A Classification of Malware Sandboxes and Their Architectures

Stefano Bistarelli[1], Emanuele P. Burberi[2] and Francesco Santini[1]

[1] *Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Perugia, Italy*
[2] *Independent Researcher*

Abstract

The development of sandboxes has evolved in parallel with malware improvements, providing secure environments for the execution and analysis of malicious software. With the increasing number of sandboxes being developed, each offering distinct features, selecting the most suitable solution for specific needs has become increasingly complex. This paper aims to thoroughly understand sandbox technologies by introducing their foundational concepts and offering a detailed review of currently available solutions. We categorize sandboxes into four groups: the first two classes are readily available in current products on the market, and we compare them by evaluating some important characteristics we identify in this work. The second group of two sandboxes includes novel architectures we propose, for which solutions still need to be customized ad-hoc. To help users choose the right solution for them, we refine this categorization by defining three distinct use cases for these sandboxes. Their core features can improve the isolation and scalability of the last two new architectures.

Keywords

Malware Analysis, Sandbox Architectures, Sandboxing Techniques, Sandboxes Comparison

1. Introduction

Over time, sandboxes have evolved both in their objectives and in their implementations. Although the original concept of a sandbox focused on isolating the environment from the system, modern sandboxes are expected not only to provide this isolation and include built-in support for malware analysis tools. These tools may either be integrated within the sandbox itself (*internal* tools) or available through third-party installations (*external* tools) [1]. This enhancement enables users to conduct comprehensive analyses of malware using *static analysis*, *dynamic analysis*, and *behavioral analysis* [2, 3, 4].

In the context of malware analysis, dynamic analysis involves executing the malware within a sandbox and monitoring the system for any changes, suspicious system calls, or requests to malicious IP addresses. Behavioral analysis is a subset of dynamic analysis that focuses specifically on understanding the behavior of the malware, aiming to recognize patterns within its actions and extract valuable information about the malware itself, such as its scope and target. In addition to dynamic analysis, static analysis is performed by examining the malware's code for patterns or signatures in the malware itself and the files it downloads during its execution, attempting to correlate the malware with previously known variants. Furthermore, sandboxes can enhance the analysis of malware leveraging *Zero-Day* exploits by providing an isolated environment for safe execution while enabling behavior monitoring through low-level data analysis [5, 6]. Additionally, they improve *malware analysis pipelines* [7, 8, 9], commonly used for preliminary assessments and threat-level evaluations, as demonstrated by tools such as *DRAKVUF* and *Noriben* (explained in Sect. 3.1).

The results of this study develop along two different lines. In the first line, we collect existing malware sandbox proposals from both the scientific literature and Internet projects. What we found can be grouped into two categories: *local* sandboxes and *online* sandboxes, with distinct characteristics and limitations between the categories and within each category. As a first result, we identify these features (for example, the analysis of network traffic, the post-mortem state, or the possibility of

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT
stefano.bistarelli@unipg.it (S. Bistarelli); emanueleburberi18@gmail.com (E. P. Burberi); francesco.santini@unipg.it (F. Santini)

capturing a memory dump) and compare the existing proposals based on them. Secondly, we extend this classification from two to four categories, outlining the architecture of two new sandbox proposals that can offer greater benefits in terms of scalability and security. These new proposals are not currently available as standalone products, although they can be assembled using existing solutions.

By matching the technical foundations of these categories with three distinct *use cases*, we aim to guide the selection of the most suitable sandbox for different application needs, facilitating informed decision-making for specific desired requirements.

The remainder of this article is structured as follows. Section 2 presents the related work. In Section 3, we provide a comprehensive list of all available sandboxes. Next, in Section 4, we analyze and compare the sandboxes, defining various use cases based on the characteristics examined. Lastly, Section 5 wraps up the paper with conclusions and possible future work.

2. Related Work

Several authors have contributed significant research to the literature on malware analysis tools, particularly in comparing sandboxes. In [10], the differences between the *Cuckoo* and *DRAKVUF* sandboxes are analyzed, with the *DRAKVUF Sandbox* used to assess the latter, aiming to determine the optimal solution from the perspective of the user. The work in [11] conducts a similar comparative analysis, where four malware analysis solutions are evaluated, including *static analysis tools*, *Cuckoo/Malwr*, *Anubis*[1], and an unnamed *commercial* solution due to privacy and security reasons. The authors in [12] provide a comparison of three solutions: *Cuckoo*, *Any.Run*, and *Intezer Analyze*[2]. In contrast, [13] examines the different approaches between *Cuckoo* and *VMRay*[3] sandboxes, concluding that the agent-less design of the latter delivers better results.

In a Ph.D. thesis work [14], the author focuses on comparing sandboxes available for *Linux* based malware analysis, including *REMnux*[4], *Limon*, *Cuckoo*, *Detux*, and *HaboMalhunter*[5]. A similar approach is taken in [15], although the focus is on *Android* malware and the challenges specific to that domain. In a master thesis [16], the authors explore the potential of a container-based sandbox approach, leveraging container virtualization via a lightweight virtual machine (VM), resulting in the development of *SecBox*[6]. Furthermore, [17] provides a detailed comparison of available sandboxes and tools, focusing on techniques such as *Information Flow Tracking* (IFT) and *Function Call Monitoring*, highlighting which techniques are used by each sandbox.

In [18], malware analysis techniques are investigated similarly, as discussed in [17], but instead of a direct comparison, the key features of the methods are listed without further evaluation.

Both [3] and [1] similarly investigate the tools available for various types of malware analysis, including sandboxes, network analysis tools, binary/static analysis tools, and online services. Both papers provide neutral comparisons using different metrics, with extensive tables and data highlighting the advantages and disadvantages of each tool. However, [3] also proposes a heuristic-based malware detection pipeline based on the tools discussed.

The work in [8] describes the structure of signature-based and behavioral malware analysis pipelines, providing a literature review and describing how these methods can be implemented. The author of [9] explores malware analysis in *IoT environments* using existing tools such as *Cuckoo*. After reviewing the strengths and weaknesses of these tools for *IoT* malware analysis, the paper introduces a custom sandbox, *LiSa*[7], which adopts *QEMU* for virtualization, *SystemTap*[8] for dynamic analysis, and a custom *C++* program for network analysis. Moreover, this sandbox pipeline is deployable as a *Docker* image.

[1] This sandbox has been discontinued and is currently unavailable.
[2] https://analyze.intezer.com.
[3] https://www.vmray.com.
[4] https://remnux.org.
[5] https://github.com/Tencent/HaboMalHunter.
[6] https://github.com/SaltySpatula/SecBox
[7] https://github.com/danielpoliakov/lisa
[8] https://github.com/groleo/systemtap

3. Collecting Malware Sandboxes

The investigation of information concerning sandboxes was conducted primarily through *Google Scholar*, utilizing specific parameters to ensure comprehensive coverage. Keywords were selected to achieve a balanced and thorough exploration, employing generalized Boolean search formulas such as: `("malware analysis" AND "sandbox")`, `("sandbox" AND ("comparison" OR "dynamic analysis" OR "sandboxing techniques"))`, and `("zero day" AND ("malware analysis" OR "malware") AND ("sandbox" OR "sandboxing techniques"))`. These formulas were designed to encompass key concepts, including sandboxing techniques, dynamic analysis, comparisons, and their relationships with malware analysis and zero-day vulnerabilities. Furthermore, only publications dated 2014 onward were considered, ensuring the inclusion of recent advances and developments in the field. Following this initial review of previous comparative studies, an additional search was conducted using *Google* with the same set of keywords. This step identified and incorporated relevant sandboxes not referenced in the earlier academic literature review.

In this section, we list all available malware-sandbox solutions, categorized into four categories: *local-based* sandboxes (Sect. 3.1), *online-based* sandboxes (Sect. 3.2), *container-based* sandboxing (Sect. 3.3) and *hypervisor-based* sandboxes (Sect. 3.4). Each category starts with explaining the primary goals and advantages of the sandboxes within that category to clarify what can be expected from them.

3.1. Local Sandboxes

These open-source sandboxes are designed to run locally on the analysis machine, typically in conjunction with a virtual machine (VM) or hypervisor. Such solutions generally offer the capability to manage both the sandboxed environment and the internal or external tools, either through the terminal when used as a CLI application or via APIs commonly exposed for Web-based dashboards and control panels.

- **Cuckoo**: *Cuckoo*[9] is an open-source sandbox environment designed for advanced malware analysis. While the main repository has been archived, nevertheless, the project has been forked by the *Estonian CyberSecurity Agency*[10], and currently developed as *version 3.x*[11]. Thanks to its modular architecture, *Cuckoo* allows for extensive customization in all aspects, including the selection of preferred virtual machines or tools (such as *Yara*[12] and *Volatility*[13]), as well as the option to enhance its functionalities by integrating external modules for additional analysis. Additionally, it can be used through the *Malwr*[14] website, which utilizes *Cuckoo* to provide malware analysis services.
- **SEE**: *SEE*[15] (*Sandboxed Execution Environment*) is an open-source, event-driven, plugin-based sandbox provider designed for building test automation within secured environments. It offers basic support for *QEMU*, *VirtualBox*[16], and *LXC*[17], and since it is built on top of *libvirt's* APIs, it can be extended to support any hypervisor supported by *libvirt*[18]. *SEE* employs a system of pluggable providers to retrieve disk images from arbitrary sources and make them available in the sandboxed environment. By default, it supports providers for *libvirt* storage pools and *OpenStack Glance*, with the flexibility to add more image providers as needed. Internally, *SEE* is divided into two core components: the *SEE Environment*, which encapsulates all required resources and acts as a handler, and the *SEE Sandbox*, which is managed by predefined plugins (also known as

[9] https://cuckoosandbox.org.
[10] https://www.cert.ee.
[11] https://github.com/cert-ee/cuckoo3.
[12] https://github.com/VirusTotal/yara.
[13] https://github.com/volatilityfoundation/volatility.
[14] https://www.malwr.ee.
[15] https://github.com/WithSecureOpenSource/see.
[16] https://www.virtualbox.org/
[17] https://linuxcontainers.org/lxc/introduction/
[18] https://libvirt.org

hooks). These plugins communicate and coordinate through internal events. Additional plugins can be added to the predefined ones and configured via a JSON configuration file.

- **DRAKVUF**: *DRAKVUF*[19] is an open-source, virtualization-based, agentless black-box binary analysis system. Its architecture leverages the open-source *Xen*[20] hypervisor to provide a stealthy tool for malware analysis. *DRAKVUF* also supports the integration of external tools beyond those already included, such as *Volatility*. In addition to this project, the *Polish CyberSecurity Agency*[21] has developed a ready-to-use solution based on *DRAKVUF*. The *DRAKVUF Sandbox*[22] is designed to offer a flexible approach to malware analysis, featuring a *REST API* along with a user-friendly *Web UI*, and facilitating further integration with the *Karton*[23] pipeline. Karton is a versatile framework that simplifies building lightweight malware analysis backends and streamlining their integration into a reliable pipeline.
- **Detux**: *Detux*[24] is an open-source sandbox designed for the traffic analysis of Linux malware and the capture of *Indicators of Compromise* (IoCs)[25] by utilizing *dumpcap* and *Detux's* scripts. It operates on top of the open-source virtual machine *QEMU*[26], providing prebuilt *QEMU* images for ease of use. *Detux* can execute several static analysis tasks, such as basic string extraction from binaries, reading ELF information using the *readelf* command, and allowing the configuration of the report script to incorporate additional third-party commands for malware analysis. Conversely, for dynamic analysis, it can capture *pcaps* and extract IoCs and readable information after parsing them with *DPKT*. Although this sandbox provides a sufficient degree of customization, its emphasis on traffic analysis limits the ability to extend it with external tools for dynamic analysis.
- **Limon**: *Limon*[27] is an open-source sandbox designed for analyzing malware within a Linux environment. It enables the inspection of Linux malware before, during, and after execution, allowing for *Post Mortem analysis* —the analysis of the machine state after it is shutdown— and supports static and dynamic analysis utilizing open-source tools. Limon conducts its analysis in a controlled environment, built on top of the *VMWare*[28] hypervisor while monitoring the activities of the malware and its child processes to ascertain the nature and purpose of the malware. It can determine the malware process tree, interactions with the file system, and network activity, in addition to providing the capability to perform memory analysis, with the analyzed artifacts being stored for further investigation. However, despite being open-source, *Limon* does not provide the option to extend its functionality with external tools.
- **Noriben**: *Noriben*[29] is a Python-based script that works alongside *Sysinternals Procmon* to automatically collect, analyze, and report runtime indicators of malware within a virtual environment, which must be installed separately. To mitigate the absence of a complete sandboxed environment, *Noriben* includes a Python script (*NoribenSandbox.py*) to manage virtual environments. This script automates the execution of *Noriben* within a guest VM and retrieves the resulting reports, although it currently runs only on *OSX*. *Noriben* also provides various customization options, such as defining custom scripts to automate the malware analysis process, utilizing *Yara* signature files to scan each file created against those signatures, and integrating with the *VirusTotal* API to automatically submit *MD5* file hashes and retrieve the number of viral results.

[19] https://drakvuf.com.
[20] https://github.com/xen-project/xen.
[21] https://cert.pl.
[22] https://github.com/CERT-Polska/drakvuf-sandbox.
[23] https://github.com/CERT-Polska/karton.
[24] https://github.com/detuxsandbox/detux.
[25] IoCs are signals of malicious activity, such as a sudden increase in network traffic.
[26] https://www.qemu.org.
[27] https://github.com/monnappa22/Limon.
[28] https://www.vmware.com.
[29] https://github.com/Rurik/Noriben.

3.2. Online Sandboxes

The following sandboxes are accessible online and can be used through the website or public API. Similarly to locally-based sandboxes, they provide the ability to manage the environment and the tools involved, using either the API or the Web Interface, with some sandboxes even allowing live direct interaction with the system. However, it is essential to note that these closed-source solutions can be provided in three forms: as a paid product, such as *Kaspersky Threat Analysis*; free with premium paid features, as with *Any Run*; and entirely free, such as *Triage* and *Hybrid Analysis*. Additionally, those offered as a free service typically log every analysis conducted, including the malicious application uploaded and any files or information generated during execution as a by-product of the malware.

- **Triage**: *Triage*[30] is an online sandbox environment service provided by *Hatching*[31]. This service is accessible to the public via a free API, allowing users to upload malware samples and retrieve the corresponding reports. Additionally, the website offers the capability to interact directly with the sandbox. During the static analysis phase, it executes activities such as archive extraction, running heuristics, checking signatures using the provided *Yara* rules, and extracting the malware's configuration for classification purposes. Conversely, behavioral analysis is conducted by gathering data about the sample during its execution, including registry events, file operations (such as modification and writing), network requests, and memory dumps. However, a notable drawback is that it operates as a closed-environment solution, limiting customization options to those offered by the configuration parameters.
- **Kaspersky Threat Analysis**: *Kaspersky Threat Analysis*[32] is a comprehensive suite of tools for malware analysis, featuring a robust sandbox equipped with anti-evasion techniques that enables the analysis of process memory, network activity, and more. This allows for the investigation of an object's origins, the collection of IoCs based on behavioral analysis, and the detection of malicious entities. The solution provides extended logging and detailed reporting, which effectively uncover the malicious nature of a file while ensuring that the sandboxed environment remains undetected. Additionally, it allows customization of guest OS images to meet user-specific requirements. However, Kaspersky Threat Analysis is a paid service with two deployment options: cloud-based or *on-premise*, within the organization's IT infrastructure.
- **Any Run**: *Any Run*[33] is an online sandbox that focuses exclusively on behavioral malware analysis, allowing users to interact with the sandbox in real time to influence the malware's behavior. The final report visualizes the attack pattern through an interactive tree structure, facilitating comprehension of the core malicious processes. It also includes details on the types of files launched (such as browser sessions, script interpreters, and office applications), the family of malicious activity (if identified), the injection path, any files downloaded or dropped by the malware, and a complete network activity dump available as *pcap* files. The reports generated by *Any Run* are customizable and can be exported as a JSON file, along with screenshots and process behavior graphs. This solution is a paid service, with the option to create a free account with limited functionality. *Any Run* is closed-source, preventing the integration of external tools.
- **Hybrid Analysis**: *Hybrid Analysis*[34] is an online sandbox that provides a free API for submitting files and retrieving reports from malware analysis. The sandbox is powered by the *CrowdStrike Falcon Sandbox*, a high-end malware analysis framework capable of functioning as a large-scale system for automating the processing of numerous files. *Hybrid Analysis* combines runtime data with extensive static analysis, examining memory dumps to extract annotated disassembly listings, including strings and API call chains, along with decrypted SSL traffic (for Windows analysis). It also extracts behavior indicators aiding in the detection of unknown threats. For behavioral analysis, it offers various input file parser scripts that extract information from various

[30] https://tria.ge/docs/.
[31] https://hatching.io/triage/.
[32] https://www.kaspersky.co.uk/enterprise-security/threat-analysis.
[33] https://app.any.run.
[34] https://www.hybrid-analysis.com/docs/api/v2.

file types that can be integrated with *Yara* signature matches for any input and extracted files, and the possibility to emulate user-input actions (such as opening a browser, moving the mouse, etc.) by using *action scripts*.

3.3. Container Sandboxing

Container Sandboxing adds a layer of protection to containers. Although direct communication with the kernel via *system calls* significantly optimizes execution speed, it also exposes the container to vulnerabilities such as *kernel exploitation*. To address this risk, *container sandboxing* techniques are employed to enhance container security.

- **Kata Container**: *Kata Container*[35] is an open-source container sandboxing framework that encapsulates containers within lightweight virtual machines (VMs), delivering the benefits of isolation and security to the containers. It is compatible with various architectures and hypervisors, including their own *dragonball*[36] hypervisor. Since *Kata Container* utilizes an agent to manage the virtual environment, it does not require a custom-built hypervisor. Additionally, it can integrate with *CI* environments (through an optional component), offering the capability to construct automated malware analysis pipelines. Furthermore, it supports integration with containerized application managers such as *Docker*[37], *Kubernetes*[38], and other cloud-native systems.
- **Firecracker**: *Firecracker*[39] is an open-source virtualization technology developed by *Amazon*, specifically designed to create and manage secure, multi-tenant container and function-based services. Firecracker's *microVM* implements a minimal device model, loading only the essential components to operate, thereby reducing the attack surface and minimizing startup time. The entire process can be managed via a RESTful API, which supports operations such as configuring the number of vCPUs and starting the virtual machine. Firecracker further strengthens isolation by employing a tool called *jailer*[40], which is designed to isolate the Firecracker process, serving as an additional layer of defense if the virtualization boundary is breached. Similar to *Kata Container*, Firecracker is capable of integrating with various containerized application managers and is compatible with *AWS* cloud-based services.
- **Quark**: *Quark*[41] is an open-source, high-performance, and secure container runtime. It delivers virtual machine-level workload isolation and security while also ensuring efficient execution of container workloads. *Quark* is built upon two tightly integrated modules: a hypervisor (*QVisor*) and a guest kernel (*QKernel*). Due to this architecture, *Quark* does not support other kernels. Additionally, *Quark* supports the *TCP Socket over Remote-DMA* (*TSoR*) architecture, initially defined in [19], which enables TCP sockets to operate over remote DMA (*RDMA*[42]) connections. *TSoR* allows existing TCP-based container applications to transfer data via RDMA without any modifications. By offloading the TCP/IP protocol stack to RDMA, *TSoR* achieves higher throughput and lower latency. Finally, *Quark* can host multiple *Docker* containers within a single *Quark* container, and it offers integration capabilities with cloud-based container solution providers.
- **gVisor**: *gVisor*[43] is an open-source *application kernel* for containers, developed by *Google*, which implements a *Linux-like* interface, offering a robust layer of isolation between running applications and the host operating system, while functioning in *userspace*. It restricts the access of container applications to the host kernel's surface while providing them with all essential features. Despite the security features it provides, gVisor should not be confused with tools or technologies designed

[35] https://github.com/kata-containers/kata-containers.
[36] https://github.com/kata-containers/kata-containers/tree/main/src/dragonball
[37] https://www.docker.com.
[38] https://kubernetes.io.
[39] https://firecracker-microvm.github.io.
[40] https://github.com/firecracker-microvm/firecracker/blob/main/docs/jailer.md
[41] https://github.com/QuarkContainer/Quark.
[42] https://www.rdmaconsortium.org
[43] https://github.com/google/gvisor.

to harden containers against external threats, offer additional integrity checks, or restrict access to services. Similar to other container sandboxing solutions, *gVisor* can integrate with existing container tools like *Kubernetes*, simplifying the process of running sandboxed containers.

3.4. Hypervisor based Sandboxes

Hypervisor-based sandboxes leverage the hypervisor's capability to manage the overlaid system, providing a more secure environment by effectively isolating the Guest system from the underlying Host Kernel. In Linux-based operating systems, it is possible to utilize the *KVM*[44] module provided by the *Linux kernel* to execute sandboxes at the same level as the kernel. To mitigate possible vulnerabilities within the hypervisor's implementation, it is generally advisable to harden the virtualized system and the hypervisor itself and, if needed, to employ external tools or specialized hypervisors to monitor and prevent any potentially malicious actions.

- **Alcatraz**: *Alcatraz*[45], although being an actual hypervisor as *KVM*, it can still be classified as a *KVM-based* sandbox, since it is proposed as an underlying layer for the KVM architecture. As it is laid under the *KVM* layer, it provides a secure environment to run virtual machines, elevating the KVM layer at *Ring 0*[46], preventing any malware from potentially subverting the system. This allows *Alcatraz* to be executed as a sandboxing environment, making it suitable for custom solutions and for enhancing the security of containerized sandboxing solutions, as it integrates natively with solutions that leverage the KVM hypervisor.
- **QEMU**: *QEMU*[26] is a virtual machine based on top of the *KVM* hypervisor. Maybe not a secure environment in general, but it is possible to enhance the security level by enabling the *seccomp* option. Even if this measure is fully functional, it is still not ideal for malware analysis, as it would still be subject to the threat of malware that escapes the VM.
- **KVMSandbox**: *KVMSandbox* [20] is an *application-level* sandbox with *x86 hardware virtualization* and *KVM*. It offers a secure environment to run malware, as the malware is executed inside a guest environment, detached from the Host's kernel and memory; furthermore, any syscall made is controlled and validated by the sandbox. Although the original paper provides an overview and implementation details, it omits the actual implementation of *KVMSandbox*.

4. Sandbox Comparison

This section determines the key differences between the sandboxes mentioned above. To achieve this, it is necessary to establish a shared basis for evaluation. The adopted solution is to compare the characteristics and features each sandbox offers to ensure the most neutral evaluation while also identifying the use cases for each sandbox. For reference, Appx. A provides figures relative to the architectural designs defined in the following use cases. Appendix B instead provides two tables: Table 1, which details the *extractable features* a sandbox can provide in its report, including *Logging*, *Process Tree/List*, *Memory Dump*, *Dropped Files*, *Network Traffic Information*, *Signatures Check*, and *Post-Mortem State*—and Tab. 2, which describes the typical characteristics of various sandboxes in a tabular format—such as *Sandbox Type*, *File Type Support*, *Supported OS*, *Last Update*, *Analysis Tools Included*, *External Analysis Tools Support*, *Feature Extraction Capabilities*, and *Configurability*. After we have listed and analyzed all the characteristics of sandboxes, we can now compare them. We distinguish three *use cases*: *Ready-to-Use Sandbox*, *Security-Enhanced Sandbox* and *Deployable Sandbox*.

[44] https://linux-kvm.org/.
[45] https://github.com/kkamagui/alcatraz.
[46] *Protection Rings* are hierarchical privilege levels in CPU architecture that manage access to system resources. Ring 0, the most privileged, grants full access to hardware and is where the kernel operates. Higher rings, where applications run, restrict access to protect critical resources.

Ready-to-Use Sandbox. If the user needs to *quickly* set up a malware analysis, perhaps shortly after identifying a potential *threat*, or prefers to avoid investing significant time in *fine-tuning* a malware analysis framework, the optimal choice would likely be a *Local* (see Sect. 3.1) or *Online* (see Sect. 3.2) sandbox. Both options offer a user-friendly interface coupled with advanced malware analysis capabilities; however, the latter may be preferred, as it can be quickly made operational even *more*.

Although these types of sandboxes are usually bounded by a virtual machine, like *QEMU* used by *Cuckoo*, and VMs based on the *Xen* hypervisor used by *DRAKVUF*, they can differ in how they implement the introspection module. In this regard, two approaches exist: *agent-based* (Fig. 1 in Appx. A) and *agent-less* (Fig. 2 in Appx. A). The first approach provides an external module (*agent*) that runs inside the sandboxed environment collecting the data and communicating with the *sandbox engine* running on the *host*. The latter is part of the *sandbox engine*, retrieving the data using a set of APIs the hypervisor exposes. Although minimal, this difference can potentially affect the behavior of the malware, as an agent-based design could compromise the stealthiness of the sandbox, hence giving the malware the possibility to detect it and consequently change its behavior.

At this point, we can compare these *ready-to-use* sandboxes. For *local* sandboxes like *DRAKVUF* and *Cuckoo*, the key difference lies in their underlying *architectures*. The *agentless* architecture, as seen in *DRAKVUF*, offers a *stealthier* solution, whereas the *agent-based* architecture, like *Cuckoo's*, allows for a *deeper* analysis and data collection since it is not limited by the *API* restrictions exposed by the *hypervisor*. Both *DRAKVUF* and *Cuckoo* are superior to other sandboxes like *See*, *Detux*, *Noriben*, and *Limon*, even though the latter, differently from the other sandboxes, supports *Post Mortem State* analysis.

As for online sandboxes, *Triage* and the *Hybrid Analysis* stand out as the best options for malware analysis, offering *more* features than other online alternatives and providing access to *free* API. While both sandboxes allow for a reasonable degree of customization and API usage, *Triage* excels by offering integration with *Yara* and enabling direct interaction with the sandbox.

Security-Enhanced Sandbox. If the main need is a *sealed environment*, then the best solution is a *hypervisor-based* (see Sect. 3.4) sandbox. While it requires more time to build and configure, the wide range of customization, due to the lack of a malware analysis framework, makes this a better solution, *also*, for those who may need a custom set of features.

For this kind of sandboxes, two types of architectures are used: *User Space* (Fig. 3 in Appx. A) based and *Hypervisor* based sandboxes (Fig. 4 in Appx. A). The first approach leverages the ability of the *KVM* kernel module to create a virtual process isolated from the rest of the system. The latter employs an external hypervisor to control the isolation of the VM, as this hypervisor will observe and then analyze every syscall execution request that exits the sandboxed environment. Both of these architectures communicate with the sandboxed environment using the API provided by *KVM* or any other hypervisor managing the *VM*, such as *KVM/QEMU* or *Xen*, to analyze the state of the virtualized system. Hence, it is essential to use an *introspector*, which employs *VMI* (*Virtual Machine Introspection*) to interact with the virtualized environment.

The User Space architecture can be used to create a virtualized environment or to use a virtual machine in User Space, such as *QEMU*. This approach offers *lower* overhead but requires a *custom-built* sandbox to interface with the *KVM* hypervisor and create the virtual environment. Conversely, a *Hypervisor* based sandbox is *easier* to implement but *less* efficient, as it involves creating a *complete* virtual machine. Furthermore, the first approach may introduce overhead, as it *continuously* operates in the background as a User Space program, while the second approach *could stop* the execution of necessary tools, potentially blocking them by *mistakenly* identifying them as threats.

Deployable Sandboxes. If there is no need to *deeply* analyze any malware, while *enhancing* the security level, and *maintaining* the scalability and deployability of *container-based* systems, the best solution may be a *containerized* (see Sect. 3.3) sandbox.

Although this technique improves the security of *containerized applications*, its main advantage lies in its *deployability*, as the level of security *may not be adequate* to counter malware capable of exploiting

vulnerabilities in the mechanisms designed to secure the containerized environment. However, it can be seen as a middle ground between a *ready-to-use* solution and a *security-enhanced* approach.

Currently, two types of architecture are used: *VM* based (see Fig. 5 in Appx. A) and *Application Kernel* based (see Fig. 6 in Appx. A) containerized sandboxing. The first solution encapsulates the containerized application within a *light-weight VM*. The latter encapsulates containerized applications within a *User Space kernel*, limiting direct interaction with the *host kernel*. Whereas the first architecture provides enhanced isolation through a *lightweight VM*—albeit with *increased* overhead—the second architecture is *more* efficient, demanding fewer resources; however, it requires additional resources for malware analysis. As a result, the first is better suited for a *scalable* malware analysis framework, such as a *cloud based* solution, while the second is ideal for *high-performance*, *parallel* in-depth analysis on a *local server or cluster*, where one external introspector can monitor *multiple* malicious applications running *concurrently* inside their containers. At the same time, it is necessary to note that each sandbox solution that employs a *VM* based architecture (like *Quark* and *Firecracker*), can also differ in the implementation of the virtual machine. Unlikely, *gVisor* is the only containerized sandbox that employs the other architecture, acting as an *application kernel*.

5. Conclusion

This paper comprehensively analyzes various sandbox solutions by extensively comparing their features. We classify sandboxes into four categories, the first two of which consist of currently available solutions. We then analyze and compare these classes in detail. In addition, we introduce two novel architectural approaches that incorporate new isolation techniques.

Ultimately, we define three application use cases, refining the four aforementioned categories to facilitate the selection of the most suitable sandbox for each specific scenario.

Our findings indicate that among currently available sandboxes—categorized under the *ready-to-use* case—those leveraging an *agent-less* architecture represent the state-of-the-art. However, in the context of future sandbox development, we determine that the core features of the proposed novel architectures, if combined effectively, could significantly enhance the next generation of sandboxes, potentially surpassing existing state-of-the-art solutions in terms of isolation (i.e., security) and scalability.

Furthermore, future sandbox designs should explore *i)* incorporating *emulated user behavior* to better disguise the execution environment from malware, *ii)* implementing algorithms to counteract *time-based* detection techniques, which adversaries exploit by analyzing execution time anomalies in virtualized environments, and *iii)* optimizing analysis time to prevent malware from delaying malicious activity until after sandbox inspection is complete.

Acknowledgments

This research was supported by MUR PNRR project SERICS (PE00000014) AQuSDIT (CUP H73C2200088 0001) and COVERT (CUP (J93C23002310006)), funded by the European Union – Next Generation EU, and Project VITALITY (CUP J97G22000170005), funded by the European Union – Next Generation EU.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT and Writefull in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] S. Pandey, B. Mehtre, Performance of malware detection tools: A comparison, in: 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, IEEE, 2014, pp. 1811–1817.

[2] R. Sihwail, K. Omar, K. Ariffin, A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis, Int. J. Adv. Sci. Eng. Inf. Technol 8 (2018) 1662–1671.

[3] Ö. Aslan, R. Samet, Investigation of possibilities to detect malware using existing tools, in: 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), IEEE, 2017, pp. 1277–1284.

[4] J. Juwono, C. Lim, A. Erwin, A comparative study of behavior analysis sandboxes in malware detection, in: International Conference on New Media (CONMEDIA), 2015, p. 73.

[5] F. Alhaidari, N. Shaib, M. Alsafi, H. Alharbi, M. Alawami, R. Aljindan, A. Rahman, R. Zagrouba, Ze-vigilante: Detecting zero-day malware using machine learning and sandboxing analysis techniques, Computational Intelligence and Neuroscience 2022 (2022) 1615528.

[6] H. Al-Rushdan, M. Shurman, S. Alnabelsi, On detection and prevention of zero-day attack using cuckoo sandbox in software-defined networks., Int. Arab J. Inf. Technol. 17 (2020) 662–670.

[7] E. Gandotra, D. Bansal, S. Sofat, Zero-day malware detection, in: 2016 Sixth international symposium on embedded computing and system design (ISED), IEEE, 2016, pp. 171–175.

[8] M. Goyal, R. Kumar, The pipeline process of signature-based and behavior-based malware detection, in: 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA), IEEE, 2020, pp. 497–502.

[9] D. Uhrıcek, Lisa–multiplatform linux sandbox for analyzing iot malware, 2020.

[10] S. Ilić, M. Gnjatović, B. Popović, N. Maček, A pilot comparative analysis of the cuckoo and drakvuf sandboxes: An end-user perspective, Vojnotehnički glasnik/Military Technical Courier 70 (2022) 372–392.

[11] M. Mehra, D. Pandey, Event triggered malware: A new challenge to sandboxing, in: 2015 Annual IEEE India Conference (INDICON), IEEE, 2015, pp. 1–6.

[12] A. Agrawal, et al., A comparative analysis of open source automated malware tools, in: 2022 9th International Conference on Computing for Sustainable Global Development (INDIACom), IEEE, 2022, pp. 226–230.

[13] M. Ali, S. Shiaeles, M. Papadaki, B. Ghita, Agent-based vs agent-less sandbox for dynamic behavioral analysis, in: 2018 Global Information Infrastructure and Networking Symposium (GIIS), IEEE, 2018, pp. 1–5.

[14] O. Olowoyeye, Evaluating Open Source Malware Sandboxes with Linux malware, Ph.D. thesis, Auckland University of Technology, 2018.

[15] S. Neuner, V. Van der Veen, M. Lindorfer, M. Huber, G. Merzdovnik, M. Mulazzani, E. Weippl, Enter sandbox: Android sandbox comparison, arXiv preprint arXiv:1410.7749 (2014).

[16] R. Mogicato, A. Zermin, Design and implementation of a collaborative lightweight malware analysis sandbox using container virtualization, Universität Zürich, Zürich, Switzerland, Tech. Rep. (2023).

[17] W. Aman, A framework for analysis and comparison of dynamic malware analysis tools, arXiv preprint arXiv:1410.2131 (2014).

[18] N. Kaur, A. Bindal, A complete dynamic malware analysis, International Journal of Computer Applications 135 (2016) 20–25.

[19] S. Yulin, Q. Qingming, Z. Chenxingyu, K. Arvind, C. Hong, X. Ying, Tsor: Tcp socket over rdma container network for cloud native computing, 2023. URL: https://arxiv.org/abs/2305.10621. arXiv:2305.10621.

[20] A. Ayer, Kvmsandbox: Application-level sandboxing with x86 hardware virtualization and kvm, 2012.

A. Architectural Diagrams

This appendix provides a visual representation of the architectures associated with the categories defined in Sect. 4, illustrating the key structural characteristics of each category through figures.

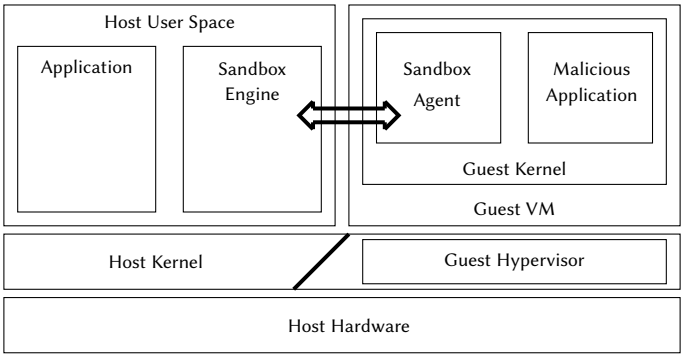

Figure 1: *Agent* based Sandbox Architecture.

Figure 1 presents the possible implementation of a sandbox that uses an agent for VM Introspection, as highlighted by the arrow that connects the sandbox engine directly to the sandbox agent.

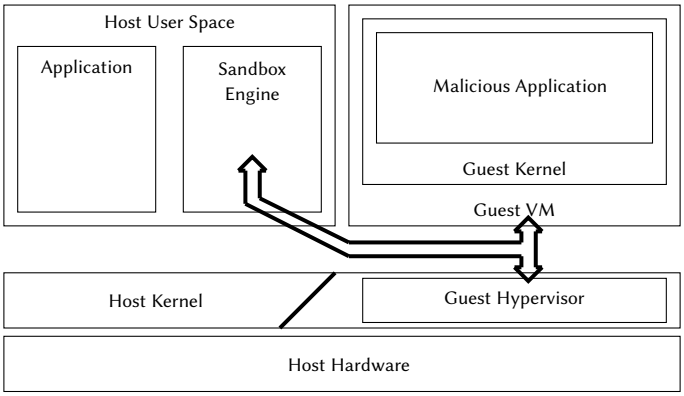

Figure 2: *Agent-less* Sandbox Architecture.

Figure 2 shows the architecture of a sandbox that takes advantage of an agent-less paradigm. As the arrows indicate, the sandbox engine interacts directly with the guest VM, via the guest hypervisor.

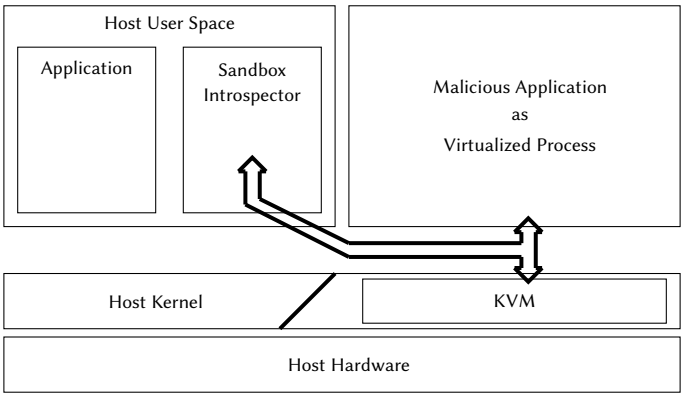

Figure 3: *User Space* based Sandbox Architecture.

Figure 3 shows how user-space sandboxing techniques, such as *KVMSandbox*, execute the process as a virtual process, which can then be inspected using a sandbox inspector, communicating with the

virtual environment, through the hypervisor or other software used to create the virtualized process, in this case KVM.

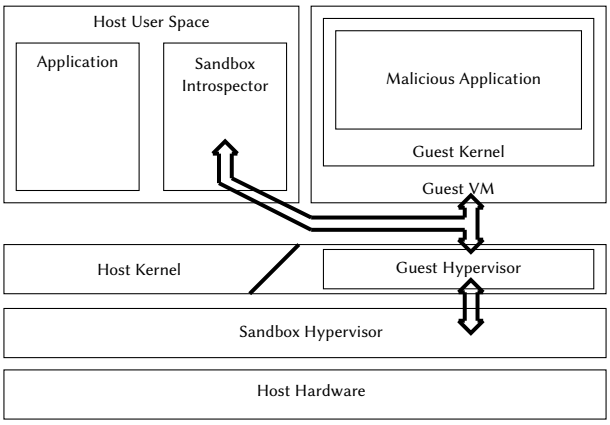

Figure 4: *Hypervisor* based Sandbox Architecture.

Figure 4 presents how a hypervisor-based sandbox can be implemented. The malicious application is executed within a guest VM that is managed by a guest hypervisor that runs over the sandbox hypervisor. The sandbox can leverage the VM Introspection feature offered by the guest hypervisor to retrieve the information relative to the guest VM.

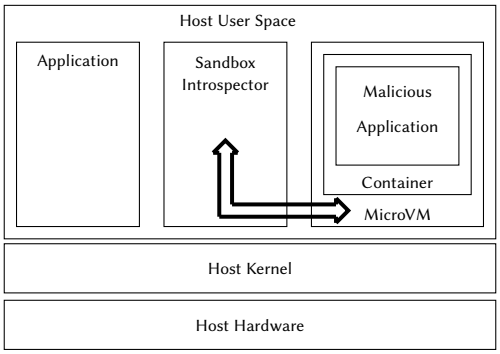

Figure 5: *VM* based Containerized Sandboxing Architecture.

Figure 5 shows how the architecture of a containerized sandbox, like Firecracker, can be implemented. As we can see, similarly to the previous figure of the hypervisor-based sandbox, this architecture leverages the VM Introspection feature offered by the VM.

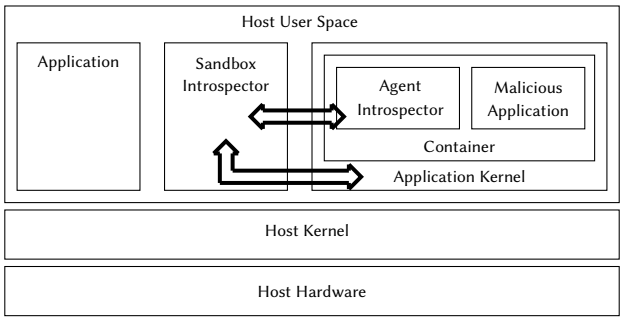

Figure 6: *Application Kernel* based Containerized Sandboxing Architecture.

Figure 6 presents the possible implementation of containerized sandboxes that employ an application kernel, like *gVisor*. The container is executed on top of the application kernel, which contains

the sandbox's agent for introspection, directly communicating with the sandbox introspector. Moreover, the sandbox introspector also communicates with the application kernel to retrieve low-level information.

B. Feature Comparison

We present a list of features that can be extracted from a sandbox report, which are used to build Tab. 1:

- **Logging**: This feature includes the logging of everything that happens inside the machine, from process creation to syscall events and file changes.
- **Process Tree/List**: The process tree/list of the machine during the analysis.
- **Memory Dump**: Information about the memory and registers content after a certain event (malware's actions/syscalls, etc.).
- **Dropped Files**: Files dropped by the malware onto the disk.
- **Network Traffic Information**: Information about network traffic (packets sent/received, etc.) during the analysis.
- **Signatures Check**: Hash signatures of the file system before and after changes in files during the analysis, and signatures of files dropped by malware. This feature can be further expanded by integrating it with Yara[12].
- **Post-Mortem State**: Snapshot of the state of the machine after shutdown.

Table 1
Extractable features table.

Sandbox Name	Logging	Process Tree/List	Memory Dump	Dropped Files	Network Traff. Inf.	Signatures Check	Post-Mortem State
Cuckoo	✓	✓	✓	✓	✓	✓	
DRAKVUF	✓	✓	✓		✓	✓	
Detux	✓	✓			✓		
Limon	✓	✓	✓		✓	✓	✓
Noriben	✓	✓			✓		
Triage	✓	✓	✓	✓	✓	✓	
Kaspersky Threat Analysis	✓	✓		✓	✓	✓	
Any Run	✓	✓		✓	✓	✓	
Hybrid Analysis	✓	✓	✓	✓	✓	✓	

In the following, we list the characteristics of the sandboxes that are used to create Tab. 2:

- **Sandbox Type**: whether it is *local*, *online*, *containerized* or *hypervisor-based*.
- **File Type Support**: type of files supported for malware analysis.
- **Supported OS**: which OSs are supported for running the sandbox.
- **Last Update**: last update of the sandbox, and if it's still maintained.
- **Feature Extraction Capabilities**: whether the sandbox can extract some features from the analysis (see Tab. 1).
- **Analysis Tools Included**: which analysis tools are included (like *Yara*[12], *Volatility*[13]).

- **External Analysis Tools Support**: whether the sandbox supports the installation of additional analysis tools.
- **Configurability**: whether the sandbox supports, maybe via a *configuration* file, the configuration of the sandbox and, eventually, of external analysis tools. For simplicity, we will use three levels of *configurability*:
  - **Limited**: for a *minimal* set of configurable parameters.
  - **Highly**: for an *extended* set of configurable parameters.
  - **Fully**: if it includes configurable parameters *also* for included and/or external tools.

Table 2
Sandbox characteristics table.

Sandbox Name	Sandbox Type	File Supported Type	Supported OS	Last Update	Feature Extraction Capabilities	Analysis Tools Included	External Analysis Tools Support	Configurability
Local Sandboxes								
Cuckoo	*local*	Windows and Linux Executables, MS Office, HTML, DLL, PDF, zip, bin, jar, .py, apk, etc.	Windows, Linux, Mac OS	Apr 27, 2021 (Archived)	See Tab. 1	Process Utility, Machine Utility, Cuckoo Router (INetSim)	Yara[12], Volatility[13], Community Developed Tools	*Fully*
SEE	*local*	*None*	Linux	Oct 5, 2020 (*unmaintained*)	*None*	*None*	Hooks for external tools	*Highly*
DRAKVUF	*local*	Windows and Linux Executables, DLL, MS Office, HTML	Windows, Linux	Sep 12, 2024 (*active*)	See Tab. 1	DRAKVUF's Plugins	Volatility and Procmon	*Limited*
Detux	*local*	All	Linux	Feb 16, 2018 (*unmaintained*)	See Tab. 1	Dumpcap	Support for 3rd party commands to analyze the binary	Highly
Limon	*local*	ELF, .pl, .py, shell and bash scripts, .PHP, LKM	Linux	Mar 25, 2016 (*unmaintained*)	See Tab. 1	Yara[12], Volatility[13], INetSim, etc.	*None*	*Fully*
Noriben	*local*	Requires *manual* interaction, hence every file Windows can support	OSX	Nov 29, 2023 (*unmaintained*)	See Tab. 1	Yara[12], ProcMon, md5deep[47]	*None*	*Highly*
Online Sandboxes								
Triage	*online*	Windows and Linux Executables, MS Office, HTML, DLL, PDF, zip, jar, apk, etc.	*cloud based*	*closed source*	See Tab. 1	Yara[12]	*None*	*Highly*
Kaspersky Threat Analysis	*online*	*200+, but are not specified*	*cloud based*	*closed source*	See Tab. 1	Yara[12], Kaspersky Analysis Tools	*None*	*Limited*
Any Run	*online*	Requires *manual* interaction, hence every file Windows and Linux can support	*cloud based*	*closed source*	See Tab. 1	*None*	*None*	*Limited*
Hybrid Analysis	*online*	Windows and Linux Executables, MS Office, HTML, DLL, PDF, zip, .ps1, jar, .py, .pl, etc.	*cloud based*	*closed source*	See Tab. 1	*None*	*None*	*Limited*
Container Sandboxing								
Kata Container	*containerized*	Requires *manual* interaction, hence every file any OS can support	*containerized*	Sep 14, 2024 (*active*)	*None*	*None*	*None*	*Limited*
Firecracker	*containerized*	Requires *manual* interaction, hence every file any OS can support	*containerized*	Sep 13, 2024 (*active*)	*None*	*None*	*None*	*Limited*
Quark	*containerized*	Requires *manual* interaction, hence every file any OS can support	*containerized*	Sep 1, 2024 (*active*)	Logging	*None*	*None*	*Limited*
gVisor	*containerized*	Requires *manual* interaction, hence every file any OS can support	*containerized*	Sep 14, 2024 (*active*)	*None*	*None*	*None*	*Limited*
Hypervisor based Sandboxes								
Alcatraz	*KVM-based*	Requires *manual* interaction, hence every file any OS can support	Linux	Aug 5, 2021 (*unmaintained*)	*None*	*None*	*None*	*Limited*
QEMU	*KVM-based*	Requires *manual* interaction, hence every file any OS can support	Linux	Sep 13, 2024 (*active*)	*None*	*None*	*None*	*Limited*
KVMSandbox	*KVM-based*	32-bit ELF	Linux	*unreleased*	*None*	*None*	*None*	*Limited*