DRAKVUF Malware Sandbox

Sukhjinder Singh

***Abstract*—A sandbox is a system that provides a safe environment for analysis and deployment of malware samples while all the activities of the malware are captured and logged. DRAKVUF is an open-source black box analysis sandbox based on the popular Cuckoo sandbox system. The current implementation of DRAKVUF has shortcomings in terms of the web user interface, which restricts user customisation of the analysis environment variables. This project will extend DRAKVUF's web user interface to support new capabilities such as, an operating system with various configurations and easy selection and assignment of environment variables that are currently hidden within the DRAKVUF configuration files or command line interface. The queuing system of DRAKVUF is also extended to allow for multiple samples to be analysed at once. The reporting component is significantly improved by implementing the capability to generate and send comprehensive reports to a given email address provided by the user during submission. The reporting interface also provides historical data on previously submitted malware samples.**

The improved DRAKVUF system is hosted on an ECS server and is accessible to staff and students to submit and analyse potentially malicious samples. The success of this project is determined through a successful deployment and fully operational malware sandbox of DRAKVUF sandbox instance and a developed web interface.

I. Introduction

MALWARE analysis is the process of examining a piece of malicious software (malware sample) to understand its behaviour and purpose. The information provided by this analysis can be used to develop defences against malware, such as antivirus signatures and firewalls [1].

There are two types of malware analysis: "static" and "dynamic". Static malware analysis is a technique that examines the code and properties of a malicious file without executing it. Dynamic malware analysis is any examination performed after executing malware [2]. This generally requires the malware sample to be executed in a system that is set up in a closed, isolated virtual environment so that it can be studied thoroughly without the risk to a production system. Malware analysed using the static analysis technique is much safer than using the dynamic analysis technique. Basic static analysis and advanced static analysis are the two steps of the static analysis method used to analyse malware [3]. On the other hand, basic dynamic malware analysis involves running malware samples in controlled environments, such as virtual machines, to observe their behaviour and identify malicious activities. Advanced dynamic malware analysis goes beyond basic observation by utilising techniques like code and memory analysis, network traffic monitoring, and behaviour profiling to gain deeper insights into the malware's capabilities, evasion techniques, and potential impact on a system or network. The efficacy of static analysis has been greatly reduced over the years due to the proliferation of metamorphic malware with malware authors using complex techniques such as anti-analysis to perform detection evasion [4].

Basic dynamic analysis and advanced dynamic analysis also have drawbacks; namely, the number of samples that need to be analysed by malware analysts. This puts a burden on hardware resources used to analyse the samples [4]. Modern malware can also leverage anti-analysis techniques for dynamic analysis. This is done by the detection of monitoring environments and hiding in unmonitored corners of the system [4].

Sandboxes are an advanced dynamic analysis method for running untrusted programs in a safe environment without fear of harming "real" systems. Sandboxes comprise of virtualised environments that often simulate operating systems and network services in some fashion to ensure that the software or malware being tested will function normally [2]. However, they do not offer the opportunity to customise how the malware samples should be evaluated. This leads to results being imposed due to restrictions on the choice of operating systems, installed packages, how long the execution will last, and many other things.

II. The Problem

Dynamic malware analysis requires a safe and isolated environment to run malware samples. This can be done in a variety of ways, sandboxes being a very safe way of execution and analysis [3]. However, sandboxes can be resource intensive, detectable, and complex to set up and operate [4].

DRAKVUF is an open-source software that provides a foundation for a malware analysis sandbox [5]. It provides a platform for stealthy malware analysis as its footprint is nearly undetectable from the malware's perspective [5]. DRAKVUF provides a basic web interface that gives users limited customisability on how the sandbox will run the malware samples and how reports will be delivered to users.

The current web interface for the DRAKVUF sandbox lacks essential features and functionalities, limiting its usability and hindering efficient malware analysis. Namely the interface does not support the selection of multiple virtual environments, it restricts the duration of malware sample execution, and it does not have a job management system, which will need to be implemented for managing submitted jobs. This deficiency inhibits effective management of DRAKVUF instances, impedes analysis of submitted jobs, and fails to provide a comprehensive solution for malware analysis.

The aim of this project is to extend the capabilities of the current DRAKVUF web interface by allowing users to customise their analysis environment and provide an easy-to-use interface for managing malware sample submission and

This project was supervised by Masood Mansoori and Lisa Patterson

queuing. When the malware sample is launched and evaluated, there will be functionality added to DRAKVUF to email the results to the user.

III. Project Objectives

A successful implementation and solution of DRAKVUF will meet these functional requirements:

- Be remotely accessible through a web interface.
- The system must show the available virtual machines and operating system and their respective configurations for the user to select.
- The system must allow management of multiple instances of DRAKVUF, including shutting down or restarting the instances, and managing the submitted jobs.
- Support multiple operating systems and versions such as Windows 7 and Windows 10.
- The system must allow configuration attributes of DRAKVUF to be altered and set through the web interface such as: which VM to execute malware on, analysis time, modules included in analysis.
- The system must allow submission of a file to be analysed by the operating systems above, through the web interface.
- The system should allow a user to download the reports generated by DRAKVUF or view them through the web interface.
- The system must allow a report to be emailed to a designated email address provided by the user.

And the following non-functional requirements:

- **Efficiency:** Concurrency of analysing many malware samples is to be maximised while performance overhead for analysing a single sample is to be minimised.
- **Stealth:** The monitored environment should not be able to detect the presence of DRAKVUF.
- **Isolation:** DRAKVUF should be isolated from the analysis virtual machines to protect against tampering.

IV. Background Research

There are several pieces of literature outlining the capabilities of the DRAKVUF sandbox from reputable sources such as Intel [4] and the Institute for System Programming of the Russian Academy of Sciences [6]. Both discuss the challenges that DRAKVUF solves around the anti-analysis techniques that malware employs to hide from malware analysis.

DRAKVUF uses the Xen hypervisor to get around the challenges that modern malwares present with anti-analysis techniques [4]. The Xen hypervisor offers novel techniques to eliminate blind-spots created by kernel-mode rootkits by extending the scope of monitoring to include kernel internal functions and to monitor file system accesses through the kernel's heap allocations. It also enables DRAKVUF to improve stealth by starting the execution of malware samples without leaving traces in the analysis machine [4]. This is a feature that other popular sandbox solutions like Cuckoo and CWSandbox do not possess.

DRAKVUF is implemented using the virtualisation technology on Intel Central Processing Units (CPUs). This provides an avenue to observe malware execution by allowing external access to the state of the virtualised hardware components. This technique is commonly referred to as Virtual Machine Introspection (VMI) [4]. To access the components externally DRAKVUF uses active VMI through breakpoint injection to hijack an arbitrary process within the VM to initiate the start of the malware sample [4]. Breakpoint injection is a technique used to inject a breakpoint into a running process. When this is done, the process will stop executing at the breakpoint and DRAKVUF will be able to modify it and continue the execution. By using existing processes running within the VM, DRAKVUF does not introduce any new code or artifact into the analysis VM, thus greatly improving stealthiness [4]. In contrast, the Cuckoo sandbox injects a Python script into the analysis VM which can be detected by malware.

DRAKVUF can also track changes and deletions to files within the VM [6]. This is done using a technique called shadow paging to track the changes made to files. Shadow paging creates a copy of each file in the sandbox and the copy is called the shadow page [6]. DRAKVUF can read the contents of the shadow pages and compare them with the original files to see what changes were made to the files. This is very helpful for malware analysts to determine how different types of malware work and help identify and track malware infections.

V. Related Work

There are a number of open-source sandboxes available to analyse malware samples. Many popular or defacto malware sandboxes have been analysed to weigh their pros and cons compared to DRAKVUF sandbox.

A. Cuckoo

Cuckoo is a sandbox technology that is widely used by researchers and malware analysts in the industry to carry out dynamic malware analysis. Cuckoo accomplishes this by creating a virtual environment like DRAKVUF, however, to analyse the malware, an in-guest (agent) module is required. Due to the metamorphic nature of many modern malwares, they have the potential to hide themselves or act benignly in the analysis environment if the agent is detected [7]. DRAKVUF solves this problem by utilising the breakpoint injection technique to execute the dynamic malware analysis [7].

The architecture diagram of Cuckoo is shown in Fig. 1. It consists of three main components, the cuckoo host machine, the virtual network, and the analysis guests. There is also the analysis manager and agent modules that are passed into the analysis guests. The user interacts with Cuckoo from the host machine either through a command-line interface or through a web interface like DRAKVUF. The analysis manager is responsible for submitting malware samples and collecting the reports after the malware analysis is completed [7]. The Cuckoo host interacts with the analysis virtual machines though the virtual network and an in-VM Python agent module is installed within the guest OS to start the analysis automatically when the malware sample is received.

The Python agent is responsible for receiving the malware sample, executing it, and sending the analysis report back to the analysis manager [7].

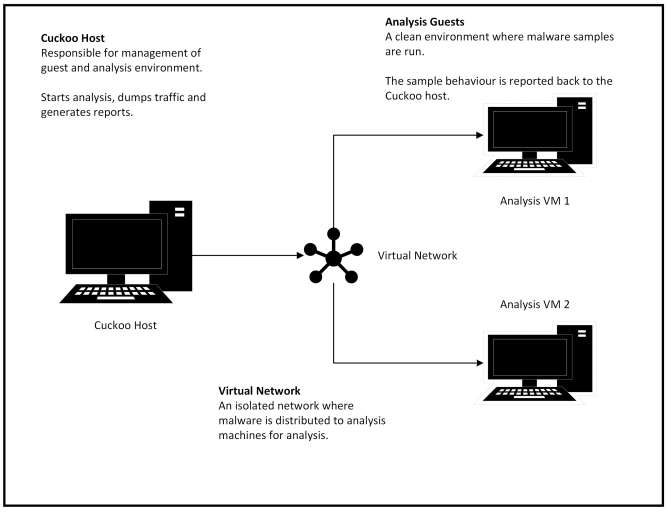


Fig. 1: Cuckoo Architecture Diagram [7]

Cuckoo and DRAKVUF were compared side by side in [7], and the superior sandbox solution is determined through a series of experiments. The criteria for determining which sandbox solution was best was:

1. Duration and throughput analysis
2. Space analysis

The first experiment that was conducted compared the duration of time that it took for Cuckoo and DRAKVUF to analyse malware samples and the throughput of each sandbox. Throughtput determines how many malware samples can be analysed by a system in a given amount of time. The experiment used a set of 40, 100 and 200 malware samples. These tests illustrated that DRAKVUF took less time than Cuckoo to analyse them and that DRAKVUF also had a higher throughput than Cuckoo [7]. Based on the experiment the authors concluded that DRAKVUF is a faster and more efficient tool for dynamic malware analysis than Cuckoo (results shown in the Tables I and II).

	40 Samples	100 Samples	200 Samples
DRAKVUF	22 min 34 s	50 min	1 h 41 min 39 s
Cuckoo	32 min 16 s	1 h 5 min 23 s	2 h 8 min 53 s

TABLE I: Duration to carry out malware analysis [7]

Throughput = Total Number of samples analysed successfully / Duration (Time taken)

	40 Samples	100 Samples	200 Samples
DRAKVUF	0.029542097	0.029666667	0.029185112
Cuckoo	0.020661157	0.022686719	0.023018234

TABLE II: Throughput values [7]

The second experiment that was conducted compared the disk utilisation of DRAKVUF and Cuckoo during malware analysis. It used a set of 100 malware samples and found that Cuckoo consumed more disk space than DRAKVUF when memory dumping was enabled [7]. However, DRAKVUF used more disk space when memory dumping was disabled.

	100 Samples (memory dump enabled)	100 Samples (memory dump disabled)
DRAKVUF	5.96 gigabytes (5.96%)	5.96 gigabytes (5.96%)
Cuckoo	11.01 gigabytes (11.02%)	0.16 gigabytes (0.02%)

TABLE III: Disk space utilisation [7]

Cuckoo was been officially discontinued at the time of writing this final report, and the project was archived on GitHub in April 2021 [8].

B. CWSandbox

CWSandbox was one of the first dynamic malware analysis systems to utilise a sandbox environment for monitoring the interaction between the OS and the malware [4]. It operates similarly to Cuckoo where it injects a kernel driver into Windows OS that hooks all exported APIs to intercept the system call performed by user-space programs. This is not as advanced as DRAKVUF as it is vulnerable to detection and tampering and the interface provided by the sandbox is insufficient for tracking kernel-mode rootkits [4].

C. Sandbox Solution Comparison

A summary of comparisons is provided in Table IV:

Criteria / Parameter	Cuckoo Sandbox	DRAKVUF Sandbox	CWSandbox
Stealthiness	X - It uses an in-guest agent, the malware may not exhibit its true malicious nature.	✓- It does not use any in-guest agent, the malware exhibits its true malicious nature.	X - It uses an in-guest agent, the malware may not exhibit its true malicious nature.
System call	✓- It can trace all system calls initiated by malware.	✓- It can trace all system calls initiated by malware.	✓- It can trace all system calls initiated by malware.
Kernel function	X - It cannot trace out kernel functions.	✓- It can trace all kernel functions.	X - It cannot trace out kernel functions.
Automation	✓- It supports the submission of malware samples automatically.	✓- It supports the submission of malware samples automatically.	X - It requires manual interaction to upload and analyse files.
Guest OS support	✓- Windows XP/ 7/8/10, Linux 2.6 or above	✓- Windows 7	✓- Windows XP/ 7/8/10, Linux 2.6 or above
Isolation	✓- Isolated environment for malware analysis.	✓- Isolated environment for malware analysis.	✓- Isolated environment for malware analysis.
Memory Snapshot	✓- It can dump the main memory of guest OS.	✓- It can dump the main memory of guest OS.	✓- It can dump the main memory of guest OS.

TABLE IV: Comparison between Cuckoo, DRAKVUF, and CWSandbox

VI. Design

Prior to the implementation of the DRAKVUF sandbox, the software and technology stack were researched. The DRAKVUF sandbox is built on top of a few layers of software and hardware technologies:

- **Intel VT-x and EPT:** Extensions to x64 architecture that allows virtual machines to run natively on a CPU [9].
- **Xen:** Hypervisor, spawns virtual machines and exposes interfaces for interaction and introspection [9].
- **LibVMI:** Abstracts away introspection interfaces, provides utilities for reading/writing VM memory, parsing VMs' kernel and handling notifications about certain events happening in a VM [9].
- **DRAKVUF:** Stealthily hooks various parts of a guest VM and logs interesting events [9].
- **DRAKVUF Sandbox:** Provides user friendly web interface built using React and high-level analyses.

The DRAKVUF Sandbox is divided into two packages:

- **drakcore:** It is the system core, provides a web interface, an internal task queue and object storage [9].
  - **drakweb:** Web interface that allows user to interact with the sandbox with either REST API or GUI [9].
  - **draksystem:** Internal task management system, used for dispatching jobs between workers [9].
  - **drakminio:** Built-in object storage in which analysis results are stored [9].
  - **drakpostprocess:** Responsible for processing raw analysis logs into a more useable form [9].
- **drakrun:** It is the sandbox worker, wrapper for DRAKVUF, responsible for managing VMs, running analyses and sending results for postprocessing [9].
  - **drakrun 1..n:** Fetches incoming sample [9].

DRAKVUF Sandbox is built around "karton", which is a microservice framework created at CERT Poland as a specialised tool for building flexible malware analysis pipelines. Its main goal is routing tasks between multiple services [9].

A. DRAKVUF Versions

A large aspect of the design phase involved selecting the appropriate versions of DRAKVUF and DRAKVUF Sandbox to be implemented in the project. This decision was instrumental in ensuring that the system not only met the technical requirements but also optimised performance, security, and reliability.

We initially considered various releases of DRAKVUF and DRAKVUF Sandbox, evaluating their features, bug fixes, and improvements. The versions under consideration were sourced from their official repositories on GitHub [10] [11].

1) DRAKVUF v0.7: This version introduced new plugins and tools such as procdump, apimon, and REPL. It also added a new helper library, libusermode, for monitoring usermode code. However we found that DRAKVUF v1.0 offered more stability and features that were crucial for our project's success.

2) DRAKVUF v0.8: This version introduced new plugins like codemon, hidsim and filetracer for Linux. It also saw major cleanups to libinjector with improvements and bug fixes. Despite these enhacements, we did not select this version due to our specific requirements and the compatibility with the DRAKVUF Sandbox version we were considering.

3) DRAKVUF Sandbox v0.17 Series: The v0.17 series introduced features like arch-based prefixes to profile names, rewritten tree generation in postprocess, and support for various file extensions to be analysed. Despite these enhancements, we opted for v0.18.2 due to its more advanced features and improvements that aligned better with our project requirements.

4) DRAKVUF Sandbox v0.18.1 and earlier: These versions were evaluated based on their feature set, stability, and user feedback. For instance v0.18.1 introduced fixes like silencing sample errors and fixing process tree generation. However, v0.18.2 offered more comprehensive improvements and bug fixes, making it a more attractive option for our project.

Our final decision to go with DRAKVUF v1.0 and DRAKVUF Sandbox v0.18.2 was influenced by a combination of factors including feature richness, stability, and compatibility. DRAKVUF v1.0 offered a stable and reliable platform for black-box binary analysis. In combination with DRAKVUF Sandbox v0.18.2, we were confident in achieving an optimal malware analysis environment.

B. Deployment Approach

Two primary deployment approaches were considered: bare metal and virtual machine (VM) deployment. In the bare metal deployment, the DRAKVUF system would be installed directly on a physical machine, offering direct access to hardware resources and optimal performance. VM deployment involves installing the DRAKVUF system on virtual machines that run on physical machines, offering flexibility and ease of management at the expense of some performance overhead.

For this project, after careful consideration, VM deployment was selected as the preferred approach. The decision was influenced by several critical factors, including the need for flexibility and the desire for a system that could be easily scaled and managed. With the VM approach offering features such as snapshots. These can be used in the case of system breaking errors where restoring from snapshots would revert the system back to a stable state.

While bare metal offers performance benefits, the flexibility of a VM deployment was deemed more critical for the project's success. The ability to efficiently manage, scale, and adapt the system to modifications outweighed the marginal performance gains of a bare metal setup.

The VM deployment was facilitated on a robust workstation, equipped with virtualisation-optimised hardware to mitigate performance overhead. The workstation was capable of hosting multiple VMs simultaneously, ensuring that the DRAKVUF system and its auxiliary components could be efficiently managed and scaled as per the projects needs.

DRAKVUF supports nested virtualisation out of the box but is limited to only one hypervisor which is VMWare

Workstation Player, this made the choice for which hypervisor to go with very easy.

C. Hardware

The hardware selection is a critical aspect of our design process, directly influencing the performance, efficiency, and effectiveness of the DRAKVUF system for malware analysis. Our goal was to ensure optimal compatibility, performance, and reliability.

Table V lists the requirements for DRAKVUF [9]:

CPU	Intel with VT-x and Extended Page Table (EPT) features
Host system	Debian 10 Buster Ubuntu 18.04 Bionic Ubuntu 20.04 Focal with at least 2 Core CPU and 5GB RAM
Guest system	Windows 7 (x64) Windows 10 (x64; experimental support)

TABLE V: DRAKVUF Hardware and OS Requirements

We evaluated several hardware options to determine the most suitable platform for deploying DRAKVUF and DRAKVUF Sandbox. The candidates are presented in Tables VI through IX:

Name	Intel NUC
CPU	Intel i7-5557U
RAM	16GB RAM
VT-x & EPT Features	Equipped with VT-x but lacks CPU Performance Counters and IOMMU
Limitations	While compact and efficient, the absence of CPU Performance Counters and IOMMU was a significant limitation for DRAKVUFs requirements

TABLE VI: Intel NUC Specifications

Name	MSI Leopard Laptop
CPU	Intel Core i7-9750H
RAM	16GB RAM
GPU	NVIDIA GTX 1660ti
VT-x & EPT Features	Equipped with VT-x but lacks CPU Performance Counters and IOMMU
Limitations	The inclusion of a dedicated GPU was a plus, but the lack of EPT features made it less ideal for our needs.

TABLE VII: MSI Leopard Specifications

Name	Home Desktop
CPU	Intel Core i7-10700KF
RAM	32GB RAM
GPU	NVIDIA RTX 3050
VT-x & EPT Features	Equipped with VT-x but lacks CPU Performance Counters and IOMMU
Limitations	Despite the impressive memory and graphics capabilities, the absence of EPT features was a drawback.

TABLE VIII: Home Desktop Specifications

Name	Dell Optiplex 7000
CPU	Intel Core i7-12700
RAM	16GB RAM
VT-x & EPT Features	Equipped with VT-x, CPU Performance Counters and IOMMU
Limitations	This supports the required configuration of DRAKVUF but is low on memory.

TABLE IX: Dell Optiplex 7000 Specifications

After a comprehensive evaluation, we selected the Dell Optiplex 7000 which was supplied by Victoria University of Wellington with an Intel Core i7-12700 and 16GB RAM for our deployment.

The Dell Optiplex emerged as the optimal choice due to several compelling reasons:

- CPU: The Intel Core i7-12700 processor offers a robust performance, ensuring that the system can handle complex malware analysis tasks efficiently. With a clock rate of up to 4.90 GHz and 12 Cores.
- Memory: With 16GB RAM, it provides ample memory to support the intensive workloads associated with malware analysis.
- VTx Support: The inclusion of VTx ensures enhanced virtualisation capabilities, a crucial feature for the effective deployment of DRAKVUF.
- CPU Performance Counters: The availability of CPU Performance Counters is pivotal for detailed analysis and monitoring, offering insights that are instrumental for comprehensive malware analysis.
- IOMMU: The support for IOMMU (Input-Output Memory Management Unit) enhances the security and performance of the system, especially in virtualised environments.

The Dell Optiplex's combination of processing power, memory capacity, and essential features like VTx and EPT such as CPU Performance Counters, and IOMMU ensures a balanced, high-performance platform for DRAKVUF. This hardware selection aligns with our project requirements, offering optimal performance, security, and reliability for hypervisor-level malware analysis. It also meant that we could deploy the sandbox to a university owned workstation that can be kept within the

university's network when we had to remotely deploy it to the network as defined in the project requirements.

D. Operating System (Host and Guests)

The choice of operating systems for both the host and guest machines is a crucial aspect of the design, directly impacting the efficiency, security, and overall performance of the malware analysis environment. In this context, we evaluated various operating systems, considering their compatibility with the hardware, support for Xen hypervisor, security features, and ease of use.

We explored a few different operating systems for the host known for their robustness, security and support for virtualisation.

1) Windows 10: Windows 10 was considered for its user-friendly interface, robust security features, and widespread use. However, the need for a UNIX system to execute specific commands and scripts efficiently, and the compatibility with Xen hypervisor, were factors that influenced our decision.

2) Ubuntu 18.04 LTS: Known for its stability and security, this OS was another strong candidate. However, we were inclined towards Ubuntu 20.04 LTS due to its enhanced features, improved security, and extended support, ensuring a future-proof and reliable host environment.

For the Guest OS, the focus was on Windows operating systems, given their prevalence in enterprise environments and the significant volume of malware targeting Windows. The options were also only limited to Windows due to the compatibility of DRAKVUF. The only supported versions were Windows 7 x64 and Windows 10 x64, with the latter having experimental support.

3) Windows 7 x64: This OS was a primary option due to its compatibility with a wide range of malware samples, including those targeting legacy systems. Its inclusion ensures a comprehensive analysis of malware variants exploiting vulnerabilities specific to this OS.

4) Windows 10 x64: This OS was considered for its modern features and security enhancements. However, its experimental support posed challenges, necessitating thorough testing to ensure reliability and effectiveness in the malware analysis environment.

The considerations of host OS options underscored the importance of a UNIX-based system, leading to the selection of Ubuntu 20.04 LTS for its advanced features, security, and compatibility with Xen. For the guest OS, the supported options of Windows 7 x64 and Windows 10 x64 were both selected to ensure a diverse and encompassing malware analysis setup, capable of handling a broad spectrum of malware types and behaviours. This selection process ensures an optimal balance of compatibility, performance, and security for the malware analysis environment.

E. Programming Languages

In the context of programming languages for the DRAKVUF web UI, the utilisation of ReactJS was predetermined, as it serves as the foundation for the existing codebase. The project inherited ReactJS, a widely acclaimed JavaScript library known for enabling the development of highly responsive and dynamic user interfaces. While the choice of ReactJS was not a selection per se, its inherent benefits, such as its component-based structure, efficiency, and robust ecosystem, were instrumental in continuing the development process. The existing codebase, built upon ReactJS, facilitated a seamless continuation of the enhancement and refinement of the DRAKVUF web UI, ensuring an interactive and user-friendly interface for efficient malware analysis and data visualisation.

F. DRAKVUF Architecture

The architecture diagram is illustrated in Fig. 2, based on the DRAKVUF System, the core of the diagram is a detailed look at the DRAKVUF system, showcasing different components and their interconnections, represented by rectangles and arrows to indicate data and process flow.

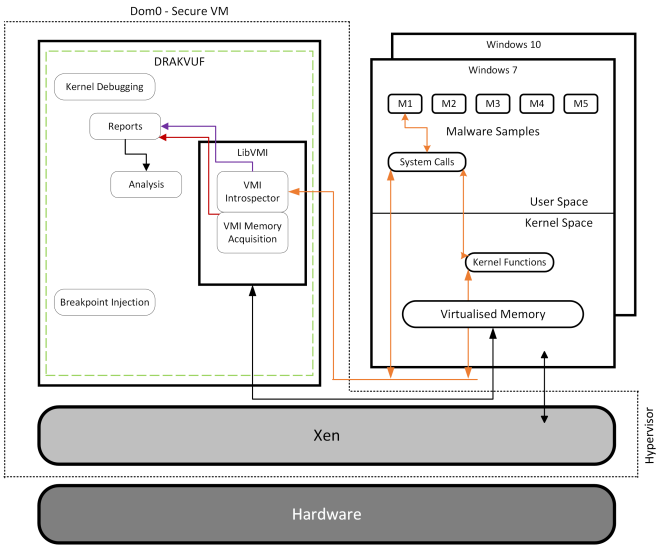


Fig. 2: DRAKVUF Design Architecture Diagram

1) Key Components and Processes:

- DRAKVUF: On the left, the DRAKVUF system is illustrated, emphasising the breakpoint injection process as discussed in the Related Work section.
- Virtual Memory: Various elements and concepts related to virtualised memory, introspection, and the hypervisor are highlighted.
- Hardware & Hypervisor: The two boxes at the bottom serves as a visual representation of a computer system and the Xen hypervisor sitting on top of it.

2) Labels and Annotations:

- Dom0 - Secure VM: Indicates a secure virtual machine environment.
- Windows 7 and 10: Specifies the operating system under analysis or inspection.
- DRAKVUF: The main focus, a tool for malware analysis.
- Kernel Debugging: Indicates processes related to analysing and inspecting the system kernel.

- Malware Samples: Points to the types of files being analysed.
- LibVMI: A library for virtual machine introspection.
- System Calls, Analysis, VM, Introspector, User Space, VMI Memory Acquisition, Kernel Space, Kernel Functions: These labels provide insights into various processes and areas within the virtual memory and malware analysis ecosystem.
- Virtualised Memory, Xen, Hypervisor, Hardware: Indicates the underlying infrastructure supporting the virtual memory system and malware analysis.

G. Submitted Sample Lifecycle

The lifecycle diagram is illustrated in Fig. 3, for the submitted samples on DRAKVUF's web UI. This outlines what the DRAKVUF system does once a sample is submitted. It helped with setting up the email feature where the report would be retrieved after finishing the analysis. We can see when and where the sample analysis results are stored.

The lifecycle of the malware analysis consists of five main parts as seen in Fig. 3:

1. User job submission.
2. Job dispatch to one of the VMs.
3. Malware analysis.
4. Raw result sent back to host machine.
5. Postprocessing of results in readable format.

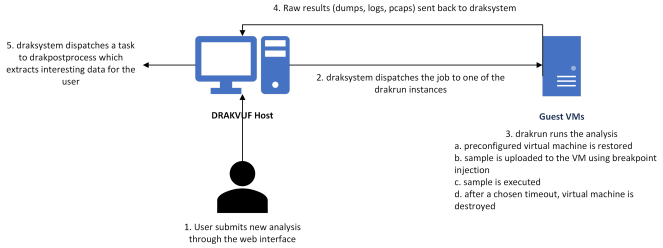


Fig. 3: Submitted Sample Lifecycle

H. Reporting Emails

One of the identified project requirements was to implement a feature in the current web UI that would send an email to a provided email address containing the finished report from DRAKVUF. This functionality would ensure that users are promptly informed, allowing for timely review and action based on the analysis results. We explored several methods to implement this feature, each evaluated based on reliability, ease of integration, and user experience.

There were two options that we came up with for what the email would contain. These were:

- Email the entire report as an attachment
- Email a link to the generated report

Each approach had pros and cons but in the end we chose to provide a link to the generated report rather than providing an attachment. This was due to the fact that malicious samples would be submitted for analysis, email providers could flag hashes and key details of the email attachment as malicious and deny them delivery to their email servers. The reports also vary in size and this could lead to issues when trying to attach them to emails. Providing the user with a link to the report would ensure that the emails would reach their recipients without ending up undelivered or in their junk/spam folders.

There were also considerations made to what tool was going to be used to send the emails. The following were the options that we considered:

- EmailJS
- SendGrid
- 'mail' command on Linux

EmailJS facilitates the sending of emails directly from JavaScript (frontend), eliminating the need for server-side code. It supports various email services, is easy to setup, and offered a user-friendly API.

SendGrid is a cloud-based email delivery service that assists businesses in sending transactional and marketing emails. It is known for its scalability, reliability, and comprehensive analytics.

The mail command on Linux is a utility that allows users to send emails from the command line. It is straightforward but may require additional configuration and lacks the flexibility and features provided by dedicated email services.

After a thorough evaluation, we chose EmailJS. EmailJS offers a straightforward integration process, making it easy to embed the email sending functionality within the DRAKVUF web UI. It eliminates the need for server-side code to send emails, reducing complexity and enhancing the security of the application. It also supports a wide range of email services and allows for the customisation of emails templates, ensuring that users receive well-formatted, informative emails.

The integration of EmailJS into the DRAKVUF web UI involves utilising its API to trigger emails upon the completion of malware analysis. Users who provide their email addresses during the submission of samples for analysis will receive notifications with a link after the analysis is completed. The email templates are designed to be informative yet concise, ensuring users receive relevant data in an easily digestible format.

Within EmailJS there are options to setup an email address that the reports are sent from. We created a Gmail account, *owhiti.sandbox@gmail.com*, which is a placeholder for an official Victoria University email address. The setup was straightforward, it required signing into the Gmail account through a Google API integration within EmailJS. This was the only step that was required for EmailJS to send emails as the Owhiti Sandbox email. The EmailJS tool uses Google's mailservers to send the reporting emails.

VII. Implementation

In the subsequent sections of this chapter, DRAKVUF's implementation process is discussed, including the configuration of DRAKVUF and DRAKVUF Sandbox. The following sections provide an overview of the tools used during the development phase, the deployment process, and the technical aspects of the implementation.

A. Host Setup

The first step for implementation was deploying a host environment that the DRAKVUF system would sit on.

During the deployment many challenges were faced including hardware compatibility issues and limited log access and documentation for troubleshooting. The hardware compatibility issues that we encountered were due to the Xen hypervisor failing to load because of the onboard GPU driver being too modern on the Dell computer that contained the deployment. This issue was resolved by running the DRAKVUF sandbox in a VM. This fixed the problem with a GPU able to be virtualised.

A type-2 VM hypervisor was needed to run the host, VMWare Workstation Player 17 was chosen. This is due to VMWare being the only supported nested virtualisation option. The hypervisor has features like snapshots where it is easy to test different configurations and restore working configurations of the sandbox in case something were to go wrong. My supervisor also made me aware of the fact that the Dell computer that was being used was on a lease by the school. This meant that having the deployment on a VM would be very simple to move to another machine if the lease ended on the computer with the deployment on.

The specifications of the host VM on which DRAKVUF was deployed on are:

- Intel i7-12700 (All 12 Cores)
  - Virtualise Intel VT-x / EPT
  - Virtualise CPU Performance Counters
  - Virtualise IOMMU
- 12GB RAM
- 400GB Storage
- Ubuntu 20.04

This set the foundation for the DRAKVUF system to be easily installed and for it to be compatible within the host environment.

B. DRAKVUF and DRAKVUF Sandbox Deployment

DRAKVUF was implemented using nested virtualisation using Xen running on the VM host environment that was deployed on the provided Dell Optiplex 7000. To begin the DRAKVUF implementation, we first installed both the identified versions of DRAKVUF and DRAKVUF Sandbox stack that we would use to submit malware samples to the analysis engine. Once the underlying components were installed, DRAKVUF created a local host server instance that hosts the sandbox interface on port 6300.

1) Guest OS Deployment: DRAKVUF requires a Windows environment to execute the samples and perform analysis. Subsequently, the next step in the implementation was to generate a clean Windows 7 and 10 ISO image that could use to configure the analysis environment.

We initiated this process with the installation of a Windows 7 ISO. The Windows version we used was Windows 7 x64 Professional SP1 as it provided the Local Security settings tool by default on install.

This Windows deployment was configured to create a virtual machine with more than the DRAKVUF documentation's [9] recommended specifications. As such, we assigned 2 vCPUs and a designated RAM of 3072MB to the VM. Using a Virtual Network Connection (VNC), we connected to the DRAKVUF VM on localhost:5900 to complete the Windows installation and setup the environment for sample analysis.

The environment setup required a few steps that were identified through trial and error. These were:

- Modification of User Account Settings to allow files to be opened without a prompt for admin rights.
- Modification of Local Security settings to allow binaries to make modifications to the Windows Registry without admin approval.
- Installation of the .NET 4.0 binaries and libraries.
- Installation of Microsoft Office 2007 suite for the analysis of Microsoft Office files.
- Modification of the Registry values for the Microsoft Office installation to allow for Visual Basic scripts to be run in the suite from the command line.
- kernel32.dll injected into the System32 folder for 'drak-setup postinstall' command to run properly.

Once the first Guest OS was installed it was time to install the second one which was Windows 10 x64 v2004.

We encountered many issues during the deployment of the second guest OS. The most critical of them being that Windows 10 support was still experimental, which was addressed in the latest release of DRAKVUF Sandbox (v0.19.0). This version was however, still under development and was unreliable due to many scenarios where the installation failed and where it did not provide analysis results. The main limitation was that DRAKVUF Sandbox v0.19.0 was not compatible with DRAKVUF v1.0. It was compatible with DRAKVUF v1.1 but this has not been released yet and both versions are still under development as of writing this report. The other issue was that DRAKVUF Sandbox did not support two guest OS's. A university researcher also had issues with multiple guest OS support and he indicated that through the utilisation of DRAKVUF's scaling feature, another operating system is able to be installed. After testing this approach Windows 10 was able to be installed on the DRAKVUF system but was not reliable. The scaling feature offers to provide multiple instances of drakrun workers. If there are multiple users of the sandbox, scaling can be used to allow all of them to submit and view analysis reports of samples. By manually injecting a different OS for the second drakrun worker multiple guest OS's can be deployed. Due to issues of experimental support for Windows 10, system stability and reliability this option was not implemented in the final deployment. Instead we implemented multiple instances of the Windows 7 guest OS which would serve multiple users of the sandbox without the need to wait for other users to complete their analyses.

Once the Windows 7 guest OS was installed, we scaled the instance to three which was a number that fit with the project requirements and allowed the DRAKVUF Sandbox to run reliably.

We then installed ProcDot which is a tool that creates behavioural graphs based on the execution of samples and their interactions within the guest environment. This was

installed and placed in a folder (/opt/procdot/procmon2dot) where DRAKVUF could access and use it.

After the installation was completed using the 'draksetup postinstall' command provided by the DRAKVUF sandbox stack, we could navigate to the web UI (http://localhost:6300/) and upload some samples to ensure the analysis engine was configured successfully.

C. Web UI Modification Environment

To make modifications to DRAKVUF's existing web UI a working DRAKVUF Sandbox instance was required. From the deployment setup previously, we had downloaded the source code for the DRAKVUF Sandbox. The code needed to be linked with the current sandbox deployment, the following commands were used to do this:

- cd drakcore/drakcore/frontend
- npm install
- export REACT_APP_API_SERVER=http://localhost:6300/
- npm start

After these prerequisites were completed we could begin with the Web UI modifications with a NodeJS development server that would demonstrate live changes to the web UI on the address, http://localhost:3000/

1) Reporting Emails: The implementation of the reporting emails feature was a critical step in enhancing the user experience of the DRAKVUF web UI. This feature ensures that users are promptly notified via email with the results of the malware analysis.

The EmailJS library was imported into the project, enabling the direct sending of emails from the client-side application without the need for a backend server to handle email sending. The 'emailjs-com' library was integrated into the application as shown below.

```
import emailjs from `emailjs-com';
```

The service ID, template ID and public key were configured to authenticate and enable the email sending process through EmailJS. These configurations were essential to ensure the security and integrity of the email sending process.

```
const serviceID = `service_------';
const templateID = `template_------';
const publicKey = `--------------';
```

Functions were implemented to set, get, and clear the recipient's email address. These functions ensure that the email address is correctly handled, stored temporarily for sending the email, and then cleared to maintain privacy and security.

```
setRecipientEmail(recipientEmailAddress){
    this.recipientEmailAddress =
    recipientEmailAddress;
},
getRecipientEmail(){
    return recipientEmailAddress;
},
clearRecipientEmailAddress(){
    this.recipientEmailAddress = null;
},
```

The 'sendEmail' function was implemented to send emails to users containing the results of the malware analysis. If a recipient email address is provided, an email, complete with a link to the analysis results, is sent to the user. The email sending process is handled by EmailJS, ensuring reliability and security.

```
sendEmail(link) {
  if(this.recipientEmailAddress){
    link = webLink + link;
    const templateParams = {
        to_email:
        this.recipientEmailAddress,
        message: link
    }

    emailjs.send(serviceID,
    templateID, templateParams,
    publicKey)

    .then((res) => {
        console.log(`res: ', res);
        this.clearRecipientEmailAddress();
    })
    .catch((err) => {
        console.log(`err: ', err);
    });
  }
}
```

The implementation was thoroughly tested to ensure the reliability and accuracy of the email notification. Each email contains a direct link to the specific analysis results on the DRAKVUF web UI, ensuring users can easily access and review the findings.

2) Analysis Time: The implementation of an analysis time slider was a significant feature for the web UI. It provides users with the ability to specify the analysis time when uploading a sample for evaluation. This feature ensures that users have flexibility and control over the duration of the analysis, allowing for a more customised and user-centric experience. Malware also does not finish execution within a set timeframe, having this slider makes it easier for the user to control how long the analysis is carried out for.

A dedicated 'MinuteSlider' component was created in React to facilitate the user's input for specifying the analysis time. This component is rendered as a slider input element, allowing users to easily adjust the analysis time according to their needs.

```
class MinuteSlider extends Component {
  render() {
    return (
      <input
        onInput={this.props.onInput}
        className=``custom-range"
        type=``range"
        name={this.props.name}
        min={60 * this.props.min}
        max={60 * this.props.max}
        step={60}
        defaultValue={60 * this.props.default}
      />
    );
  }
}
```

The 'MinuteSlider' component is configured with properties to set the minimum, maximum, and default values for the analysis time. The values are calculated in seconds, with the slider stepping in 60-second intervals to represent minutes. The onInput event handler is used to capture the user's input and update the state accordingly.

```
<MinuteSlider
  min={1}
  max={10}
  default={10}
  name=``analysisTime"
  onInput={this.handleInput}
/>
```

The 'handleInput' function is implemented to manage the state updates when the user adjusts the 'MinuteSlider'. The analysis time, represented in seconds, is stored in the component's state to be used during the sample upload process.

```
handleInput(event) {
  this.setState({ error: null });
  const field = event.target.name;
  if (field === ``analysisTime") {
    this.setState({timeout:
    event.target.value});
  }
}
```

The selected analysis time is displayed to the user in minutes, offering a clear and understandable representation of the duration for which the analysis will run.

```
<small>{this.state.timeout/60} min</small>
```

The implementation was rigorously tested to ensure that the slider responds accurately to user input and that the selected analysis time is correctly stored and utilised during the sample analysis process. The feature was validated to ensure a seamless user experience, accurately influencing the duration of the malware analysis.

3) Virtual Machine Configuration: A modification made to the web UI was the addition of the Virtual Machine Configuration section. This is a static table that shows the available configurations of virtual machines. It pulls the details from Xen and displays them as seen in Fig. 4 below.

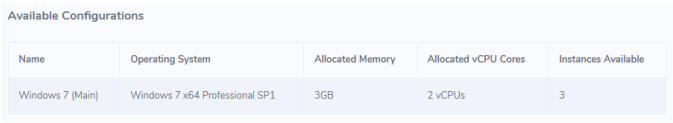


Fig. 4: Available Configurations Screenshot

Since only one guest OS was implemented there is only one configuration available. There are however multiple instances available of that one guest OS.

4) Instructions: Instructions for the sample upload page were added. There was ambiguity around what was and was not needed for the analysis to run and what file types the system accepted. The instructions pose as a guide to help new users how to navigate and use the upload sample page.

The instructions can be viewed in Fig. 5 below.

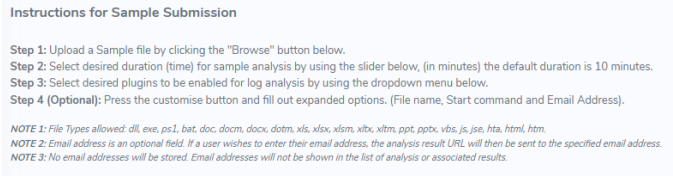


Fig. 5: Submission Instructions Screenshot

They provide clear steps for users on how to submit a file for analysis and what the different configuration options mean.

D. Building drakcore Package and final deployment

After the web UI modifications were made the system needed to be built and deployed again as the changes were on a NodeJS development server and not on the actual DRAKVUF instance that was deployed. This process ensures that the updated and enhanced web UI is packaged and ready for deployment, allowing users to benefit from the new features and improvements.

During the build process, a significant challenge was encountered. The SSL certificates of some external packages such as Minio and Python had expired, leading to issues in building the drakcore package. This obstacle required immediate attention to ensure the successful compilation and packaging of the updated web UI.

To address this issue, a meticulous review of the build process was undertaken, and the problematic external package links were identified. The expired SSL certificates necessitated the modification of these links to ensure the seamless retrieval of the required packages. The build process was guided by the instructions provided in the official documentation for building the drakcore package [12].

After the package was built we had to deploy the entire DRAKVUF system again as per the steps outlined in Section B of the implementation chapter.

Following this implementation and deployment, the DRAKVUF system on the Dell Optiplex workstation was transitioned to a more robust and accessible environment. The deployment was extended to the Engineering and Computer Science (ECS) school servers to enhance accessibility and performance. The web address for the deployment is 'http://103.196.108.47:6300/'. This move was instrumental in elevating the system's availability, making it remotely accessible to users. The integration into the ECS servers marked a significant milestone, ensuring that the system's capabilities could be leveraged efficiently and securely.

The connection to the network switch was a pivotal aspect of this deployment, ensuring network integration and data flow. By being connected to the network switch, the DRAKVUF system was effectively positioned within the university's DMZ. This strategic placement ensured that the system was not only secure but also benefited from the enhanced network infrastructure, ensuring high-speed data processing and transmission.

VIII. Evaluation

The methods used to assess this project's effectiveness are presented in this section. This covers the survey's methodology and questions. Following the data gathering, this section addresses the project's applicability and the solution's measured effectiveness.

A. Survey

A survey was undertaken to measure usability and user satisfaction when performing malware analysis through the

DRAKVUF Sandbox. Eight individuals undertook the survey. All individuals within the survey pool were associated with the Owhiti Cyber Security group, which included students and academic researchers. Participants were emailed the survey specifications, which included the steps required to gain access to submission samples if they did not have any on their computer alongside a link to the DRAKVUF Sandbox web UI. The subjects were then asked to access the tool and fill out a web survey provided via a Microsoft Form.

1) Survey Questions: The survey consisted of nine questions that asked users about their prior knowledge and about the tool's usability. It encouraged individuals to use the tool to perform analysis and extract information which enabled them to answer specific questions. The questions asked and the reasoning behind them are summarised below:

1. Have you ever used an online malware sandbox before? This question targeted an evaluation of the subject's prior knowledge on sandboxes and if they were familiar with how to submit samples and the common configuration settings on them.

2. Do you know how a malware sandbox works? This question targeted an evaluation of the subject's prior knowledge on sandboxes and if they were familiar with the inner-workings of them.

The following questions were choice questions, they spanned from Strongly Disagree to Strongly Agree and assessed the usability, effectiveness and feel of the web UI.

3. It was easy and intuitive to submit a file for analysis on DRAKVUF This question targeted an evaluation of the tool's usability to gauge if it was too complicated and required more explanation or if users could understand the different features and perform analysis.

4. It was easy to find and select additional configuration options (eg: Analysis Time, Modules enabled, Email input, etc.) This question tests a subject's ability to find and select additional configuration options on the web UI.

5. It was easy to find and view/download the logs and reporting artefacts created by the sandbox This question targets the evaluation of the tool's reporting feature and if subjects were able to easily find the artefacts and view/download them.

6. The sandbox 'look and feel' was consistent throughout the whole process This question targets the evaluation of the tool's 'look and feel' and if the subjects found it consistent throughout the process.

7. Did you receive an email for the sample file submission results in your inbox or junk mail? (After inputting your email in the 'Customise' options) This question aims to find out whether the email was received in the Inbox or Junk folder (or not received at all), to see if there are any bugs with this feature.

8. Is there any new functionality you would like to be added to this system? We can use this information to provide further features based on the information provided by the subjects.

9. Is there any other feedback/suggestions you would like to provide? This question provides space for additional comments and feedback regarding the proposed system, which gives insight into possible future developments.

2) Survey Results: Eight individuals completed the survey, with the overall trend being that they were satisfied with the tool's usability and the information it provided.

The subjects' feedback on usability was generally positive. A majority expressed that submitting a file for analysis on DRAKVUF was straightforward, attributing their positive experience to the intuitive design and user-friendly interface. The ease of locating and selecting additional configuration options was also highlighted, with most participants agreeing or strongly agreeing on its simplicity.

The email notification system received positive feedback. Five participants received the email notifications in their inbox, three did not receive any. Upon further inspection however, it revealed that the users that did not receive a notification did not enter their email address into the input box, meaning the system functioned appropriately. However, for any future surveys, the point to provide an email address should be emphasised.

Participants provided valuable insights on potential enhancements. Suggestions included the incorporation of more detailed explanations of features and a progress bar to inform users of the analysis status.

The consistency in the 'look and feel' of the sandbox throughout the process was generally well-received, with most participants agreeing or strongly agreeing on its consistency. However, one participant noted that the graphics for the behavioural graph did not align with the overall UI design, suggesting an area for aesthetic enhancement.

The feedback from the eight participants provides invaluable insights for the refinement of the DRAKVUF web UI. The general satisfaction with the tool's usability is encouraging, while the identified areas for improvement offer clear pathways for enhancement.

Although the survey had a small pool of participants regarding usability testing, eight participants allowed us to find almost as many usability problems as you'd find using many more test participants [13].

B. Functional Evaluation

The DRAKVUF web UI, deployed on the ECS servers on Friday 22nd September 2023, has been tested to ensure its functionality, reliability, and efficiency. The web UI has proven to be a robust tool, consistently delivering accurate and comprehensive results without any failures or significant issues since deployment.

There have been more than 30 sample submissions to the web UI since deployment, without failure. Compared to the Cuckoo system that the Owhiti Cyber Security group currently uses this is a dramatic increase in reliability. My supervisors have indicated that the Cuckoo system is notorious for crashing even while it is not in use. DRAKVUF has been active for three weeks now on the ECS servers and has not failed nor crashed.

A detailed examination of one of the sample submission instances provides a comprehensive insight into the system's

capabilities. The web UI is structured to offer a user-friendly experience, ensuring that users can navigate through diverse sections to access the required information. The sample in question is a famous ransomware based malware called WannaCry. It is a heavily documented ransomware, when it was first released, it infected over 230,000 computers across 150 different countries.

The analysis results are presented in a well-organised manner, offering a blend of graphical and textual data. The behavioural graph, for instance, provides a visual representation of the malware's behaviours. It maps out the interactions and activities of the malware, offering a clear and concise view of its operational patterns. For the case of WannaCry, it shows that a script (cscript.exe) is used to encrypt all of the files in the analysis environment. Then a program is created called WannaDecryptor.exe which is the tool the victim is given access to after the ransom has been paid. Using DRAKVUF malware analyst's can locate where the WannaDecryptor.exe tool is stored and can find and use it without having to pay the ransom the attackers want. This visual aid is instrumental in quickly understanding the malware's behaviour, interactions, and potential impact on the system.

The detailed logs and reports generated by the system are another significant asset. They offer a granular view of the malware's headers and operations, including metadata (as seen in Fig. 6, system calls, and file interactions. For malware analysts, this level of detail is invaluable. For the case of WannaCry the report provides many Indicators of Compromise (IoCs) such as its hashes, IP addresses, and specific arguments passed to Windows DLLs. DRAKVUF not only aids in understanding the specific characteristics and behaviours of the malware but also facilitates the development of effective countermeasures and mitigation strategies.

Metadata

SHA256	be22645c61949ad6a077373a7d6cd85e3fae44315632f161adc4c99d5a8e6844
Magic bytes	PE32 executable (GUI) Intel 80386, for MS Windows
Start command	C:\Users\drakvuf\Desktop\WannaCry.exe
Plugins	syscalls, apimon, procmon, tlsmon, memdump
Started at	2023-09-25 05:24:07
Finished at	2023-09-25 05:34:31

Fig. 6: DRAKVUF - WannaCry Metadata Results

The system's capability to provide a detailed analysis of the memory dumps is another noteworthy feature. Analysts can delve deep into this aspect, extracting critical information that can be pivotal in understanding the malware's infiltration techniques, propagation mechanisms, and data exfiltration strategies.

For more information on DRAKVUFs findings on the ransomware WannaCry, please visit this link: http://103.196.108.47:6300/analysis/a72d43bf-0994-4eda-9891-4387acd29187.

Screenshots of the analysis can be viewed in the appendix.

C. Project Requirements Evaluation

The evaluation of the DRAKVUF Malware Sandbox project is essential to assess the alignment with the initial requirements. This section provides an analysis of the met and unmet requirements.

The project has achieved notable successes. The web interface is remotely accessible and user-friendly - confirmed by positive user feedback. The email notification system is operational, ensuring users are informed of the analysis results in a timely manner. The file submission process is straightforward, and users have found the instructions clear and helpful.

However, there remains an unmet requirement pertaining to the support for multiple operating systems. The initial objective was to incorporate two operating systems, Windows 7 and Windows 10, to offer a diverse and adaptable environment for malware analysis. This diversity was aimed at ensuring that the tool could be utilised for a comprehensive analysis, catering to a wide array of malware designed for different operating systems. However, the current implementation is confined to a single guest OS type. This limitation is attributed to constraints related to resource allocation, technical complexities associated with integrating multiple operating systems, challenges in ensuring stable and reliable performance across the platforms, and the lack of support in the official stable release of the system.

The DRAKVUF Malware Sandbox project has made significant progress on the original implementation. The unmet requirement does not overshadow the achievements but rather offers an opportunity for growth and enhancement.

IX. Conclusion

The DRAKVUF Malware Sandbox project, represents a significant stride in the field of malware analysis. The project successfully extended the DRAKVUF sandbox's web UI, enhancing capabilities. The project was designed to be remotely accessible through a web interface and showed the available virtual machine and operating system along with their respective configurations. It allowed the management of multiple instances of DRAKVUF and supported the submission of a file to be analysed by the operating systems through the web interface. Users could download the reports generated by DRAKVUF or view them through the web interface, and reports could also be emailed to a designated address provided by the user.

Despite the project's achievements, it faced challenges including the limitation of supporting only one guest OS type. However, the accomplishments in enhancing the user interface, improving the reporting component, and extending the queuing system are notable milestones that contribute to the project's overall success.

In conclusion, the DRAKVUF Malware Sandbox project shows the potential of continuous innovation and refinement in malware analysis. While there are areas for improvement, the achievements made are significant and lay a robust foundation for future enhancements. The project not only contributes to the body of knowledge in malware analysis but also offers

practical solutions that can be adopted and adapted in real-world scenarios to understand and mitigate the impacts of malware effectively. The collaborative and iterative approach to the project's development ensures that it remains responsive to emerging needs and challenges in the dynamic landscape of cybersecurity.

X. Future Work

As the DRAKVUF Malware Sandbox project comes to a close, several opportunities for future work and enhancement have been identified to augment its functionality, efficiency, and user experience.

A. Integration of Multiple Operating Systems

One of the primary areas for future development is the integration of multiple operating systems. The current limitation to a single guest OS can be expanded to include a variety of operating systems, enhancing the sandbox's versatility and making it adaptable to a broader range of malware types.

There is currently an issue on the DRAKVUF Sandbox GitHub [14] that outlines the same problem.

B. Incorporation of Machine Learning and AI

The integration of machine learning and artificial intelligence algorithms can augment the sandbox's capability to identify, analyse, and mitigate malware. These technologies can enhance the accuracy and speed of malware detection and analysis, offering predictive and prescriptive insights.

C. Integration with Elastic Stack for Comprehensive Visualisations

The integration of DRAKVUF Malware Sandbox with Elastic Stack is a considered step to improve data interpretation and usability. This integration allows for efficient data management and real-time processing, enhancing the tool's responsiveness. Users can benefit from interactive visualisations, offering a clear and direct understanding of malware analysis findings.

The use of Elastic Stack enables the creation of custom dashboards, allowing users to personalise data views and analytics according to specific needs. The advanced analytics feature of Elastic Stack provides deeper insights into malware trends and patterns. Data integration protocols and APIs will be developed to ensure a smooth transfer of data between DRAKVUF and Elastic Stack, maintaining data consistency and reliability.

Acknowledgments

I would like to thank my two supervisors Masood Mansoori and Lisa Patterson for providing support throughout the project. It would not be in the current state without their help. I would also like to thank Muhammad Shabbir Abbasi for his support with the multiple OS feature. Although the implementation was not stable; it provided insights into the workings of the system.

References

[1] S. Megira, A. Pangesti, and F. Wibowo, "Malware analysis and detection using reverse engineering technique," in *Journal of Physics: Conference Series*, vol. 1140, no. 1. IOP Publishing, 2018, p. 012042.

[2] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[3] S. YusirwanS, Y. Prayudi, and I. Riadi, "Implementation of malware analysis using static and dynamic analysis method," *International Journal of Computer Applications*, vol. 117, no. 6, pp. 11–15, 2015.

[4] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014, pp. 386–395.

[5] "Drakvuf® black-box binary analysis system." [Online]. Available: https://drakvuf.com/

[6] S. Kovalev, "Reading the contents of deleted and modified files in the virtualization based black-box binary analysis system drakvuf," *Proceedings of the Institute for System Programming of the RAS*, vol. 30, no. 5, pp. 109–122, 2018.

[7] A. A. R. Melvin and G. J. W. Kathrine, "A quest for best: A detailed comparison between drakvuf-vmi-based and cuckoo sandbox-based technique for dynamic malware analysis," in *Intelligence in Big Data Technologies—Beyond the Hype: Proceedings of ICBDCC 2019*. Springer, 2021, pp. 275–290.

[8] C. Sandbox, "Cuckoo sandbox is an automated dynamic malware analysis system," 2021. [Online]. Available: https://github.com/cuckoosandbox/cuckoo

[9] "Understanding the sandbox - drakvuf sandbox v0.18.2 documentation." [Online]. Available: https://drakvuf-sandbox.readthedocs.io/en/latest/understanding_sandbox.html

[10] Tklengyel, "Releases · tklengyel/drakvuf," 2021. [Online]. Available: https://github.com/tklengyel/drakvuf/releases

[11] CERT-Polska, "Releases · cert-polska/drakvuf-sandbox," 2021. [Online]. Available: https://github.com/CERT-Polska/drakvuf-sandbox/releases

[12] "Building installation packages - drakvuf sandbox v0.18.2 documentation." [Online]. Available: https://drakvuf-sandbox.readthedocs.io/en/latest/building_packages.html

[13] J. Nielsen, "How many test users in a usability study?" 2012. [Online]. Available: https://www.nngroup.com/articles/how-many-test-users/

[14] CERT-Polska, "Support for multiple os malware analysis · issue 462 · cert-polska/drakvuf-sandbox." [Online]. Available: https://github.com/CERT-Polska/drakvuf-sandbox/issues/462

APPENDICES

DRAKVUF ANALYSIS - WANNACRY METADATA AND PROCESS TREE OUTPUT

Report

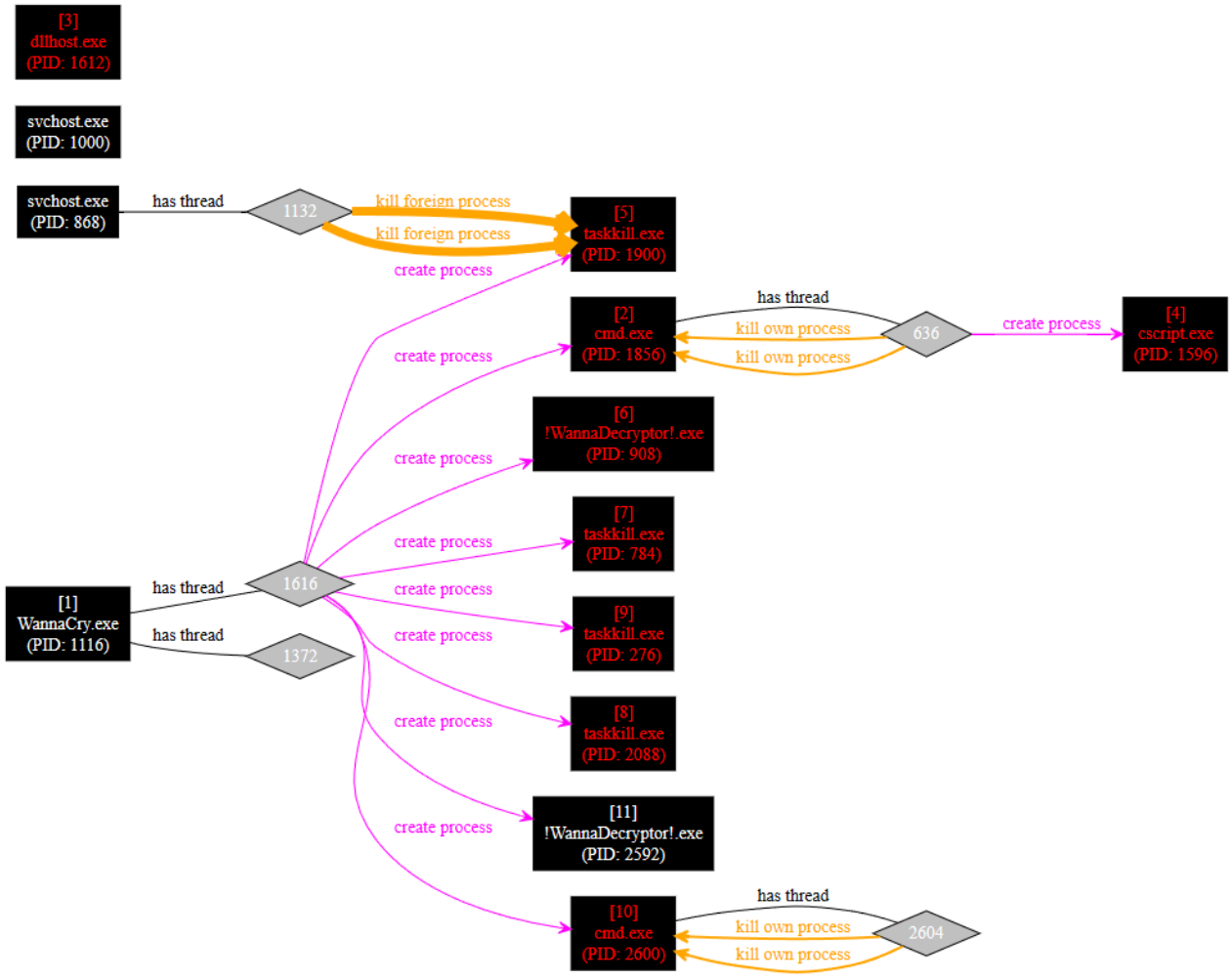
Process tree

- Mocked parent (0)
- Mocked parent (332)
- Mocked parent (392)
- Mocked parent (1680)
  - explorer.exe (1724)
    - WannaCry.exe (1116)
      - taskkill.exe (276)
      - taskkill.exe (784)
      - !WannaDecryptor!.exe (908)
      - cmd.exe (1856)
      - taskkill.exe (1900)
      - taskkill.exe (2088)
      - !WannaDecryptor!.exe (2592)
      - cmd.exe (2600)
      - !WannaDecryptor!.exe (2608)

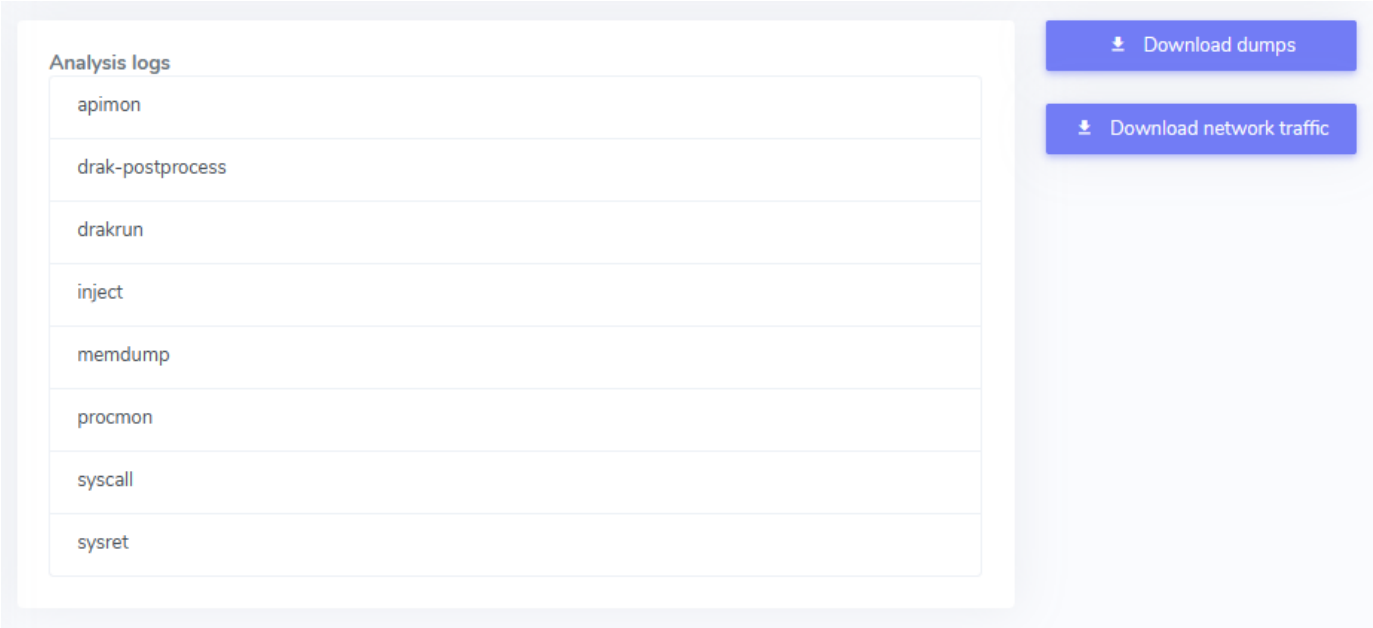
Metadata

SHA256	be22645c61949ad6a077373a7d6cd85e3fae44315632f161adc4c99d5a8e6844
Magic bytes	PE32 executable (GUI) Intel 80386, for MS Windows
Start command	C:\Users\drakvuf\Desktop\WannaCry.exe
Plugins	syscalls, apimon, procmon, tlsmon, memdump
Started at	2023-09-25 05:24:07
Finished at	2023-09-25 05:34:31

DRAKVUF ANALYSIS - WANNACRY BEHAVIOURAL GRAPH OUTPUT



DRAKVUF ANALYSIS - WANNACRY LOGS OUTPUT



DRAKVUF ANALYSIS - WANNACRY API CALLS OUTPUT