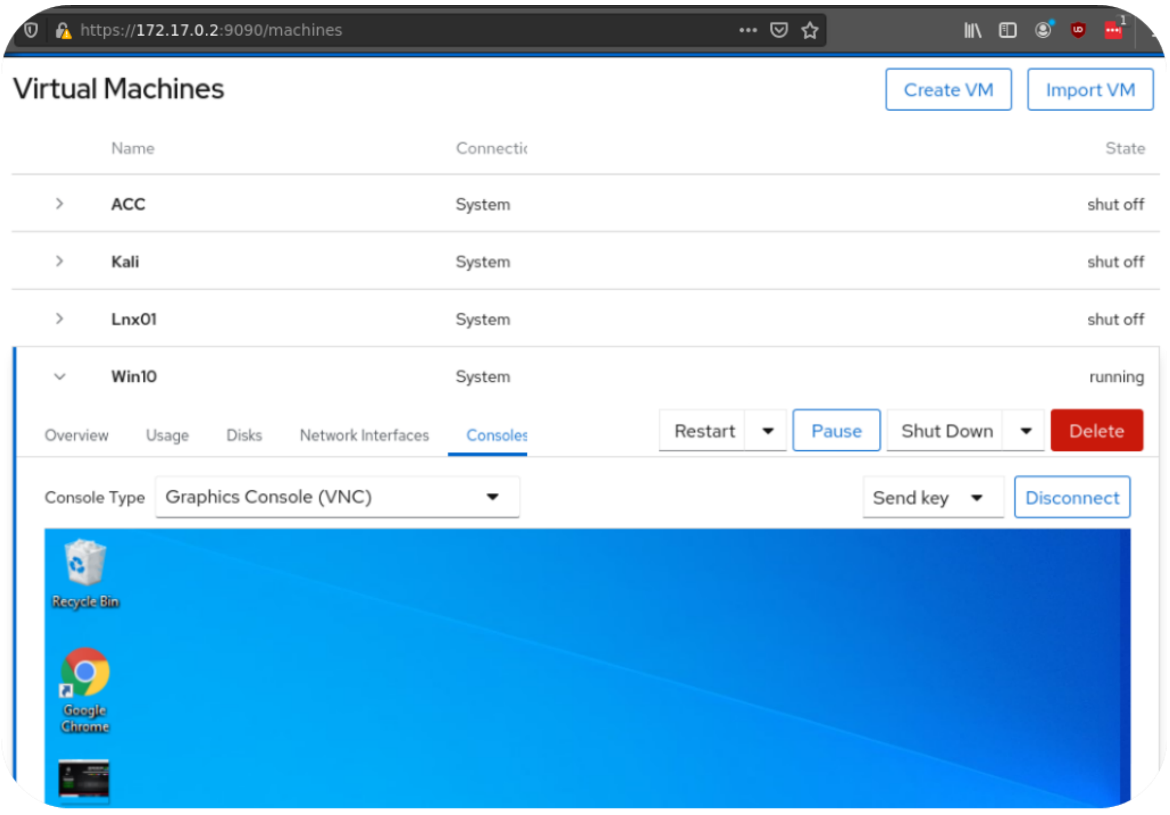
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

4 Integration Capabilities

As presented in Section 3, it is possible to deploy different topologies using KVM and/or Docker containers. Initially our focus was in executing high-performance solutions requiring less system resources (e.g. RAM, hard disk). However, for successfully deploying other operating systems and services our goal was to discover compatible and flexible solutions.

4.1 Running VMs in a Docker Container

The first option which includes high compatibility is to deploy a KVM using Docker containers. This is our main approach which is already deployed as a demo and by including an existing topology that is also used for the Automated Cybersecurity Certification (Deliverable 3.5). In this approach we execute KVM and the virtual machines using Docker containers. The benefits of deploying KVM instead of Docker containers is that virtualization using KVM provides stronger isolation than Docker containers and therefore any unwanted or potential malicious processes are strictly restricted. An issue from our deployment is that KVM is initiated using Docker containers requiring for containers to access the root services (privileged mode). Therefore, we propose for this deployment to use another virtual machine which will execute the Docker containers. However, it is a secure solution if we assume that a malicious process is difficult to escape from KVM and proceed to the main host.



https://172.17.0.2:9090/machines

Virtual Machines

Name	Connectic	State
ACC	System	shut off
Kali	System	shut off
Lnx01	System	shut off
Win10	System	running

Console Type: Graphics Console (VNC)

Figure 19. Running Win10 and other VMs using KVM inside a Docker container

As presented in Figure 19, a Docker container with the IP address of 172.17.0.2 is running KVM which hosts 4 different operating systems:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

1. ACC – Linux distribution running 4 dockers with ELK stack
2. Kali Linux – For executing attacks and security tests
3. Ubuntu/Debian/Lnx01 – A Linux distribution for hosting the services which are for testing
4. Windows10 – Operating system for hosting services that require Windows OS

Of course the deployment could be different and include Windows Server and other Linux distributions that will eventually result in a more complex topology.

4.2 Running Virtual Machines in a VM

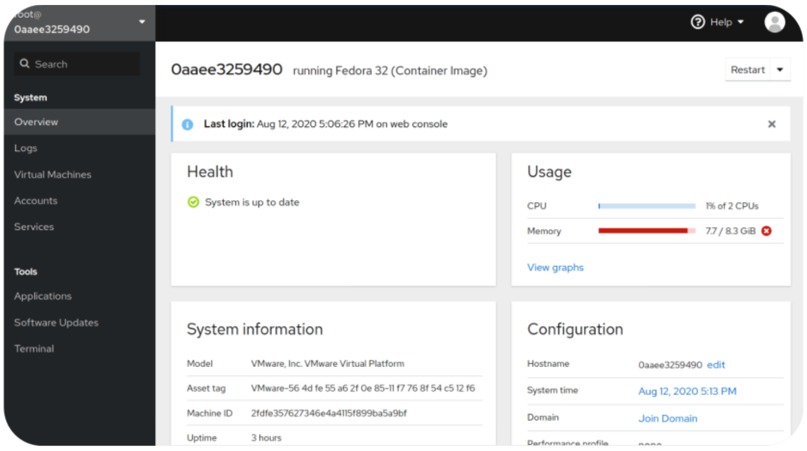
In the approach presented in Section 4.1, the possibility to run nested virtualization was also presented in order to isolate more the processes and reduce the attack surface of the host. Nested virtualization provides increased security and isolation and most of the approaches (e.g. Cuckoo sandbox), uses this approach. The drawbacks coming from this deployment is the increased overhead, high resource consumption resulting in lower performance. However, we tested the execution of ELK stack and other services and it was possible for a moderate machine to host the required systems. However, in terms of scaling this approach is difficult to manage and the approach of using MicroVMs seems a better solution.

4.3 Testing Untrusted Sources

Our deployment and tool can be used for conducting security tests or to monitor the behaviour of the deployed services and the interaction between them. Therefore, untrusted services can possibly be deployed and used for extraction of reports that describe the interactions. For meeting this requirement, we can replicate the systems-on-the-test using qcow2 images and by executing a KVM. Using Clonezilla for example we can clone an entire operating system or pull/import and execute a Docker image of the service.

4.4 Management and Reports

Management and reporting is important to the sandbox component, since it is very useful to extract information regarding the deployed services. Therefore, we can deploy test cases relevant to security incidents and to include a testbed that will provide us information which can be used for other software components and tools as well. For example, it is important for Intrusion Detection Systems (IDS) to have example datasets to test with or for a SIEM to have test cases to validate with. Furthermore, management and reports are very important to the end users to increase their situational awareness and to understand the topologies better.



0aaee3259490 running Fedora 32 (Container Image)

Last login: Aug 12, 2020 5:06:26 PM on web console

Health

System is up to date

Usage

CPU		1% of 2 CPUs
Memory		7.7 / 8.3 GiB

System information

Model	VMware, Inc. VMware Virtual Platform
Asset tag	VMware-56 4d fe 55 a6 2f 0e 85-11 f7 76 8f 54 c5 12 f6
Machine ID	2fdfe357627346e4a4115f899ba5a9bf
Uptime	3 hours

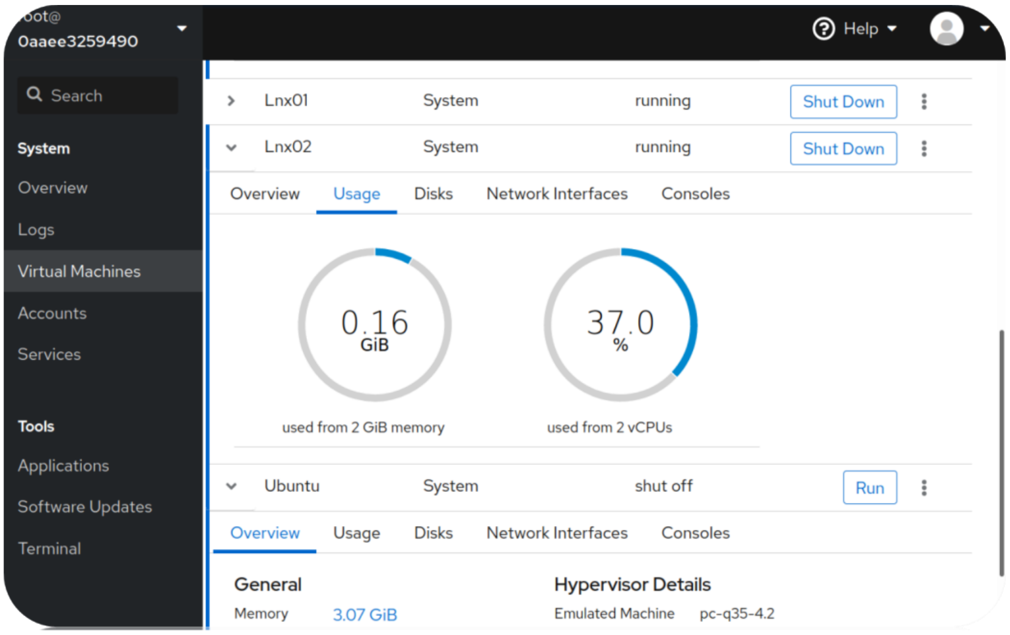
Configuration

Hostname	0aaee3259490 edit
System time	Aug 12, 2020 5:13 PM
Domain	Join Domain
Performance profile	none

Figure 20. Management and status of the Host

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

For the deployment, it is not considered an important approach to include by-default remote control tools such as VNC and we can only have SSH connection to parse any required commands. However, by using VNC for example it is possible for the end users or administrators to get to the debugging mode. Therefore, we can reach end users which are not very familiar or experienced to CLI or more advanced options for managing the deployed services/operating systems. Using Cockpit[9] we can have such information and manage better our systems. Not only Cockpit can provide information regarding the Virtual Machines but can also provide major updates or other information regarding the status of the host. For example, in our deployment we execute a Fedora distribution as a Docker Container and the management services provide information of potential updates of that container. We also have the opportunity to manage Docker Containers and Docker images using a graphical user interface (Figure 20).



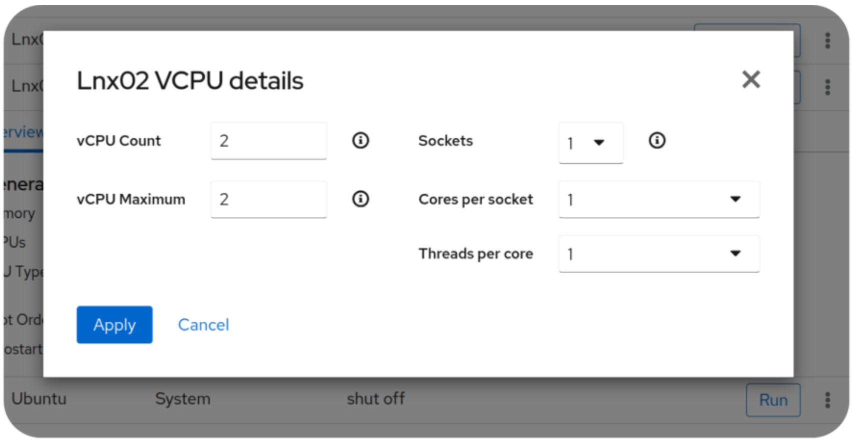
Lnx01	System	running	Shut Down	⋮
Lnx02	System	running	Shut Down	⋮
Ubuntu	System	shut off	Run	⋮

Memory 3.07 GiB

Emulated Machine pc-q35-4.2

Figure 21. Configuration options of the Virtual Machines

Using a graphical user interface the end users can easily configure the Virtual Machines and set the minimum/maximum requirements (Figure 21, Figure 22).



Lnx02 VCPU details

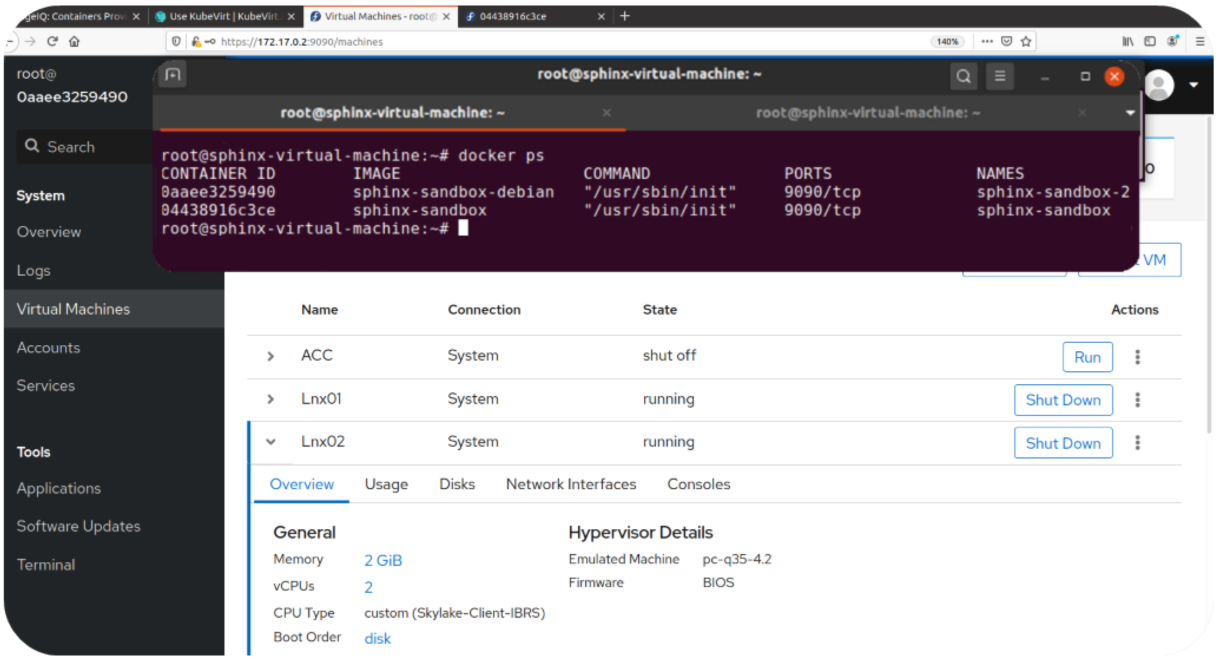
vCPU Count	2	Sockets	1
vCPU Maximum	2	Cores per socket	1
		Threads per core	1

Figure 22. Setting up the resources (CPU)

[9] https://cockpit-project.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

This way the end user can scale up or down the systems according to the requirements. We intend to set the initial details for the services using an API (will be provided in a later deliverable). This way the end user will be able to define the required service to be deployed (e.g. provide a qcow2 image, Docker image etc.) and the minimum requirements for the deployed services, as well as the network topology.



```
root@sphinx-virtual-machine:~# docker ps
CONTAINER ID        IMAGE                  COMMAND            PORTS          NAMES
0aaee3259490        sphinx-sandbox-debian  "/usr/sbin/init"   9090/tcp       sphinx-sandbox-2
04438916c3ce        sphinx-sandbox         "/usr/sbin/init"   9090/tcp       sphinx-sandbox
root@sphinx-virtual-machine:~# █
```

Name	Connection	State	Actions
ACC	System	shut off	Run
Lnx01	System	running	Shut Down
Lnx02	System	running	Shut Down

Overview Usage Disks Network Interfaces Consoles

General

Memory	2 GiB
vCPUs	2
CPU Type	custom (Skylake-Client-IBRS)
Boot Order	disk

Hypervisor Details

Emulated Machine	pc-q35-4.2
Firmware	BIOS

Figure 23. Sandboxing using KVM and Docker containers

Summarizing, Figure 23 presents an instance of our deployment showing two deployed sandboxes (sphinx-sandbox and sphinx-sandbox-2). Deploying the docker images is easy and in this figure the two Docker containers have two different IP addresses (172.17.0.2 and 172.17.0.3). Inside each of them 3-4 VMs are running separately providing the isolation which is already discussed and will be described in more details in Section 5.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

5 Network and System Isolation

This section is appointed to the actions that include testing of the system isolation and network interaction between the deployed services or operating systems. Since it is important for the deployed services to be running in a restricted and controllable environment, maintaining isolation while allowing them to interact with each other is important. To meet this requirement, we deploy virtual machines using KVM and using the same virtual interaface (e.g. virbr0) it is possible for the services to interact each other, keeping the Docker container intact and so the main host. Escaping a docker container might be possible; however, escaping from virtual machine is very difficult for a potential malicious service to succeed.

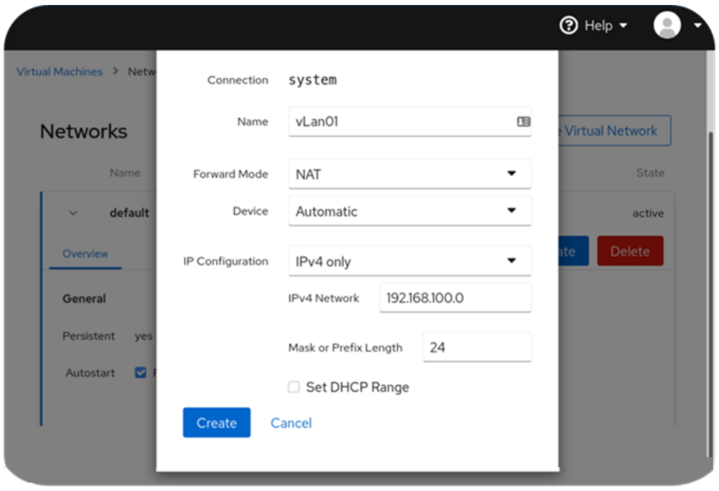
5.1 Overview

For being confident and to evaluate the isolation capabilities of the selected approaches we selected a list of common ransomware and malicious software. For testing purposes, we deployed WannaCry retrieved from theZoo[10], a popular Github repository. Our initial purpose was to execute both common services (e.g. apache, SQL server) and malicious software as well to test the approach. Towards this direction, we tried to have interactions between the virtual systems and to examine the network traffic going beyond the strict boundaries of the virtualization.

WannaCry itself is a ransomware worm, meaning that it can infect other windows computers; however, this procedure was not produced any results and did not infect other operating systems. Therefore, we are currently focused more on providing more malicious worms (e.g. SSH worms that extend to Linux distributions). The results from this evaluation will be provided in a later and final version of this deliverable.

5.2 Networking

Using virbr0 from KVM and veth from Docker Containers it is possible to create subnetworks which are similar to existing ones. Of course, services that are provided such as gateways, DHCP or other are provided using virtualization; however, it is possible to even deploy some of these services as separate services in order to apply more to the reality if required.



Connection system
Name vLan01
Forward Mode NAT
Device Automatic
IP Configuration IPv4 only
IPv4 Network 192.168.100.0
Mask or Prefix Length 24
☐ Set DHCP Range

Figure 24. Network confirmation for creating Virtual LAN

Therefore, it is possible to deploy realistic network topologies accordingly. Using the graphical user interface (or APIs) it possible for us to initiate such options (Figure 17). Being able to create our own network topologies

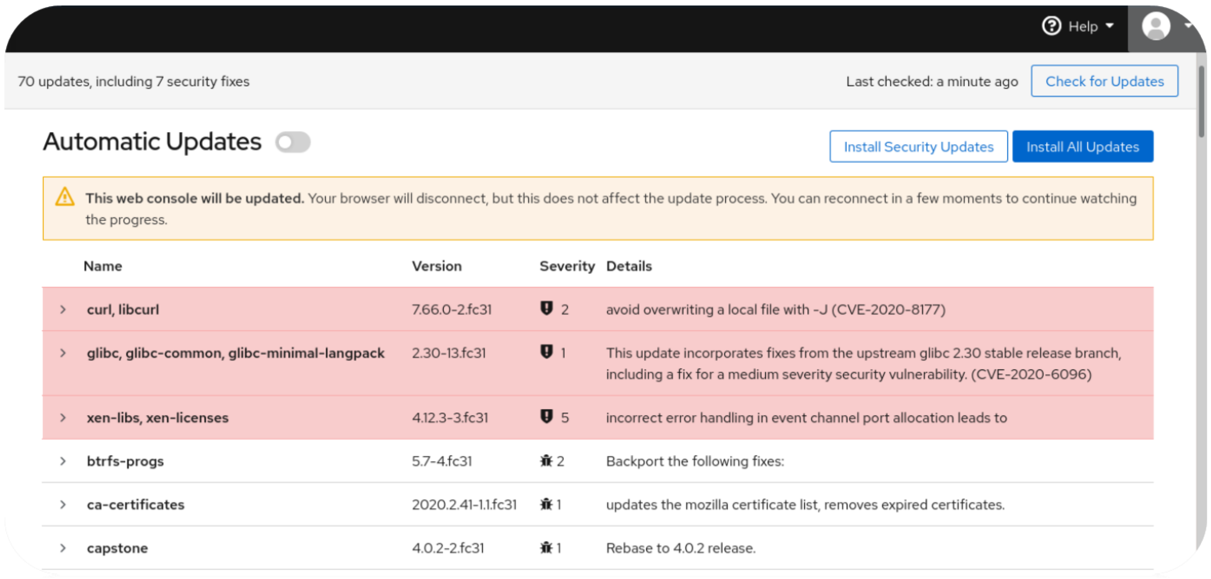
[10] https://github.com/ytisf/theZoo

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

or replicate the existing ones we are able to execute security scenarios and retrieve/collect network traffic, data, log files being able to understand the interactions between the deployed or replicated systems and services.

5.3 System Behavior

Using internal services for managing the updates and being able to access the managers internal processes we are able to understand more about the sandbox itself. The options we have maintain and increase the security posture of the deployed services even more, without affecting the internal deployed services that were created using KVM. Therefore, we maintain a security border between the host, the sandbox and the virtual machines.



70 updates, including 7 security fixes

Last checked: a minute ago

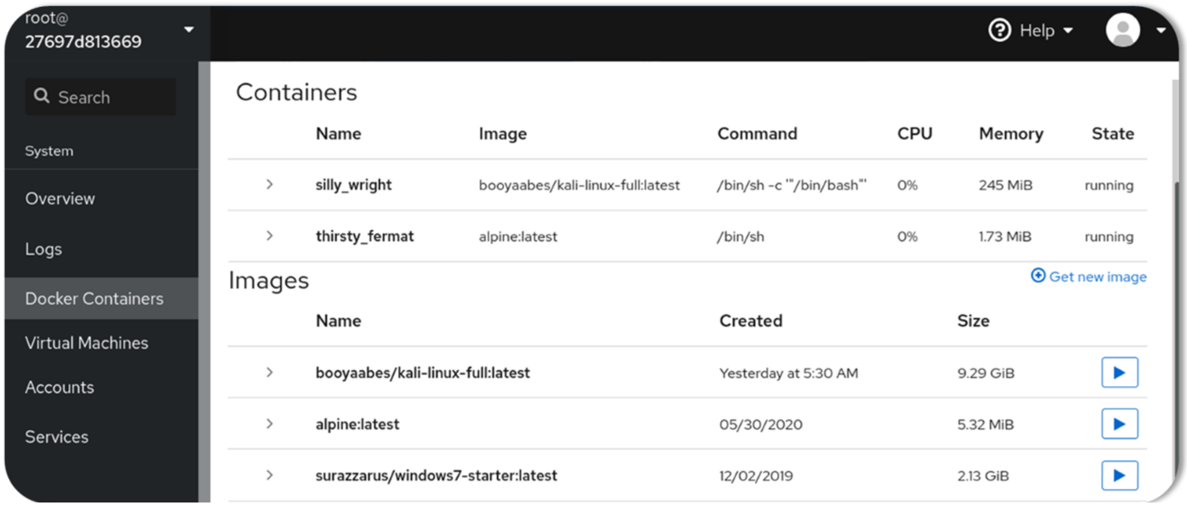
Automatic Updates

This web console will be updated. Your browser will disconnect, but this does not affect the update process. You can reconnect in a few moments to continue watching the progress.

Name	Version	Severity	Details
curl, libcurl	7.66.0-2.fc31	2	avoid overwriting a local file with -J (CVE-2020-8177)
glibc, glibc-common, glibc-minimal-langpack	2.30-13.fc31	1	This update incorporates fixes from the upstream glibc 2.30 stable release branch, including a fix for a medium severity security vulnerability. (CVE-2020-6096)
xen-libs, xen-licenses	4.12.3-3.fc31	5	incorrect error handling in event channel port allocation leads to
btrfs-progs	5.7-4.fc31	2	Backport the following fixes:
ca-certificates	2020.2.41-1.1.fc31	1	updates the mozilla certificate list, removes expired certificates.
capstone	4.0.2-2.fc31	1	Rebase to 4.0.2 release.

Figure 25. Automated updates for the Docker container which manages the KVM

Figure 25, presents the potential updates and the importance (severity) addressing also the CVE (Common Vulnerabilities and Exposures) numbering. Furthermore, using the graphical user interface (Figure 26) the possibility to manage docker containers is presented in case we want to deploy the required services not in a KVM but as Docker containers (by understanding the security risks that this approach provides).



Containers

Name	Image	Command	CPU	Memory	State
silly_wright	booyaabes/kali-linux-full:latest	/bin/sh -c "/bin/bash"	0%	245 MiB	running
thirsty_fermat	alpine:latest	/bin/sh	0%	1.73 MiB	running

Images

Name	Created	Size
booyaabes/kali-linux-full:latest	Yesterday at 5:30 AM	9.29 GiB
alpine:latest	05/30/2020	5.32 MiB
surazzarus/windows7-starter:latest	12/02/2019	2.13 GiB

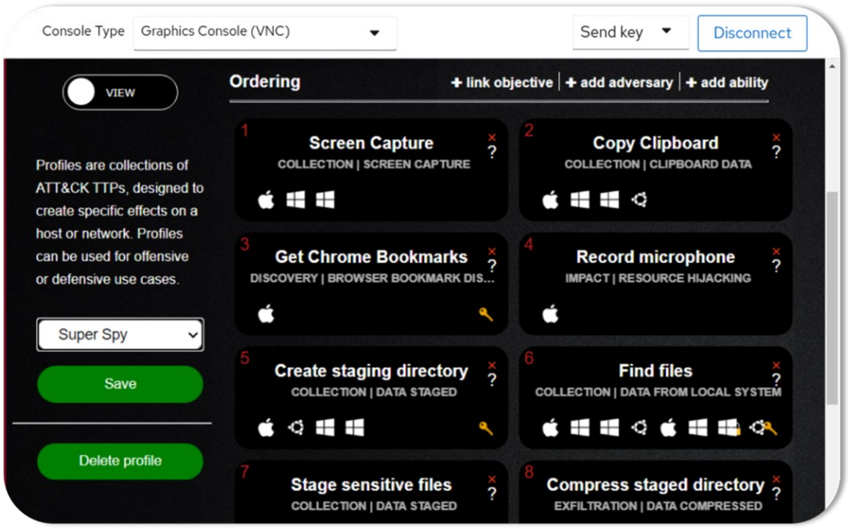
Figure 26. Interface for managing the Docker images and Docker containers

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

It is possible as well to deploy services directly as Docker containers; however, this option is applicable and secure if the host system that includes docker is running in a Virtual Machine. Using Docker containers could include some elements of isolation; however, is not a secure option since a malicious process could escape the containerization and access the host system resources and escalate privileges. For example, if malware analysis is to be conducted, it is safer to execute in isolated inside a virtual machine and to have a separate kernel for the main processes to be executed.

5.3.1 Malware Analysis

For testing the sandboxing we deployed Cuckoo sandbox and afterwards we deployed adversary emulation using Caldera and tried to capture the triggered security events (Figure 26). Sandboxes are very popular for conducting static and dynamic malware analysis.



Profiles are collections of ATT&CK TTPs, designed to create specific effects on a host or network. Profiles can be used for offensive or defensive use cases.

1 Screen Capture
COLLECTION | SCREEN CAPTURE

2 Copy Clipboard
COLLECTION | CLIPBOARD DATA

3 Get Chrome Bookmarks
DISCOVERY | BROWSER BOOKMARK DIS...

4 Record microphone
IMPACT | RESOURCE HIJACKING

5 Create staging directory
COLLECTION | DATA STAGED

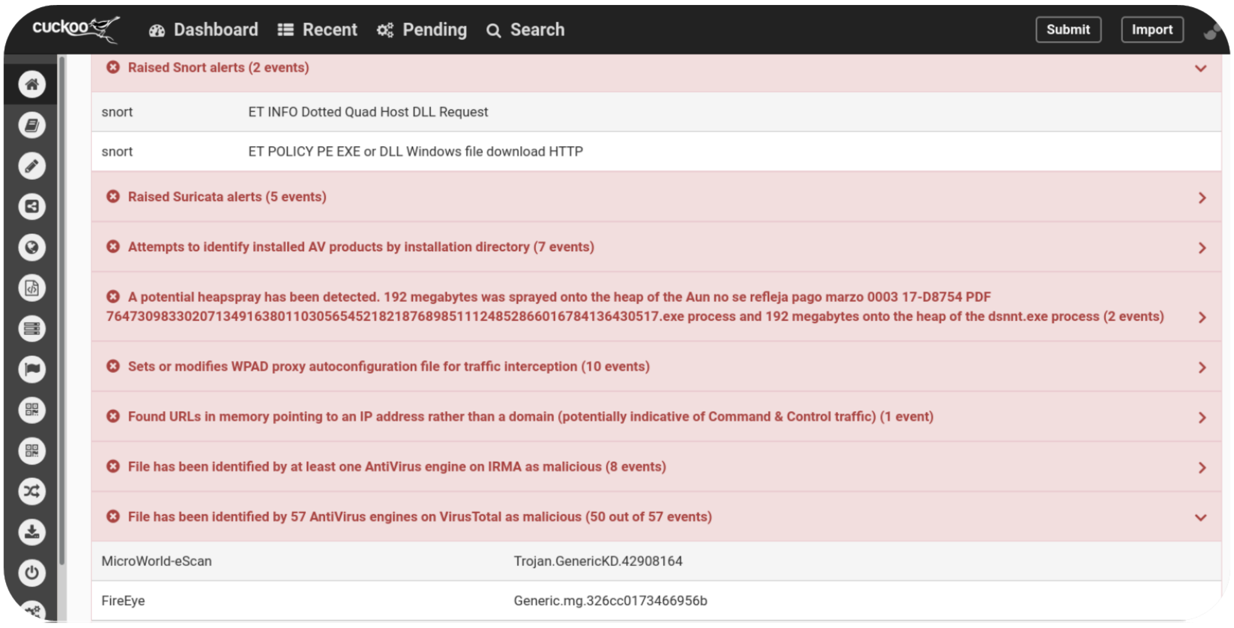
6 Find files
COLLECTION | DATA FROM LOCAL SYSTEM

7 Stage sensitive files
COLLECTION | DATA STAGED

8 Compress staged directory
EXFILTRATION | DATA COMPRESSED

Figure 27. Adversary emulation using Caldera

As presented in Figure 28, it is possible to analyze a file or executable file and discover if it a malicious process or hidden payload inside the uploaded file.



Raised Snort alerts (2 events)

snort	ET INFO Dotted Quad Host DLL Request
snort	ET POLICY PE EXE or DLL Windows file download HTTP

Raised Suricata alerts (5 events)

Attempts to identify installed AV products by installation directory (7 events)

A potential heapspray has been detected. 192 megabytes was sprayed onto the heap of the Aun no se refleja pago marzo 0003 17-D8754 PDF 76473098330207134916380110305654521821876898511124852866016784136430517.exe process and 192 megabytes onto the heap of the dsnnt.exe process (2 events)

Sets or modifies WPAD proxy autoconfiguration file for traffic interception (10 events)

Found URLs in memory pointing to an IP address rather than a domain (potentially indicative of Command & Control traffic) (1 event)

File has been identified by at least one AntiVirus engine on IRMA as malicious (8 events)

File has been identified by 57 AntiVirus engines on VirusTotal as malicious (50 out of 57 events)

MicroWorld-eScan	Trojan.GenericKD.42908164
FireEye	Generic.mg.326cc0173466956b

Figure 28. Cuckoo Sandbox analyzing a Trojan listed in 57 AntiVirus engines

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

The Cuckoo Sandbox is possible to be executed inside our sandbox in order to extend our approach for discovering malware and ransomware accordingly. More details and screenshots are extracted when executing the scanning for the end user to understand better the behaviour of the potential malicious payload. Summarizing, in our approach the sandbox is not deployed for extracting details regarding a potential malicious file, but to create a safe and secure environment for conducting security tests. Therefore, we aspire to extend this approach and to provide the offensive tactics for inspecting the elements inside the sandbox. This inspection includes malware analysis but extends more to the vulnerability scanning and for understanding better the internal interactions of the systems which are executed inside the sandbox.

5.4 Sandbox Isolation Capabilities

KVM or running systems using virtualization guarantees the isolation in terms of filesystem access and processes and it is possible to isolate networks as well in order to not directly interact with other components. Docker containers also include some of these options; however the security risks are increased. For evaluating the isolation capabilities we intend to execute malicious worms or other offensive tactics which focus on privilege escalation, escaping the Docker containers and to conduct research for the possibility to escape from the virtualization.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

6 Summary and Conclusions

The capabilities for the SPHINX to automatically conduct the processes that are related to security testing are easy to demonstrate using the sandbox and data inspection component. Using the sandbox (SB) is is easy to deploy services and systems and to conduct security analysis accordingly. Inside the sandbox it is possible to deploy other SPHINX components or to interact with the deployed systems and services, extracting realistic data from virtualized systems.

Our goal was to discover the best practices and to easily deploy complex topologies in order to create the digital environment where SPHINX components will be tested, to extract data which will be used to enhance the process of the other SPHINX components and to provide a safe and secure environment for conducting security tests. Therefore, The Automated Cybersecurity Certification (ACC) sub-component is directly related to the sandbox as it is important to include ACC and SB when we want to conduct security tests to untrusted or unknown resources and services. Furthermore, using the SB it is possible to sandbox the behavior and monitor the interactions between software components, either if are services or systems. The automation in terms of deployment and other enhanced options are currently under development and we aspire to include processes for the end user to be able to easily deploy services and systems using APIs. Such details will be included in a later version of this deliverable.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

7 References

[1] Canfora, G., Di Sorbo, A., Mercaldo, F., & Visaggio, C. A. (2015, May). Obfuscation techniques against signature-based detection: a case study. In 2015 Mobile Systems Technologies Workshop (MST) (pp. 21-26). IEEE.

[2] Jamalpur, S., Navya, Y. S., Raja, P., Tagore, G., & Rao, G. R. K. (2018, April). Dynamic malware analysis using cuckoo sandbox. In 2018 Second international conference on inventive communication and computational technologies (ICICCT) (pp. 1056-1060). IEEE.

[3] Gupta, D., & Mehte, B. M. (2013, August). Forensics analysis of sandboxie artifacts. In International Symposium on Security in Computing and Communication (pp. 341-352). Springer, Berlin, Heidelberg.

[4] Kale, G., Bostanci, E., & Çelebi, F. (2018, October). Differences between Free Open Source and Commercial Sandboxes. In Proceedings of the International Conference on Cyber Security and Computer Science, Safranbolu, Turkey (pp. 18-20).

[5] Inoue, D., Yoshioka, K., Eto, M., Hoshizawa, Y., & Nakao, K. (2009). Automated malware analysis system and its sandbox for revealing malware's internal and external activities. IEICE transactions on information and systems, 92(5), 945-954.

[6] Lindorfer, M., Kolbitsch, C., & Comparetti, P. M. (2011, September). Detecting environment-sensitive malware. In International Workshop on Recent Advances in Intrusion Detection (pp. 338-357). Springer, Berlin, Heidelberg.

[7] Gu, Z., & Zhao, Q. (2012). A state-of-the-art survey on real-time issues in embedded systems virtualization.

[8] Bui, T.: Analysis of Docker Security. (2015).

[9] Eiras, R.S.V., Couto, R.S., Rubinstein, M.G.: Performance evaluation of a virtualized HTTP proxy in KVM and Docker. 2016 7th International Conference on the Network of the Future, NOF 2016. (2017).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.