Isolation-Centric Operating Systems for the Enterprise

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Computer Science

By:

Aidan Wech
Annalisa Allan
Caleb Scopetski
Charles Kneissl-Williams
Cole Ouellette
Connor West
Edison Zhang
Jacob Chlebowski
Mira Plante
Quentin Hall
Zaq Humphrey

Project Advisors:

Craig Shue　Jun Dai　Xiaoyan Sun

Date: March 2024

Abstract

Every year, ransomware introduces over $2 billion in damages to enterprises. Current antivirus technologies do not sufficiently serve this need. With the majority of computers set up in a single execution environment, one compromised application may spread into the entire system and, from there, the entire network. Because attackers are always developing new malware attacks, preventing malware from getting into a system entirely is impossible. IT administrators need a way to minimize the damage that malware can inflict across their system once it has already infiltrated a system.

One way to minimize potential damage is through isolation. Isolation mitigates the damage that malware can cause by containing the malware within the infected environment and limiting communication with other components on the system. Existing methods of isolation introduce usability and performance costs on the user. As such, IT administrators need a more usable and performant way of providing isolation to the end user. Isolation restricts communication between environments to prevent malware from spreading. However, strict isolation limits user productivity and collaboration within the enterprise. IT administrators need a way to maintain the benefits of isolation while maximizing inter-environment productivity. In doing so, the effectiveness of isolation can be uncompromised. Administrators need an efficient way to observe and identify anomalous activity to enable collaboration while maintaining the integrity of isolation.

To implement strict isolation while maintaining usability and reducing performance costs on the end user, we created a system that utilizes virtualization to provide isolation between applications. In doing so, we created a web interface that mirrors a traditional environment allowing for seamless VM access. Additionally, we leveraged remote computational resources to reduce the burden of running multiple VMs on the end user system and provide rapid availability of VMs.

To maintain the benefits of isolation while maximizing inter-environment productivity, we provided a more usable way of creating and maintaining policy by modifying an existing SDN-based policy management tool. The tool uses user intent data in decisions by adding machine context labels to assist decision making in multi-machine environments. We performed testing on the updated tool to compare it to traditional networking systems and the unmodified version of the tool using three scenarios based on real-world threat models.

To provide an efficient way to observe and identify anomalous activity, we created a custom virtual environment that utilizes a series of sensors to record workflow data which our visualization scripts parse to present the user with time series graphs detailing different types of system activity over runtime. Our time series graphs aid the user in deciding if sandbox activity is indicative of a benign workflow.

To test the usability and performance of our system for isolating environments, we evaluated application

startup time, wasted screen space, VM startup processes, and computing resource costs. We compared our system to a traditional desktop system and a traditional VM, VirtualBox. For application startup time, our system took slightly longer than a traditional system to start all applications, but was significantly faster than VirtualBox across all scenarios. For screen space, our system contained 4.85% wasted screen space running an application in comparison to a traditional system rendering the same application, while VirtualBox contained 7.8% wasted screen space under the same circumstances. For the VM startup process, the KLM time for accessing a VM on our system was faster than in VirtualBox. In evaluating performance costs, our client performed similarly to the traditional desktop environment and heavily outperformed VirtualBox in the most intensive scenario. Additionally, our client consumed minimal additional resources as the number of running applications increased, which suggests increased scalability over the traditional environment. As expected, our server consumed the most resources in the most intensive scenario due to the overhead associated with single application isolation with VMs.

To ensure that policy scalability in a multi-machine environment did not interfere with firewall accuracy, the modified version of the tool was compared to the original tool and the traditional networking system. This was done by running each system through a set of scenarios representing real-life security threats. Our results indicated that the system integrating the modified tool had increased policy scalability without majorly affecting accuracy when compared to the other two systems. The modified tool was most successful in situations where user or machine groups were implemented to create policy decisions.

To observe and identify anomalous activity, the team recorded time series data to test the idea that untrusted activities can be identified when there is misalignment between user activities and system behavior. When observing activities of malicious and benign scenarios, our expectations were met. In scenarios involving ransomware, there was a noticeable absence of consistent user-initiated keystrokes and mouse clicks, or an irregular pattern that deviated significantly from expected human behavior. Some samples revealed their malicious intent almost immediately after execution through a rapid succession of Indicators of Compromise, while others required a more extended observation period. Nevertheless, the combination of network behavior, file activity, and the absence or irregularity of user interaction data typically yielded enough information within the first few minutes of execution to classify the scenario accurately.

Acknowledgments

We extend our sincere gratitude to Adam Beauchaine for his unwavering support and encouragement throughout this project. His mentorship not only broadened our perspectives but also challenged us to think creatively, making this endeavor truly feel like our own.

Furthermore, we are deeply thankful to Professor Jun Dai and Professor Xiaoyan Sun for their invaluable feedback and diverse insights, which enriched our approach to the project from various angles.

Lastly, we express our heartfelt appreciation to Professor Craig Shue for his steadfast guidance and patience as our research advisor, without which this work would not have been possible.

Contents

1 Introduction **10**

1.1 Need . . . 10

1.2 Approach . . . 11

1.3 Benefit Over Cost . . . 12

1.4 Competition . . . 14

1.5 Associated Research Questions . . . 17

1.6 Contributions . . . 18

2 Background **20**

2.1 Security by Isolation . . . 20

2.1.1 Virtual Machines . . . 20

2.1.2 Hypervisors . . . 21

2.1.3 Qubes OS . . . 23

2.1.4 Sandboxes . . . 24

2.2 Remote Connection Technologies . . . 30

2.2.1 Remote Desktop Protocol . . . 30

2.2.2 Virtual Network Computing . . . 31

2.2.3 Apache Guacamole . . . 31

2.3 Detection . . . 33

2.3.1 Intrusion Detection Systems . . . 34

2.3.2 Antivirus . . . 34

2.3.3 Security Information and Event Management Systems . . . 34

2.4 Sandbox Instrumentation . . . 35

2.4.1 Cuckoo Sandbox Report . . . 35

2.4.2 Suricata . . . 36

2.4.3 File Permissions . . . 36

2.5 Harbinger . . . 37

2.6 Network Access Control . . . 37

2.6.1 Domain Name System Protocol for NAC . . . 38

2.7 Firewalls . . . 38

2.7.1 Common Misconfigurations . . . 38

2.7.2 Virtual Firewalls . . . 39

3 Methodology **39**

3.1 System for Isolating Environments 40

3.1.1 System Design Overview 40

3.1.2 System Implementation 40

3.1.3 Experiments 44

3.2 Modified Harbinger for Improved Accuracy and Policy Scalability 52

3.2.1 Rationale for Harbinger Use 53

3.2.2 Harbinger Installation and Setup 53

3.2.3 Modifying Harbinger 53

3.2.4 Modified Harbinger Testing Suite 54

3.2.5 Scenario 1: Sensitive Financial Information Protection 55

3.2.6 Scenario 2: Automatic Scripts from Malicious Files 55

3.2.7 Scenario 3: Role-Based Access Control 56

3.3 Accessibility of Understanding UI Influence over Time Series Data 56

3.3.1 Research Design 57

3.3.2 Indicators of Compromise 58

3.3.3 Experiment 1: Synthetic Malware 62

3.3.4 Experiment 2: Legitimate Malware 63

3.3.5 Experiment 3: Benign Operations 63

3.3.6 Aligning IOCs with UI Activity 64

4 Results and Findings **65**

4.1 System for Isolating Environments 65

4.1.1 Usability Experiment 1: What effect do VM hot loading techniques have on startup time? 65

4.1.2 Usability Experiment 2: How much wasted screen space is incurred as a cost of using nested windows? 67

4.1.3 Usability Experiment 3: What are the number of steps to create a new VM when compared to other VMMs? 68

4.1.4 Performance Experiment 1: What computing costs are incurred due to using our virtualized system on the client and server-side? 69

4.2 Modified Harbinger for Improved Accuracy and Policy Scalability 72

4.2.1 Scenario 1: Sensitive Financial Information Protection 73

4.2.2 Scenario 2: Automatic Scripts from Malicious Files 73
4.2.3 Scenario 3: Role-Based Access Control 74
4.2.4 Discussion 75
4.3 Understanding UI Influence over Time Series Data 75
4.3.1 IOC Detection Capabilities of Our Solution 75
4.3.2 Behavioral Analysis and IOC Correlation with Cuckoo Sandbox and Keylogger Data . 76
4.3.3 Temporal Analysis of UI and System Activities in Different Events 77

5 Limitations and Future Work Opportunities **83**
5.1 System for Isolating Environments 83
5.2 Modified Harbinger for Improved Accuracy and Policy Scalability 85
5.3 Understanding UI Influence over Time Series Data 85

6 Conclusion **87**
6.1 System for Isolating Environments 87
6.2 Modified Harbinger for Improved Accuracy and Policy Scalability 88
6.3 Understanding UI Influence over Time Series Data 88

List of Tables

1 Comparison of Bromium, Cuckoo Sandbox, Joe Sandbox, and GFI Sandbox 25
2 Tolerance Values and Percent Similarities for WPI Homepage Images 47
3 KLM atomic actions and their interaction time 49
4 App Startup Time Across User Systems Results 66
5 Wasted Screen Space comparing Traditional Environment, VirtualBox, and Our System . . . 68
6 KLM Key 69
7 KLM Results 69
8 Computing Resources used across all scenarios and environments 71
9 Number of flows detected and number of policies for each system in Scenario 1 73
10 Number of flows detected and number of policies for each system in Scenario 2 74
11 Number of flows detected and number of policies for each system in Scenario 3 74
12 Key features of malware families 76

List of Figures

1 Flow diagram illustrating the life cycle of an application in our system. 41
2 System architecture diagram. It demonstrates all the components in the system and how they interact. 42
3 Screenshot of a user utilizing the web page client to access a remote Blender VM 42
4 Image Difference Visualization Using www.wpi.edu Homepage 46
5 Comparison of Application Loading Times Across Systems 66
6 Average % CPU Load across all scenarios 70
7 Average RAM (MB) across all scenarios 70
8 Run 1.1: Synthetic Crypto Locker 77
9 Run 1.2: Synthetic Locky 78
10 Run 2.1: Encryptorps1 79
11 Run 2.2: Locky 79
12 Run 2.3: Jigsaw 80
13 Run 2.4: Xorist 81
14 Run 2.5: Cerber 81
15 Run 3.1: Benign General Usage 82

16 Run 3.2: Benign Docm Usage . . . 83
A.1 Alphabet . . . 100
A.2 CryptoWall . . . 100
A.3 EDA2 . . . 101
A.4 EternalRocks . . . 101
A.5 GhostCrypter . . . 102
A.6 Golden Eyes . . . 102
A.7 L0CK . . . 103
A.8 Mamba . . . 103
A.9 Matsnu . . . 104
A.10 Petya2 . . . 104
A.11 Ransomeware . . . 105
A.12 ShellLocker . . . 105
A.13 WannaCrypt . . . 106

1 Introduction

We present our introduction structured as a value proposition. We cover the need for the research, our approach, benefits over costs of the approach, and existing competition.

1.1 Need

In 2022, there were 5.5 billion malware attacks [1], and the average cost of a data breach worldwide is \$4.35 million [2]. Because attackers are always developing new malware attacks, preventing malware from getting into a system entirely is impossible. IT administrators need a better, more scalable way to secure their organization's assets and reduce the damage that can be caused by a single attack. One such approach is security by isolation, which separates data assets into groups using virtualization or containerization. A lack of isolation means malware can compromise all the assets contained within that system. Resiliency is needed so that if one isolated environment is compromised, the remaining environments are still available for the organization to continue operating, and isolation is a common approach to add this into systems.

A simple solution that achieves isolation is to run multiple virtual machines (VM) to keep assets separate, so if one machine is compromised, the others are not. However, this would involve the user having to manually manage multiple VMs and having multiple desktop windows open on their screen. This is overwhelming for many users, which reduces the usability by having to learn and adjust to this new environment. Following the principle of familiarity [3], mirroring the traditional desktop environment reduces the learning time for users and decreases the negative impact on usability [4]. One issue with running multiple VMs locally is that each Windows 10 VM is recommended to have at least 2 GB of RAM [5], so most modern computers cannot run multiple VMs efficiently. As such, IT administrators need a more usable and performant way of providing isolation to the end user.

Isolation restricts communication between environments to prevent malware from spreading. However, strict isolation limits user productivity and collaboration within the enterprise. Current network access control solutions apply many blanket policies over a whole network. These are often too strict or lenient for some hosts, and are difficult for network administrators to manage. IT administrators need a way to maintain the benefits of isolation while maximizing inter-environment productivity.

In relaxing protection to allow for inter-environment productivity, the effectiveness of isolation can be compromised. Administrators need an efficient way to observe and identify anomalous activity to enable collaboration, while maintaining the integrity of isolation. IT administrators need a way to determine if unknown assets and destinations are trustworthy. Trust can be established through transparent behavior. Thus, IT administrators need a mechanism to ensure destinations behave deterministically and transparently.

1.2 Approach

To implement strict isolation while maintaining usability and reducing performance costs on the end user, we created a system that utilized virtualization to provide isolation between applications. To protect the assets even if a VM is breached, we utilized a type 1 hypervisor to provide security through isolation. Whenever the user launches an application, the hypervisor will create or access a VM in the respective environment. Even if one VM is compromised, only the data assets in the breached virtual machine will be vulnerable, reducing the impact of a data breach. Additionally, this allows the other environments to continue operating normally even if one environment is compromised. However, requiring multiple VMs needs high-end hardware capabilities if hosted locally. Our VMs are hosted on a server, reducing the RAM and CPU burden on the local machine. The end user system only needs to connect to this server instead of running the VMs locally. For the remote connection infrastructure, we utilized remote desktop protocol (RDP) via Apache Guacamole [6], a browser-based remote connection client, on a dedicated server to facilitate client-server communication. For our system to be user-friendly, it mirrors a traditional desktop environment. The user's device has a desktop environment with applications to launch. When the user selects an application, the client will signal to the server that the user wants access to VM with the selected application. The server would then create or spin up a VM and provide the user with an RDP connection to the VM with the desired application. From the user's perspective, the user launches an application, and the application window appears, just like on a traditional desktop. When the user wants to close the application, the user closes the application normally, and that will terminate the RDP connection as well.

To maintain the benefits of isolation while maximizing inter-environment productivity, we provided a more usable way of creating and maintaining policy by modifying an existing SDN-based policy management tool. The tool incorporates user intent data into decisions by adding machine context labels to assist decision making in multi-machine environments. We tested the updated tool to compare it to traditional networking systems and the unmodified version of the tool using three scenarios based on real-world threat models.

To provide an efficient way to observe and identify anomalous activity, our approach encompassed the development of custom analysis tools tailored for automatic analysis of unknown assets and destinations within a virtual environment. Moreso, we streamlined the installation process of our sandbox environment through automated shell wrappers that significantly reduced user actions and simplified setup procedures. Additionally, we created custom virtual environments to optimize resource usage and minimize exposure to potential threats during analysis tasks. By integrating a specialized keylogger within our sandbox environment, we enhanced data collection by monitoring keyboard and mouse interactions, ensuring comprehensive data capture for both sandbox and GUI interactions. By running a set of experiments focusing on synthetic ransomware, real malware samples, and benign scenarios, we collected workflow metrics to analyze the utility of time series data from a sandbox environment. The collected data allowed us to understand the trends of alignment between different sensors. Further, we automated log report generation and developed visualization tools to facilitate data analysis, providing comprehensive insights into potential threats and their behavior.

1.3 Benefit Over Cost

Our system for isolating environments offers both usability and performance benefits. It provides the benefit of increased usability by mirroring the traditional desktop environment in comparison to traditional isolation approaches. Applications within the VMs seamlessly integrate onto the user's screen, providing a user-friendly experience. By replicating a traditional desktop experience, the system removes the need for end users to learn how to use VMs and VM managers, manage VM windows, and everything else that has to be known to work effectively with VMs. By following the principle of familiarity, users will spend less time learning and adjusting to the new UI. The end users never receive any indicators that they are operating in a different environment than they are used to. The result is that the user can operate their standard workflows uninterrupted, but now with increased security and asset protection.

Additionally, our system also provides the benefit of increased scalability, due to utilizing server resources. This means organizations can run VMs without being constrained by local hardware capabilities and can efficiently distribute computing resources. Because the server runs the VMs and all associated processes, that heavy computational burden no longer lies with the end user systems, which significantly reduces their required computing power.

Furthermore, our system removes the need for end users to interact directly with any VM management environments by automatically creating the appropriate RDP commands and Guacamole tunnels. This abstraction of the privileged environment out of the user workflow reduces its visibility for attacks. The automatic RDP and Guacamole actions are handled similarly to template statements, further protecting the privileged domain from outside threats by recognizing accepted inputs and rejecting everything else. This minimal client and server model removes the requirement for distributed privileged domains, as would exist if traditional VMs were used.

There are several associated costs with implementing this system design. Firstly, our design demands more computing resources than a traditional desktop environment. The financial cost of the end systems would decrease because of the lower computational needs but would be at least partially offset by the need to purchase and operate a server. Operating a server comes with many additional costs. There are the costs associated with setting up, hosting, and maintaining large server infrastructure.

Moreover, the use of remote connections may introduce increased latency in the system, affecting user experience. The ideal use of the system would be via intranet, where no cross network traffic needs to occur. If this is not feasible, the system will introduce additional network latency that would not exist in a traditional desktop or if local VMs were used.

User experience and productivity must also be balanced with network security in the form of firewall policies. Strict data protection laws like HIPPA and FERPA require compliance to avoid penalties, and these make it difficult for network admins to create policies. Our application allows for network admins to dynamically adapt policies for different machines and have different levels of policy depending on execution environment, user permissions, and data asset group. For example, there are some policies that require a universal allow or deny for all members in an organization, and some policies that are only specific to a particular VM, user, individual application and a persons role in the origination. Our application would allow the creation of further layered policies, allowing a higher level of specificity when creating controls, preventing the transmission of potentially malicious traffic. Having this level of control has significant benefits for organizations. As it is common for organizations to have 20,000 to 30,000 policies on the low end and 100,000 policies on the higher end [7]. With the large amount of policies on the firewalls, it is difficult to fully understand every policy leading to 30-50% of these policies to be "redundant or serve no purpose" [7]. These redundant and useless policy could create bugs, vulnerabilities into the network and hamper productivity. Our application will allow network admins to eliminate redundant and useless polices while reduce the overall size of the polices by 30 to 50 percent, eliminating potential bugs, vulnerabilities and increasing the organization overall productivity.

Our application also allows for easier policy creation, as network administrators can focus on how smaller groups of policies affect individual applications and their risk pools instead of modifying one large set of policies. These smaller groups will help speed up the training process of new network administrators as it will be more intuitive and easier to see how the policy is effecting the network. Our application will also help speed up training by creating a visual graph of all interconnected VMs, routers, and devices so that new and current employees can see what is connected where. The application also reduces the amount of time needed to train new employees as it allows all new policies to be tested in a sandbox environment to make sure it has the desired result. Although the application will speed up training, training will still be a cost associated with the application. Initially creating policies at the global, risk pool, role, or individual application level will have a significant time cost, as ensuring the policies are set up correctly is important to the functionality of our application. This is a one-time cost, as once the organization has policies in place, network administrators simply need to modify the smaller individual policies as needed. Changing policy is easy, as it is clear which assets or applications the policy affects and where they are implemented. An additional cost comes in the multiple enforcement points that will need to be kept synchronized in order to eliminate stale decision making, which is dangerous to an organization. Having these points will allow easy policy creation for administrators and clear enforcement for network traffic, internally between VMs, or externally to the larger internet.

IT administrators will benefit from detailed logging inside the sandbox environment, allowing them to learn what behaviors are associated with an asset or destination. The administrators will be able to see if behaviors are consistent or erratic and be able to associate them with the workflow context within the sandbox, allowing them to determine if the behaviors are consistent with a benign workflow. The sandbox collects information from 312 sensors, covering UI, file system, and network activity. IT administrators will benefit from the generated report highlighting sensor data regarding potential indicators of compromise coming from within the isolated sandbox environment and our observations will provide intrusion model designers a more useful slate of features.

IT administrators will naturally incur a file storage cost when recording sandbox sensor information. Generated reports for a 10 minute sandbox lifespan average 60.63 MB in size and a unique report must be created for each application's runtime.

1.4 Competition

Without our system for isolating environments, IT administrators would need to assemble their own set of tools to achieve asset/application isolation. Some existing systems that perform similarly to our system include Qubes OS [8], Qubes Air [9], Docker [10], and Microsoft Win32 app isolation [11]. Docker and Microsoft Win32 app isolation attempt to provide isolation by using containerization but still have the applications on the same device [11]. Docker even states that they deploy "loosely isolated environments" [10]. If a data breach were to occur, all assets would be vulnerable. Even though Qubes may provide true isolation of kernels and assets within their VMs, Qubes requires specific hardware [12], and while it mirrors a traditional desktop environment, the UI and the functions it provides are complex, forcing users to learn and adapt to Qubes' specific environment. Additionally, Qubes runs its VMs locally, requiring the user to have enough RAM and CPU throughput on the device to run it. Qubes developed Qubes Air in an attempt to offload the processing requirement, but development has been stagnant, and Qubes Air will still require high hardware requirements. Our system provides true isolation by isolating applications onto different virtual machines in a data-centric environment, reduces hardware requirements by offloading the VMs onto a server, and is user-friendly by mirroring the traditional Windows environment.

Controlling flow between isolated resources on different machines will be important to upholding the security of the system. Organizations currently rely on access control on a per-device or per-user group basis, resulting in coarse-grained policies that can only reliably work in one environment at a time. Often times, static policy creation and enforcement is the norm, causing organizations to struggle when new data privacy laws are created and all policies need to be reworked on a device-by-device or user group-by-user group basis.

Some methods for access control exist, however, they lack the flexibility of our system, especially when it comes to misclassification of information. For example, the Department of Defense along with other branches use a layered clearance system in which users must obtain clearance (e.g. secret clearance) to work with certain materials. Users below a specific clearance level cannot access materials above their level. However, many assets are classified at a higher level than needed based on related assets or work, resulting in overclassification. By the DoD's own self-admission, this approach is too conservative leading to overclassification being a major problem [13]. With access to assets usually being denied to lower clearance levels with the current approach, the DoD must invest in more expensive clearances for civilian and military personnel to work productively [14]. On the other end of the scale, underclassification can also lead to issues with data leakage.

Cisco's Identity Services Engine (ISE) is a Network Access Control (NAC) tool designed to enforce network policies at the device or user level, which can be used to deny or allow access to VMs based on predefined policies. For example having only certain VMs connect to https://www.google.com could be a policy. Devices can be automatically assigned to profiles that meet group or individual policies upon joining the network based on attributes like MAC and IP addresses and HTTP user-agent activity [15][16]. Policy violations on VMs can also be logged and the origin of the VM connection can be traced if system administrators need to dig deeper in their search. Though the Cisco approach to access control through policy implementation and automatic profile assignment is fairly fine-grained, it comes with some restrictions to profiling. For example, not all information communicated in certain packets such as DNS packets will be used during the automatic profiling process, resulting in applications being profiled incorrectly and then being assigned the wrong policies to abide by, creating access control issues. The only Cisco ISE resolution for this is to statically create endpoint profiles, a time-consuming and non-dynamic process. Many network administrators have reported having to use the CLI for many Cisco products, including Cisco ISE, due to the GUI not being user-friendly.

AppGate SDP implements a zero trust policy for access control on the application or service side rather than the endpoint device side by setting up a software-defined perimeter that only allows devices access to certain applications if they match the profile needed to view the applications. It works similarly to our solution by restricting or allowing access to certain devices in a dynamic way based on profile information. Similar to the Cisco ISE, this automatic profiling is based on network administrator policy creation for specific types of devices. AppGate SDP mainly focuses on analyzing security posture of devices before allowing them access to applications, but doesn't look at other important information, like current activity, to profile [17]. The system does not act directly on security threats as a whole when it comes to network access and can only limit a suspicious device's access to applications based on the dynamic profile adjustment. This is based on the device's security posture rather than its activity. For example, if a firewall is not enabled on a device, or if a user is on a public network, application entitlements would fall away. However, if a user downloaded a malicious file, the device could still access all applications [18].

Our solution works to secure environments at a machine, application, and user level that network administrators can interact with using an intuitive UI. It limits misclassification and promotes dynamic policy implementation through updated profiling standards while giving administrators access to graphical representations of their networks to reduce misconfigurations. The solution takes the whole environment down to application level into account to provide extremely fine-grained access control without losing sight of the larger network and policies for that.

Our solution competes with Joe Sandbox, a commercial, machine-learning-assisted malware analysis engine. Joe Sandbox enables analysts to examine program behaviors on Windows, macOS, Linux, iOS, and Android operating systems. Joe Sandbox's multi-technology malware analysis platform uses URL analysis, phishing detection, instrumentation, simulation, hardware virtualization, hybrid and graph - static and dynamic analysis methods to identify threats [19]. Joe Sandbox can automate all user behavior, including browsing a URL with a web browser, logging into an email account, or running a file with special arguments. During the sandbox's life cycle, the user is presented with live data regarding any malicious signature hits or indicators of compromise, and immediately following the interaction's conclusion, Joe Sandbox generates a detailed report of all activity that transpired in the sandbox. Joe Sandbox is used to mitigate risks that come via phishing, follows best practices by isolating untrusted assets in sandboxes, and automatically identifies untrustworthy assets. Joe Sandbox fails to meet IT administrators' needs with a way to identify whether sensor information is aligned with a benign workflow.

Another competitor of our solution is Bromium, a software that utilizes hardware enforced virtualization to protect end-user data from malware [20]. Its key purpose is to virtualize processes that are launched by the end-user or application in order to keep the operating system separate from potential threats. The end-user is unaware of the initiation of Bromium's micro-VMs, which are lightweight sandbox environments, when they interact with an unknown link or application that often stems from phishing attacks [21]. The only time an end-user might be notified is if they are alerted of potential malware. While Bromium is robust at containing threats, mitigating risks that come via phishing, follows best security practices, and identifies untrustworthy assets without requiring user intervention, Bromium fails to meet IT administrators' need in identifying whether sensor information is aligned with a benign workflow.

Various detection technologies play crucial roles in identifying and mitigating threats. These technologies include Intrusion Detection Systems (IDS), antivirus software, and Security Information and Event Management (SIEM) systems.

An IDS is a valuable tool for monitoring and detecting deviations in network behavior. They come in different types, such as signature-based and anomaly detection systems. While they offer comprehensive monitoring, traditional IDS variants may face challenges in detecting insider threats and novel attacks, particularly in switched networks. Human intervention becomes necessary in dealing with determined adversaries.

Antivirus software provides real-time protection against malware by scanning for predefined patterns of known attacks. While effective, they can be resource-intensive and less effective against new threats. Additionally, they may face compatibility issues and require frequent updates, adding complexity to cybersecurity maintenance.

SIEM systems serve as central intelligence hubs within Security Operations Centers (SOCs). They collect and standardize data from various sources for security analysis, aiding in threat detection and response. Leading solutions like Splunk and IBM Security QRadar offer comprehensive log aggregation and analysis capabilities. However, SIEMs come with challenges such as complex rule management, lack of contextual information, and high costs associated with deployment and maintenance. They also do not factor UI activity data into their decision making processes that may help fulfill IT administrator's need in identifying whether sensor information is aligned with a benign workflow.

1.5 Associated Research Questions

In order to direct research our efforts, we develop the following questions for each sub-goal associated with our research.

In implementing strict isolation while maintaining usability and reducing performance costs on the end user, we aimed to answer the following questions:

RQ1. How does VM hot loading impact the startup times of applications in isolated systems?

RQ2. How much wasted screen space is incurred as a cost of using nested windows?

RQ3. What are the number of steps to access a VM with our approach when compared to other VMMs?

RQ4. What computing costs are incurred due to using our virtualized system on the client and server-side?

In enforcing granular network access control and maintaining policy scalability, we aimed to answer the following question:

RQ5. To what extent does incorporating VM contextual labels and user intent data through UI monitoring benefit firewall accuracy and policy scalability?

In providing an efficient way to observe and identify anomalous activity, we aimed to answer the following questions:

RQ6. With the sensors available in the sandbox, what amount of information (from time or data points) is needed to distinguish the different scenarios? Which sensors are the ones collecting that information?

RQ7. Can our report distinguish aligned and non-aligned UI activity and background micro-activity during a high-level (macro) event?

1.6 Contributions

In exploring these questions, we make the following contributions:

- **Creation of Remote Computing Server and Thin Client Infrastructure:** We explored a system to achieve security by isolation, while maintaining performance and usability. We utilized KVM [22] to facilitate isolation by virtualization. We minimized performance costs by utilizing remote server resources. Virtual machines are managed through a REST API and "hot loaded" by pre-loading applications and suspending/resuming the virtual environments. We established communication with the virtual environments via Remote Desktop Protocol [23], providing access to remote virtual machines through an Apache Guacamole [6] web browser interface that mirrors a traditional desktop interface. The system can be used to spin up an application in isolation on a remote virtual machine with the click of a button masked as a "desktop icon."

- **Evaluation of System Usability and Performance:** After creating the system described above, we evaluate its usability and performance by comparing it with a traditional desktop environment and a traditional VM environment. We examine the startup times, wasted screen space, VM creation process, and CPU and RAM usage of the systems. We test the systems with a variety of scenarios to see how different computational loads and simulated workflows would differ across the various systems.

- **Modification of Harbinger to Include Machine Context Labels in Decision-Making Process:** To increase policy scalability without sacrificing firewall accuracy in multi-machine distributed environment, the team created a text file that contains context label data and modified Harbinger's processing mechanisms on the controller side to take in, understand, and make decisions based on the additional information. The modified version of Harbinger was called Harbinger+. The team created infrastructure and scenarios to test accuracy and policy scalability of the Harbinger+ tool against traditional network defense systems and the unmodified version of Harbinger.

- **Custom Analysis and Dependency Packages:** To tailor Cuckoo Sandbox to our project's needs, we leveraged its modular design to create custom analysis and dependency packages. Our contributions include custom analysis and dependency packages, specifically designed to support the automatic analysis of Microsoft Word .doc, .docx, and .docm samples within the virtual machine. This customization allows for a more focused and efficient analysis of potential threats.

- **Streamlined Cuckoo Installation Process:** Installing and configuring a Cuckoo Sandbox environment typically involves a series of complex steps, including dependency installations, user inputs, and complex configuration instructions. Recognizing the need for efficiency, we developed shell wrappers that significantly streamline the entire Cuckoo installation process. These scripts handle downloading required libraries, setting permissions, creating the Python virtual environment for Cuckoo, generating virtual machines, editing Cuckoo's configuration files, and providing global routing for all sandboxes. To optimize the VM creation process, our Bash script wrapper allows users to specify desired features before initialization, reducing the required user actions from 71 to 4.

- **Custom Virtual Environments:** To address end-user needs effectively, our solution supports the creation of custom and configurable virtual environments. The number of virtual environments is only limited by server hardware. These virtual machines contain a minimal working set of applications necessary to support specific operations. For instance, a virtual machine designed for Word file analysis includes only LibreOffice and additional logging tools, minimizing the risk of exposure to potential threats.

- **Keylogger Integration:** In parallel with Cuckoo Sandbox, a specialized Keylogger played a pivotal role in enhancing the data collection process alongside Cuckoo Sandbox. Implemented in Python3, the Keylogger monitored keyboard interactions and mouse clicks, providing a detailed log file with timestamps. This log file proved instrumental in visualizing end-user UI interactions during experimental runs.

- **Log Report Automation and Visualization Scripts:** To create visualizations, we developed a Python scripts tailored to our research objectives. The script was dedicated to creating graph visualizations using time series data, offering a dynamic representation of temporal aspects within the collected data. This visualization tool enhanced our ability to interpret patterns over time, contributing to an analysis of alignment.

2 Background

In order to enhance security of our system against potential attacks and breaches while reducing the likelihood of future compromises, we needed to address elements and interactions that are found within a traditional computer system and then craft our research around these potential areas of compromise. To start, we first investigated various tools and services such as virtual machines and sandboxes that would support our vision of a secure, isolation-centric system while keeping performance at an accessible level and allowing for high system scalability. We then explored numerous methods of detection strategies, such as intrusion detection systems and sandbox reports, that would help identify risk within the system and provide valuable data to analysts that helps decrease the risk of compromise. Finally, we analyzed instruments related to network security such as network access control policies and firewalls that would maintain protection over the system's network traffic.

2.1 Security by Isolation

The focus on the start of our research was directed towards using virtual machines to virtualize processes; in the event of compromise, the damage would be self-contained within the process, and the rest of the system can theoretically remain functional. To address this, we first started researching virtual machine and hypervisor design, as these tools provided features that were closely related to our isolation goals. We also investigated Qubes OS in order to obtain a better sense of how our system should be structured. Finally, we looked into various sandboxes that would examine malware and other forms of compromise; the information provided by these sandboxes would help analysts determine what occurred within the compromise and how the system could be further secured in the event of a similar attack.

2.1.1 Virtual Machines

Currently, one of the most popular methods to achieve security by isolation is through running virtual machines (VMs). A VM is a virtualized computing environment that virtually emulates its own CPU, memory, network interface, and storage [24]. VMs have gained popularity due to their ability to be efficiently created, duplicated, saved, and deleted, allowing them to evaluate suspicious software while protecting data. If the tested piece of software is malicious, the isolation by virtualization contains any incurred damage. If this was done in a non-virtualized and non-isolated environment, the entirety of the computer, from the hardware to the operating system, could be at risk. The isolation prevents the spread of the malware to the rest of the computer or the operating system by confining the malicious product within a VM [25]. Users would only need to replace the VM instead of the entire computer, which would potentially save them thousands of dollars. The VM architecture protects the local hardware by using virtualized hardware instead [26]. Hypervisors are the software or firmware that creates VMs and manages their interaction with local hardware or OS. They are the foundational piece of technology to achieve security by isolation.

2.1.2 Hypervisors

There are two main types of hypervisors: Type 1 hypervisors, like KVM and Xen, and Type 2 hypervisors, like Oracle VirtualBox. Type 1 hypervisors are known as "bare metal" or "native" hypervisors; they work directly with the hardware of the host. In addition to interfacing with the hardware, they also fulfill the role of Virtual Machine Manager, or VMM [27]. While the user does not directly interact with a Type 1 Hypervisor like they would a native operating system, they do interact with a management tool that controls the hypervisor. This can take the form of a terminal, Graphical User Interface (GUI), or one of the virtual machines that the hypervisor creates. Type 1 hypervisors are also more scalable than Type 2 hypervisors, meaning that a given allotment on computing resources could concurrently run more Type 1 hypervisors than Type 2 hypervisors. Because they do not run on top of a native OS, they have lower overhead for CPU, disk I/O, and memory usage. Lastly, because Type 1 hypervisors do not rely on a native host OS, they have a smaller Trusted Computing Base attack surface and are therefore not susceptible to attacks that take advantage of that OS.

Type 2 hypervisors are convenient and widely used virtualization solutions, as they are the backbone for traditional Virtual Machines. Unlike their Type 1 counterparts, Type 2 hypervisors run on top of a native operating system rather than directly on the hardware. This design choice makes them more user-friendly, with accessible graphical user interfaces, abundant public documentation, and large user communities. Type 2 hypervisors provide an excellent environment for development, enabling users to test various operating systems while ensuring the safety of their local system.

Trusted Computing Bases and Trusted Domains. In computers, the set of hardware and software components that are critical for system security are called the Trusted Computing Base (TCB). The TCB is responsible for enforcing security policies and is vital for establishing and maintaining security throughout the life of a system [28]. It is important to clarify that being a part of a TCB does not guarantee that those components are actually secure; it just means that such components are necessary for the security of the system [29]. A smaller TCB with fewer components generally means that it will be easier to secure than a large and complex TCB. Having fewer items critical to system security means that fewer things can go wrong and break system security. By minimizing our TCB, we can directly decrease the number of attack paths against our system.

In addition to TCBs, most VMs are run through a VM manager, or VMM. They are the root of trust for VM architecture, as they have full control over all of the VMs that they manage [30, 31]. In Docker for example, there is a shared kernel that contains these escalated privileges and manages the otherwise separated Docker containers. If an attacker breaks into a single Docker container, the original user can erase, discard, and replace that container in no time. However, if an attacker gains access to a shell with the shared Docker kernel, they gain total power over all Docker containers. The issue with Docker's root of trust system in this example is that Docker is not designed to protect the container management environment. It is made to keep the containers secure and discardable, but places little emphasis on the environment that manages those containers, hence the "loose isolation" that they claim [10].

Xen and KVM. There are many Type 1 hypervisors, including VMware vSphere, Microsoft Hyper-V, Xen, Kernel-based Virtual Machine (KVM), Oracle VM Server, Citrix XenServer, and Proxmox Virtual Environment [32]. Because our project is being built from the ground up, we want to start with a basic hypervisor that does not have a lot of proprietary features built into it. That left the choice between the Xen hypervisor and KVM. While KVM is labeled as a Type 1 hypervisor because it operates directly with the hardware, it has many features of a Type 2 hypervisor. Being Kernel-based means that KVM still relies on being integrated with a Kernel and integration with a Host OS. As a result of this, KVM is less scalable than Xen [33], as it reintroduces some of the Type 2 hypervisor overhead resource requirement. The reason that it is labeled as a Type 1 hypervisor is because once the VMs are created, KVM creates a tunnel through the OS layer so the VMs can operate on the same level as the host OS, directly above the hardware layer. Xen is a more traditional Type 1 hypervisor and does not sit on top of a host OS [34]. When it comes to protecting individual VMs from extreme resource consumption from a single VM, both Xen and KVM perform well [33].

Both KVM and Xen are recognized as being useful security tools. Xen employs various features like "guest separation, privileged access rights, tiny code base and operating system isolation" [35] to enhance system security.

Xen has an additional TCB for the hardware, known as domain 0 or dom0. Dom0 is the only software that is able to interact with the hardware. Dom0 is not able to be directly interacted with by the user. The user can interact with other domains, and those domains can in turn talk to dom0. By preventing anything else from talking with the hardware, Xen makes it significantly more difficult for attackers to take command of multiple VMs at once or infiltrate the system hardware [36] [35].

KVM "inherits the standard Linux security model to offer separation and resource control" [35]. Because KVM VMs act like standard Linux processes, using a more secure Linux distribution like AppArmor and SELinux derivatives will increase the security of KVM.

2.1.3 Qubes OS

Qubes OS is a security-oriented operating system that utilizes Xen virtualization to create isolated virtual machines for both applications and hardware drivers [37]. The goal of the Qubes OS project is to solve security vulnerabilities due to the lack of isolation between programs in modern operating systems. If one program on a user's machine gets compromised, modern OSes generally fail to protect the user's other applications and assets from also being compromised due to a lack of isolation. Qubes domains are lightweight VMs with strong isolation between them. Qubes accomplishes this strong isolation by limiting the amount of code in the Trusted Computing Base (TCB), thereby reducing the attack footprint.

Qubes has two main classes of VMs: AppVMs (user-facing) and system-level VMs. AppVMs are user-facing VMs used to run the user's applications [38]. AppVMs run applications utilizing another OS (Linux, Windows, etc.) instead of the native QubesOS. The AppVMs have the same functionality as a computer natively running the AppVM OS. The user would typically have multiple AppVMs for different types of tasks. For example, a user might have a "work" AppVM and a "personal" AppVM. By separating these two tasks into different AppVMs, if the "work" VM was compromised, the "personal" VM and all of its assets would remain secured.

The second class of VM are the hardware/system-level VMs. The user does not directly interact with these VMs. Instead, they are used to isolate system-level components such as the networking, storage, and GUI subsystems [38]. Qubes isolates these subsystems to prevent complete system compromise if one of the subsystems is breached by the attacker. One common example of this is the network subsystem. In a normal OS, if the network subsystem is compromised, all applications and data assets on that machine will/would be also compromised. Qubes' network domain VMs provide isolation of the network VM, so only AppVMs connected to that network VM will be compromised.

Qubes virtualized GUI is one of the main benefits of the system. By virtualizing the GUI, Qubes is able to provide one desktop interface for interacting with all of the AppVMs seamlessly, while also maintaining isolation between them. To accomplish this, Qubes has created a Qubes GUI protocol. Each AppVM has a Qubes GUI agent running within it, which connects to a local graphics and user input server (Xorg) in the AppVM. The GUI agent's job is to pass user input (keyboard, mouse) and the memory location of a framebuffer to a GUI daemon in dom0. The GUI agent is connected to a GUI daemon in dom0 over a Xen virtual memory channel (vchan). The GUI daemon in dom0's job is to access the framebuffer and relay user input and graphical information to the dom0 Xorg server to paint the desktop image [39].

From a security perspective, this solution effectively utilizes compartmentalization to finely control communication between AppVMs and dom0 through the Qubes GUI protocol. However the GUI protocol still relies on dom0 level privileges and exposes the memory location of the framebuffers, increasing vectors for potential attacks.

To combat this, Qubes has released an experimental feature, the GUI domain [40]. The goal of the GUI domain is to decouple GUI and dom0. In this new solution AppVMs will communicate with a new GUI VM using an updated GUI protocol. Instead of passing the addresses of the frame buffers, the new GUI protocol will use Xen's grant tables to grant access to the framebuffers' memory page instead of directly exposing the memory address, and utilizing dom0 level access. The GUI VM then communicates with dom0 using Qubes GUI protocol, where dom0 will display a full screen proxy of the GUI domain's graphical server. The GUI domain abstraction also allows for different methods of graphical rendering, like VNC. Using VNC circumvents exposing dom0 by sending the graphical data from the GUI domain to a remote server instead, which removes the need for dom0 to paint the final full screen proxy from the GUI domain.

Like Qubes, our approach aimed to provide a seamless user experience while maintaining strict isolation. However, Qubes requires specific hardware capabilities that make it inaccessible to some clients. Our solution aimed to circumvent hardware constraints by implementing a web-based user interface with a remote server for hosting the virtual environments. The experimental Qubes GUI domain provides an alternative solution to this issue, which could be explored as future work as detailed in Section 5.1.

2.1.4 Sandboxes

Sandboxing tools isolate a virtual environment to contain malicious activity and record event details regarding the overall workflow. This section analyzes four prominent sandboxes: Cuckoo Sandbox, HP Bromium, Joe Sandbox, and GFI Sandbox, and offers a comparative assessment of their features and capabilities. In our study, we chose to analyze these four sandbox solutions due to their recognized expertise in automated malware analysis. They each employ different methodologies and technologies, offering a diverse spectrum of capabilities for detecting suspicious activities. Our objective is to conduct an in-depth examination of these sandboxes to gain a comprehensive understanding of their functionalities and limitations. Table 1 presents a breakdown of features offered by these sandboxes, facilitating a comprehensive understanding of their functionalities and limitations.

Table 1: Comparison of Bromium, Cuckoo Sandbox, Joe Sandbox, and GFI Sandbox

Feature	Cuckoo	Bromium	Joe	GFI
Virtualization Approach	Virtualization	Micro-virtualization	Custom hypervisor	Dynamic analysis
Malware Isolation	Yes	Yes	Yes	Yes
Detection Mechanism	Signatures, VirusTotal, Static, Dropped, Network, etc.	Live Attack Visualization and Analysis (LAVA)	Signatures, Mitre Attack Matrix, Screenshots, etc	Behavior-based analysis, API hooking, DLL injection
Report Contents	Signatures, behavior analysis, memory dump results	Micro-VM behavior, Live Attack Visualization, analysis results	Comprehensive analysis results, screenshots, etc.	Automated, machine-readable report
Ease of Use	User-friendly features, JSON format reports	Learning curve, support from IT needed	Easy to use, configurable sandbox environment	Lacks scalability, not suited for sandbox-detection
Resource Intensiveness	Impact on CPU and hardware resources	Resource-intensive on older hardware	May take up to 10 minutes for larger files	Lightweight and efficient
Configuration Complexity	Configuration options, network settings, may be complex	May be complex, requires IT administrator's expertise	Configurable platform, easy to use web portal	Scalability issues, not suited for sandbox-detection
Compatibility	Primarily designed for Windows, supports multiple types	Primarily designed for Windows, supports various file types	Supports most common enterprise OS	Built for the WIN32 family

Cuckoo Sandbox. In our research, we leverage Cuckoo Sandbox [41], an open-source, automated malware analysis system widely utilized in the cybersecurity domain. It provides valuable insights into the behavior of suspicious files or URLs. Users submit suspicious files or URLs to Cuckoo Sandbox for analysis. Cuckoo Sandbox creates an isolated virtual environment to execute the submitted sample. During analysis, it logs various aspects of a file's execution, including changes to files and folders, memory dumps, network traffic, processes, API calls, and more. After the analysis is complete, Cuckoo generates a detailed report that includes information about the sample's behavior, any indicators of compromise, network traffic logs, screenshots, and any known signatures.

Cuckoo Sandbox gathers a wide array of data related to malware behavior. It categorizes its analysis into six main sections:

- Signatures: Searches for user-defined malware patterns.
- Static: Analyzes executable files, providing version, sections, resources, and library info.
- Dropped: Shows files created by the malware and later deleted, including temporary files.

- Network: Monitors real-time network traffic, capturing IP addresses, port numbers, DNS, and more.
- Behavior: Logs and interprets process behavior, detecting anomalies and summarizing activities.
- Volatility: Displays results of memory dump analysis for insights into malware behavior [42].

This comprehensive approach ensures that analysts have access to a rich data set for threat assessment. However, the accuracy of this data can sometimes be influenced by the presence or absence of specific malware signatures, leading to variations in threat scores [43]. Storage constraints may arise due to the extensive logs and reports generated during analysis, with file sizes ranging from 1MB to 1GB per analyzed file [42]. Performance is a critical factor when evaluating Cuckoo Sandbox. Its ability to analyze multiple suspicious files concurrently within isolated environments can significantly impact CPU and hardware resources [44]. Improved performance not only reduces the hardware requirements but also minimizes latency, ensuring quicker results for security analysts [45].

Cuckoo Sandbox is designed for compatibility with Windows, as it primarily targets Windows executables [46]. However, it also supports a range of other file types, including PDF documents, Microsoft Office files, URLs, Python scripts, and more [47]. This cross-platform compatibility makes it a versatile tool for analyzing a wide variety of potential threats.

In terms of ease of use, Cuckoo Sandbox aims to provide user-friendly features. The generated reports are easily parse-able in JSON format, simplifying the interpretation of analysis results [42]. While it offers comprehensive capabilities, it strives to present data in a clear and organized manner, facilitating the work of cybersecurity professionals.

The installation process is often considered cryptic and confusing, especially for first-time users [48]. Cuckoo Sandbox's user interface has been criticized for its complexity and lack of intuitiveness, which can make it challenging for users to navigate effectively. The lack of comprehensive documentation and support resources further adds to the challenges faced by users in troubleshooting and resolving issues on their own [49].

Cuckoo Sandbox is used in our research because it is robust automated malware analysis system, excelling in logging, comprehensiveness, and compatibility while navigating constraints and performance considerations. Its user-friendly reports and extensive data collection capabilities make it a formidable contender in the realm of cybersecurity tools. However, the accuracy and completeness of its logs are areas that warrant further investigation. Additionally, Cuckoo Sandbox encounters issues with its cryptic installation process and a complex user interface.

Bromium. HP Bromium was a software that utilized hardware-enforced virtualization to protect end-user data while aiming to increase security within the enterprise [50]. Its key purpose was to not only virtualize applications, but also virtualize every process that is launched by a user or application for the benefit of malware prevention. Bromium excelled in malware isolation, ease of use, secure file technology, and micro-virtualization. By isolating and containing potential threats within micro-VMs, Bromium prevented malware attacks that attempt to compromise sensitive data.

Bromium's micro-virtualization concept was similar to the concept of virtualization—an independent entity with its own dedicated CPU, memory, and storage—only it virtualized individual processes. This means the micro-VMs were given task specific access to networks or files on a "need to know" basis. The micro-VM did not have access to every library on the local machine, but only the ones necessary for the process to run [51]. End-users, for instance, would trigger a micro-VM for every Chrome tab, which may have required the use of specific system files. These files that the micro-VM had access to were part of a "copy on write" feature, which were cloned system resources and data so that the original copy was not infected by potential malware.

Bromium also knew if a micro-VM was under attack via inspection. If an end-user visited a website and encountered a redirect or phishing attempt, Bromium employed what is called a Live Attack Visualization and Analysis (LAVA). LAVA uses the behavioral signature of the attack to determine if it should shut down the VM and notify the end-user before the compromise occurs. Even sophisticated malware attacks that utilize polymorphism, as well as rootkits and boot-kits, are accounted for [51].

While Bromium excelled in micro-virtualization, the approach could be resource intensive on older hardware. Running hundreds of micro-VMs lead to performance issues. When a micro-VM was destroyed, Bromium faced another issue: data persistence. Bromium discarded the contents of micro-VMs after tasks were completed or when tabs were closed. This could be inconvenient for users who needed to access or replicate certain (non-malicious) actions on their local machine. Furthermore, Bromium did not contain any time series graphs of sensor data that could help IT administrators prevent future attacks.

Bromium configuration could also be complex; network isolation customization required an IT administrator's expertise [52]. Organizations with many end-users needed to invest more time to properly set up and manage configurations. While this learning curve resulted in fewer ransomware attacks, it came at the cost of needing more time and support from IT.

Joe Sandbox. Joe Sandbox is a proprietary, machine-learning-assisted malware analysis engine that enables analysts to examine program behaviors on Windows, macOS, Linux, iOS, and Android operating systems. Joe Sandbox's multi-technology malware analysis platform uses URL analysis, phishing detection, instrumentation, simulation, hardware virtualization, hybrid and graph - static and dynamic analysis methods to identify threats [19]. When a user instantiates a new Joe Sandbox environment, they may select the base operating system and network constraints and upload the media to be analyzed. Joe Sandbox configurable platform can automate all user behavior including browsing a URL with a web browser, logging into an email account, or running a file with special arguments. During the sandbox's life cycle, the user is presented with live data regarding any malicious signature hits or indicators of compromise and, immediately following the interaction's conclusion, Joe Sandbox generates an in-depth report of all activity that transpired in the sandbox.

Joe Sandbox outputs malware analysis reports that are extremely thorough in the information they capture. Report content from a Windows 10 sandbox environment includes:

- Process tree containing information about all processes in the sandbox's lifetime and the parent-child relationships of each
- The contents of any malware configuration files found
- List of user-provided Yara, Sigma, and Snort detection signatures that matched
- List of Joe Sandbox signatures that matched
- Screenshots taken periodically throughout the sandbox's lifetime
- Domains, URLs and IPs that the environment tried to connect with
- Information on all created or dropped files, including a link to download each
- Network Behavior, including DNS queries and answers as well as TCP, UDP, and HTML packets sent and received
- CPU and memory usage by each program in the sandbox
- System behavior [53].

Although Joe Sandbox is a tool designed for malware analysts, the logs are comprehensive to capture everything in the sandbox's lifetime at an environment-wide level. If a user wanted to add to the list of malware signatures that Joe Sandbox is looking for, the platform supports adding custom Yara, Sigma, and Snort rules.

Joe Sandbox utilizes a custom hypervisor which does not derive from virtualization solutions, such as XEN and KVM. Due to this independence, Joe Sandbox Hypervisor can run on any device, including both virtual and physical machines. The Joe Sandbox Hypervisor analyzes malware with native speed and doesn't introduce any latency or delays [54]. While Joe Sandbox's developer, Joe Security, claims their code has been optimized, loading files for analysis in live interaction mode, the scanning process, and the analysis process may take up to 10 minutes for larger files [55].

The Joe Sandbox web portal is easy to use and configuring a sandbox environment to the user's liking is straightforward. Once the sandbox is instantiated, Joe Sandbox's automated user behavior tool begins interacting with the file sample and the user may step in at any point to interact with the sandbox. While the sandbox is live, users are displayed real time Yara, Sigma, behavior signatures, and indicators of compromise flagged by Joe Sandbox [56].

While Joe Sandbox provides many appealing features for malware analysts, there are some factors that require improving. For one, the generated report's exhaustive features include just about everything an analyst would need to identify malicious behavior, but sometimes an analyst only cares to observe information about a given attribute. In this case, Joe Sandbox gives critical processing time to capturing extraneous details that the analyst has no interest in. Joe Sandbox provides no option during the sandbox initial configuration stage for the user to declare which features they are interested in analyzing to limit the number of unneeded attributes in the final report. Additionally, Joe Sandbox's price-by-quote model comes with a hefty price tag for organizations wanting to conduct large quantities of analyses [57].

Joe Sandbox provides a largely configurable sandbox environment that can be quickly spun up to analyze whatever sample media an analyst provides. Upon a sandbox's conclusion, users are presented with a comprehensive report that covers a wide array of system behaviors at the environment level with high precision. However, the analysis' scope is not configurable and the software is locked behind a subscription model which may make the product inaccessible for smaller enterprises.

GFI Sandbox. GFI Sandbox, previously known as CWSandbox, is a dynamic malware analysis tool focusing on automation, effectiveness, and correctness. The tool is built for the WIN32 family of operating systems. GFI Sandbox uses behavior-based analysis, which lets analysts monitor relevant system calls and generate an automated, machine-readable report that provides information about system events and changes [58]. Behavior-based analysis combines dynamic malware analysis, API hooking, and DLL injection. Dynamic malware analysis observes malware behavior and analyzes it in a sandbox, showing changes to the system and network traffic generated. API hooking is used to access system resources, which allows for dynamic analysis if relevant API calls and parameters are monitored. DLL injection allows API hooking to be implemented in a reusable way, essentially enabling and empowering API hooking [59]. These different methods combine to allow GFI Sandbox to have in-depth malware analysis and true automation.

GFI Sandbox's dynamic threat analysis provides information on how applications execute, monitors system changes, tracks network traffic, and identifies malicious actions. In addition, this process can be done with true automation, meaning it can simulate user interactions with malware and log and analyze the resulting activities automatically. This significantly reduces the analysis time required. Using the analyses created, it compares the similarities and differences in malware's behavior, giving insight into understanding threats [60]. GFI Sandbox does not require emulation or virtualization, but rather hosts and contains the malware in a controlled environment. This allows for GFI sandbox to be lightweight and efficient at analysis. While GFI Sandbox has many good aspects, there are some areas for improvement. GFI Sandbox lacks scalability, which is a somewhat common problem for sandboxes. In addition to lack of scalability, the sandbox is incapable of evading sandbox detection from advanced malware.

2.2 Remote Connection Technologies

Virtual machines (VMs) require a substantial amount of resources, especially when multiple are running concurrently. To give our system for isolation accessibility and scalability, we wanted our approach to provide high performance and usability to clients regardless of the client device's capabilities. One solution was hosting the system and its virtual machines on a central server, which would serve the system resources remotely to the client. In order to address client/server communication, we first looked into popular remote connection protocols such as RDP and VNC that would provide system communication and compared them against one another. We also explored Apache Guacamole as a tool that would implement these remote connection protocols to provide client interaction with our system.

2.2.1 Remote Desktop Protocol

Remote Desktop Protocol, or RDP for short, is a popular remote connection protocol developed by Microsoft. RDP allows users to communicate with a Windows server from a remote client through a secure network connection. Inputs are transferred to the server, and display data is transferred back based on connection-oriented transport layer protocols [61]. In an RDP session, the client first establishes a two-way handshake with the server using RSA public keys before transmitting data. The data is then interpreted by the server and sent back to the client. RDP is useful for remote workers that need to access assets or applications on Windows servers within the company's network. Some companies even make use of the technology to provide access to virtual learning laboratories to multiple users [62]. Even universities use RDP to develop and deploy multi-user remote access virtualization systems for classes [63]. Although RDP is used for both enterprise and educational purposes, RDP has vulnerabilities that can be exploited by attacks, mainly password guessing attacks and man-in-the-middle attacks [63].

2.2.2 Virtual Network Computing

Another popular remote connection protocol is VNC, or virtual network computing. VNC is “an ultra-thin client system based on a simple display protocol that is platform independent... achieving mobile computing without requiring the user to carry any hardware” [64]. Similarly to RDP, it allows a user to access and operate a computer environment hosted on a server with a graphical user interface with keyboard and mouse control, providing full remote functionality. Numerous universities and institutions utilize VNC to provide virtual laboratories and other learning environments for E-Learning and E-Education [65, 66]. There have also been extensions for VNC to allow for synchronous collaboration with various access policies [67].

VNC works by utilizing the remote framebuffer protocol (RFB). RFB is designed to remotely update the frame buffer on a server’s interface. VNC builds off this to display the server’s interface on the client’s machine by sending regular updates from the client to the VNC server, which processes the updates and sends the results back to the client. Each update sent to the server contains pixel placement and input data that tells the hardware what to draw and what to interact with [64]. Because of this, VNC can be easy to implement and operate on a variety of hardware and systems, which indirectly provides a multi-platform focused design. However, the wider compatibility requires a larger bandwidth to use effectively in comparison to RDP. VNC is also “demand-driven by the client,” which in short means that “the slower the client and the network, the lower the rate of updates” [64]. Because of this, it is crucial to have a good, reliable network connection to provide a smooth experience.

With the introduction of Java, a client can establish a connection to a VNC server through a web browser [64]. Services such as Apache Guacamole can take advantage of this by creating clientless VNC services; the term clientless in this case means that the client does not need to have special software installed on their end to establish a VNC connection [6].

Similar to RDP, VNC also has vulnerabilities, especially due to its authentication-less aspect. Due to this, it is recommended to wrap SSH within a VNC configuration, since SSH implements industry standard encryption algorithms to protect traffic and standard hashing algorithms to ensure data integrity [68].

2.2.3 Apache Guacamole

Apache Guacamole is “a clientless remote desktop gateway [that] supports standard protocols like VNC, RDP, and SSH” [6]. Apache Guacamole is considered clientless because the client does not need to have special software installed on their end to establish a VNC connection [6]. Guacamole’s infrastructure allows a user to remotely connect to multiple servers and virtual machines all through its web browser. This service would be extremely beneficial towards hosting multiple environments remotely, as it allows a user to seamlessly transition between multiple virtual machines in the same window and provides support for multiple types of connection protocols. It also comes stocked with a plethora of security configurations, such as multi-factor authentication and signed certificate support [6].

Contrary to how it may appear, Guacamole does not natively provide standard remote connection protocol services. Rather, it utilizes the Guacamole protocol; this protocol is a custom-built remote display renderer and event handler [69]. Guacamole uses a daemon process called the Guacamole Protocol Daemon Process, or guacd for short, that acts as a translator between the Guacamole protocol and the desired remote protocol service, which are also known as "client plugins". Guacd first listens for any incoming TCP requests coming from the Guacamole web application that is being hosted on the desired server. Using some context clues from the Guacamole protocol, it determines which remote service should be used and what arguments it needs, bridging the protocols together. Upon completion, the client plugin is given full communication between itself and the web application until it is terminated [69].

It is important to mention that since Guacamole does not natively support remote desktop protocols, the configuration is entirely dependent on the client plugins that the server has installed on its system. For example, if you do not have an SSH client plugin, such as OpenSSH, nor an RDP client plugin, such as FreeRDP, on your server, then the Guacamole configuration for that server will not support those protocols [69].

Apache Guacamole supports 5 types of connection protocols: telnet, SSH, RDP, VNC, and Kubernetes, providing us high configurability on which protocols are allowed on the server running Guacamole. Other supported security features such as MFA and certificate validation can strengthen the authorization and non-repudiation of our remote systems, which in turn maintains confidentiality in our environment [69]. Upon successful installation, Guacamole provides an easy-to-use admin portal that can be used to add client connections, each with their own customizable userbase. This, combined with Guacamole's hardware compatibility, provides easy-to-implement options for scalability and security risk management [69]. Adobe Guacamole also has API functionality, meaning that we could implement custom-built Guacamole integrations into our system. The API is made up of a C and Java library that builds core Guacamole infrastructure to communicate with the Guacamole protocol and guacd as well as a JavaScript library that is used for Guacamole component interfacing [70].

The features that Guacamole provides are significant; however, a large quantity of machines can cause the dashboard to be cluttered, and the technology is limited when it comes to multi-user stability. Without external tools, Guacamole can cause two or more users to be running the same machine at the same time without change management, making it susceptible to race conditions [71]. Researchers have been able to combat this by developing external administration tools to help efficiently and safely manage resources for each user [71, 72].

Today, Apache Guacamole is becoming a popular remote connection solution for many researchers and professors due to its ease of accessibility. For example, a cybersecurity research team worked with Guacamole to develop a security course that utilized Docker containers to run security challenges for students to partake in. This system was tested on a single server to a class of 250 people, where it was deemed to be both effective and reliable [72]. In order to increase the accessibility of running special software in their electrical engineering program, a university was able to implement Guacamole into their computer lab subnet to provide students the ability to remotely access workstations with higher hardware specifications. This allowed students to work with computationally intensive software without having to buy a better computer to run it [71]. Since this capability is built into modern hypervisors, we did not explicitly research if such optimizations were necessary.

The main reason that we investigated Guacamole was due to its design aligning with how we wanted to address accessibility and scalability in our system for isolating environments. There were several features that stood out to us as helping us streamline and customize our system development. First, Guacamole's support towards a wide range of remote connection protocols would allow us to better suit the needs of the operating system and the applications contained within each virtual machine hosted on the server. Second, its well-documented API would allow us more developmental freedom and easier integration with our codebase. Finally, its security settings offered high customization that we could utilize to further secure our accessibility to our system.

2.3 Detection

In the realm of cybersecurity, while Intrusion Detection Systems (IDS), antivirus software, and Security Information and Event Management (SIEM) solutions are essential, they have limitations: IDS systems lack capability to detect unknown attacks or suffer from high false positives, antivirus software struggle with zero-day attacks, and complex SIEM solutions have difficult maintenance. It is important to consider these drawbacks and explore alternative approaches like sandbox environments to enhance cybersecurity strategies.

2.3.1 Intrusion Detection Systems

When it comes to identifying insider threats, IDS serve as valuable tools with strengths in comprehensive monitoring and early deviation detection. However, they do have limitations, such as potential delays under heavy loads and challenges in detecting novel attacks, particularly in switched networks. Despite their classification into various types, including signature-based and anomaly detection systems, IDS alone may not suffice against determined adversaries. Human intervention becomes necessary in such cases.

Researchers' work sheds light on the base-rate fallacy and the inherent difficulty in intrusion detection. They highlight the challenge of distinguishing between normal and abnormal behavior accurately, especially in dynamic environments where patterns constantly evolve [73].

Signature-based IDS rely on predefined patterns of known attacks, while anomaly detection systems identify deviations from normal behavior. However, these methods may not effectively detect insider threats, as the behavior of legitimate users can resemble malicious activity. Additionally, IDS may face difficulties in switched networks where traffic is directed only to its intended recipient, hindering comprehensive monitoring [74].

2.3.2 Antivirus

Antivirus software is a critical defense against malware, providing real-time protection through signature-based scans. While effective in detecting and eradicating various malware types with user-friendly interfaces and proactive updates, antivirus has limitations such as resource intensiveness, diminished effectiveness against new threats, and potential compatibility issues. These factors, coupled with the need for frequent updates and possible disruptions through pop-up notifications, highlight the complexities in maintaining robust cybersecurity.

2.3.3 Security Information and Event Management Systems

A Security Information and Event Management (SIEM) system serves as the central intelligence hub within a Security Operations Center (SOC). Its primary role is to collect and standardize various data sources for the purpose of security analysis. While its core function is to identify and respond to security threats, the efficacy of a SIEM system often hinges on the expertise of cybersecurity professionals who configure it to recognize indicators of compromise and abnormal system behavior.

SIEM systems provide several valuable advantages for SOCs. They efficiently gather, organize, and analyze security data from various sources, simplifying monitoring and threat detection. For instance, leading SIEM solutions such as Splunk and IBM Security QRadar offer comprehensive log aggregation, correlation, and analysis capabilities, enabling SOC teams to detect and respond to security incidents effectively. Additionally, they support long-term data retention for post-incident analysis, a critical asset for modern SOC operations. SIEMs handle large log volumes effectively by consolidating them into a central system, expanding event and device collection capabilities [75, 76].

Despite their significance in cybersecurity, SIEM systems come with notable limitations and operational challenges. One significant challenge is the complexity of creating and managing rules, which can be overwhelming, especially for organizations with limited resources. Additionally, SIEM systems often lack contextual information, making it difficult to fully grasp the importance of detected events. Handling long-term data in an ad hoc manner is another weakness; SIEMs excel at short-term threat detection but struggle with advanced persistent threats (APTs) requiring extended analysis. Furthermore, large-scale SIEM deployments can be costly, with substantial upfront hardware expenses and ongoing maintenance costs. On the technical side, managing event collection and storage poses difficulties, given the vast amount of daily data and the need to balance storage expenses. Event correlation and analysis, vital for detecting sophisticated threats, can be intricate and resource-intensive. Lastly, visualizing SIEM data can be challenging, impeding the ability to extract actionable insights from the gathered information.

2.4 Sandbox Instrumentation

This section describes the techniques and tools used within a sandbox environment. Cuckoo Sandbox collects data and constructs detailed reports of what occurs within the sandbox. Suricata deals with network security monitoring within the sandbox. File permissions deals with access control within the sandbox. Together, these tools contribute to a secure sandbox.

2.4.1 Cuckoo Sandbox Report

Cuckoo comprehensively collects a diverse set of data to construct detailed reports encapsulating the behavior of analyzed samples within its sandbox environment. The network data segment of the report encompasses information such as URLs, UDP communication details, source and destination IP addresses, timestamps, protocol types, and requested URLs. Furthermore, it provides insights into DNS records, including domains, signatures, calls, APIs, severity, and descriptive information about the detected activity. Cuckoo's logs also capture details related to file operations, such as SHA1, SHA512, and MD5 hashes; file paths; and the names of processes involved. Behavioral information, including generic behaviors, process paths, process names, process IDs (PIDs), and summaries is meticulously documented. Additionally, the report includes the first seen timestamp; parent process IDs (PPIDs); and information on files opened, created, written, failed, and recreated. This wealth of data extends to cover system, file, and process attributes, providing a comprehensive overview of the observed behavior within the Cuckoo sandbox. The culmination of these detailed logs facilitates a thorough understanding of the malware's actions and aids in effective threat analysis [43, 77, 78].

2.4.2 Suricata

Suricata is an open-source Network Security Monitoring engine known for its emphasis on security, usability, and efficiency. It seamlessly integrates with diverse network solutions, logging activities such as HTTP requests, TLS certificates, and DNS queries. Its signature language enhances threat detection capabilities beyond traditional IDS. Suricata offers high performance, capable of inspecting multi-gigabit traffic with native hardware acceleration support [79]. However, we cease its use due to compatibility issues with generating events in the Cuckoo report.

2.4.3 File Permissions

Window's New Technology File System (NTFS) model implements access control by using Access Control Lists (ACLs). Within an ACL is a list of Access Control Entities (ACEs) that contain information about a user and their access levels regarding a particular file or folder object. In an enterprise environment, administrators are able to combat permission creep by assigning users to a custom user group, such as Finance or HR, and granting that group access to an object. This is particularly helpful when employees change work roles often where the admin can simply change the group that the user belongs to and all permissions that the user inherits are replaced. Administrators have the ability to create special permissions with any number of fine-grained attributes to best suit their needs.

An administrator must interact with permissions on a per object level, rather than per user. This does not allow for the admin to evaluate user permission across a whole directory structure. Interacting with a single ACL using Windows Explorer requires the traversal of four different interfaces. Interacting with multiple ACLs soon becomes a cumbersome task, which could ultimately result in permissions being overlooked. Not only is the administrator required to examine users or groups within the ACL, they also need to remember or explore group association to evaluate the inheritance of permissions from different groups. For an administrator to view the propagation rules, the admin is required to traverse through several interface windows to retrieve the required information [80].

2.5 Harbinger

At a high level, Harbinger is a tool that monitors and logs each user's interaction with the computer's user interface (UI) and correlates these acts with its respective network flows. This allows for a significantly higher level of accuracy and precision when creating network profiles, while also being able to distinguish malicious and non-malicious traffic. Harbinger is more accurate than traditional IP or DNS hostname profiling, with an error rate of 0.02%, down from 29-38% [81]. Harbinger runs on the client side and must be configured with the client's host OS. Harbinger's output is sent to the policy engine, which can run remotely on a centralized network controller, and is client independent, which allows it to work with any machine on the network [81].

With the increase in accuracy, precision, and the ability to work with an independent controller we ultimately decided to use Harbinger as the base for our project. However, we chose to modify the tool to have VM context labels, as well as a group label to make policy management easier.

2.6 Network Access Control

Network access control (NAC) implements policy enforcement to ensure only authorized and compliant users and devices are able to access certain resources. Files, applications, websites, and other resources are subject to NAC policies. Typically, on connecting to a network, there are a series of steps enforced to ensure the user and device are authorized to access the network and compliant with security requirements. These could include authentication procedures like logging into a service with a username and password, device security posture assessments to ensure devices are not vulnerable, and a profiling of the user's role and other attributes that indicate which resources the user-device pairing has access to. The success of network access control is reliant on the policies written by the network administration team, and these policies can quickly become unmanageable due to quantity and complexity [82].

Role-based access control (RBAC) grants access to resources based on a user's assigned roles or duties. Many RBAC policies implement the principle of least privilege, meaning users should only access resources that are absolutely needed to fulfill their responsibilities. Roles can be based on job titles, such as "Network Administrator," or on a collection of duties. Organizations define these roles and assign permissions to each one. Network administrators can do this manually, though several identity and access management (IAM) tools like Microsoft Azure Active Directory will assist in defining roles for their specific services [83].

After roles are defined, users onboarded to the system can be assigned to certain roles. Administrators can also perform this manually, though several tools exist to automate this process. Automation can occur based on the introduction of new user attributes, such as a new job title, or by analyzing a user's workflow and determining which role the flow most closely matches. When a user is assigned a role, the user inherits all access control policies associated with that role. RBAC is often a dynamic process, with user attributes and workflow being monitored closely to detect potential permission changes. Application-based access control (ABAC) is a more granular extension of typical RBAC implementations [84].

2.6.1 Domain Name System Protocol for NAC

The Domain Name System (DNS) protocol can be used for coarse-grained access control by gaining attributes when devices connect to the network, then implementing policies to control which resources a user can control based on their profile. These policies can change dynamically depending on the allow or deny lists of an organization, and DNS servers are able to implement these changes quickly to allow or deny access to certain devices. This means it can be used to quarantine devices by disallowing them to access resources when there is suspicion that a device has been compromised. While this approach is still coarse-grained, it is a necessary piece of NAC that helps organizations to apply broad policies across devices based on user or device profiles [85].

2.7 Firewalls

Firewalls are an essential part of network security and access control. Their main role is to examine a packets source and destination IP address, port number, and network protocol to determine if the packet should be allowed or denied. Firewall policy configuration is highly customizable and is designed to filter traffic in accordance to an organization's security requirements. Network administrators are responsible for defining rules within the firewall, specifying which traffic should be permitted and which should be blocked. These rules are typically organized in lists called ACLs. Administrators can create rules that target specific IP ranges, certain ports, or restrict access to specific services or applications. However, these policies must be regularly updated to adapt to new security threats and network needs, which can often be difficult when organizations have distributed firewalls across their network with different configurations.

2.7.1 Common Misconfigurations

Misconfigured firewalls can lead to security vulnerabilities and operational challenges within an organization's network. Often these issues can stem from the scope and complexity of firewall rules, as some organizations can have over 100,000 firewall rules [86]. When policies become overly intricate, it becomes increasingly challenging to ensure that all rules are correctly configured and coordinated, sometimes resulting in errors. These misconfigurations may inadvertently permit unauthorized access, exposing sensitive data, or creating security gaps that malicious actors can exploit. Firewall misconfigurations are a huge issue for network security and account for 99% of firewall breaches [87]. Inversely, overly restrictive rules can disrupt legitimate traffic and hinder the functionality of network services. The consequences of misconfigurations can include network downtime, data breaches, compliance violations, and costly security incidents. To prevent these issues, organizations must proactively manage their firewall configurations, which can be an unfeasible task for small teams of network administrators.

2.7.2 Virtual Firewalls

Virtual firewalls are software-based firewalls that maintain network security inside virtual environments. They inspect and analyze network traffic entering and exiting VMs. By using preconfigured policies, firewalls can identify potential security threats, vulnerabilities, or violations of data privacy. Administrators configure security policies that dictate how traffic is managed, ensuring that only authorized traffic is allowed to pass through. Virtual firewalls can also be application-aware, recognizing specific applications or services running in VMs for more precise control. A large advantage of virtual firewalls over physical firewalls is that they adapt dynamically to changing environments, scale with the addition or removal of VMs, and facilitate isolation and segmentation within the virtual network. Hardware-based firewalls are unable to detect traffic between two virtual machines running on the same host, making virtual firewalls more versatile [88]. Another benefit of virtual firewalls is that they can be centrally managed, making them easier to deploy and manage policies for.

3 Methodology

While the passive containment benefits isolation provides are highly desirable, current solutions are not feasible for modern enterprises due to increased costs in computer resource usage and difficulty in managing VM environments, network policies, and anomalous activity. Therefore, the overarching goal for our project was to maintain the effectiveness and integrity that isolation provides for a computer environment without sacrificing performance, user productivity, and inter-environment collaboration.

To implement strict isolation while maintaining usability and reducing performance costs on the end user, we aimed to create a web interface that mirrors a traditional environment, allowing for seamless VM access. Additionally, we aimed to provide rapid availability of VMs by leveraging remote computational resources. To maintain the benefits of isolation while maximizing inter-environment productivity, we designed a solution to monitor and label network flow based on risk groups and originated user action. Finally, to provide a way to observe and identify anomalous activity, we created a sandbox system that allows users to safely experiment with new tools and receive detailed reports on malicious activity.

3.1 System for Isolating Environments

In implementing strict isolation while maintaining usability and reducing performance costs on the end user, we aimed to answer the following questions:

RQ1. How does VM hot loading impact the startup times of applications in isolated systems?

RQ2. How much wasted screen space is incurred as a cost of using nested windows?

RQ3. What are the number of steps to access a VM with our approach when compared to other VMMs?

RQ4. What computing costs are incurred due to using our virtualized system on the client and server-side?

3.1.1 System Design Overview

To explore these ideas, we created a research environment to allow us to conduct our experiments. The system provided a minimal client for the user to interact with. Access to the hypervisor's privileged domain, domain0, was restricted through minimal backend API routes. These routes are accessed through the unprivileged user domain, the minimal client system user interface. The minimal client system reduced the trusted computing base by minimizing the total system footprint that is vulnerable to user-interaction. This user interface mirrored the design of a traditional desktop environment to provide the user with a familiar experience and reduce the usability costs normally associated with utilizing virtual machine based systems. When an application is launched, a VM is started on the backend server with that application. This provides isolation by virtualization while abstracting virtual machine interaction through the replication of a traditional desktop environment. Additionally, there is less load on the client system because computing resources are offloaded using remote server resources.

3.1.2 System Implementation

The two diagrams below give a high level overview of how the system functions. Figure 1 demonstrates the life cycle of an application in our system and Figure 2 gives a high level overview of all the system components, and how they interact with one another.

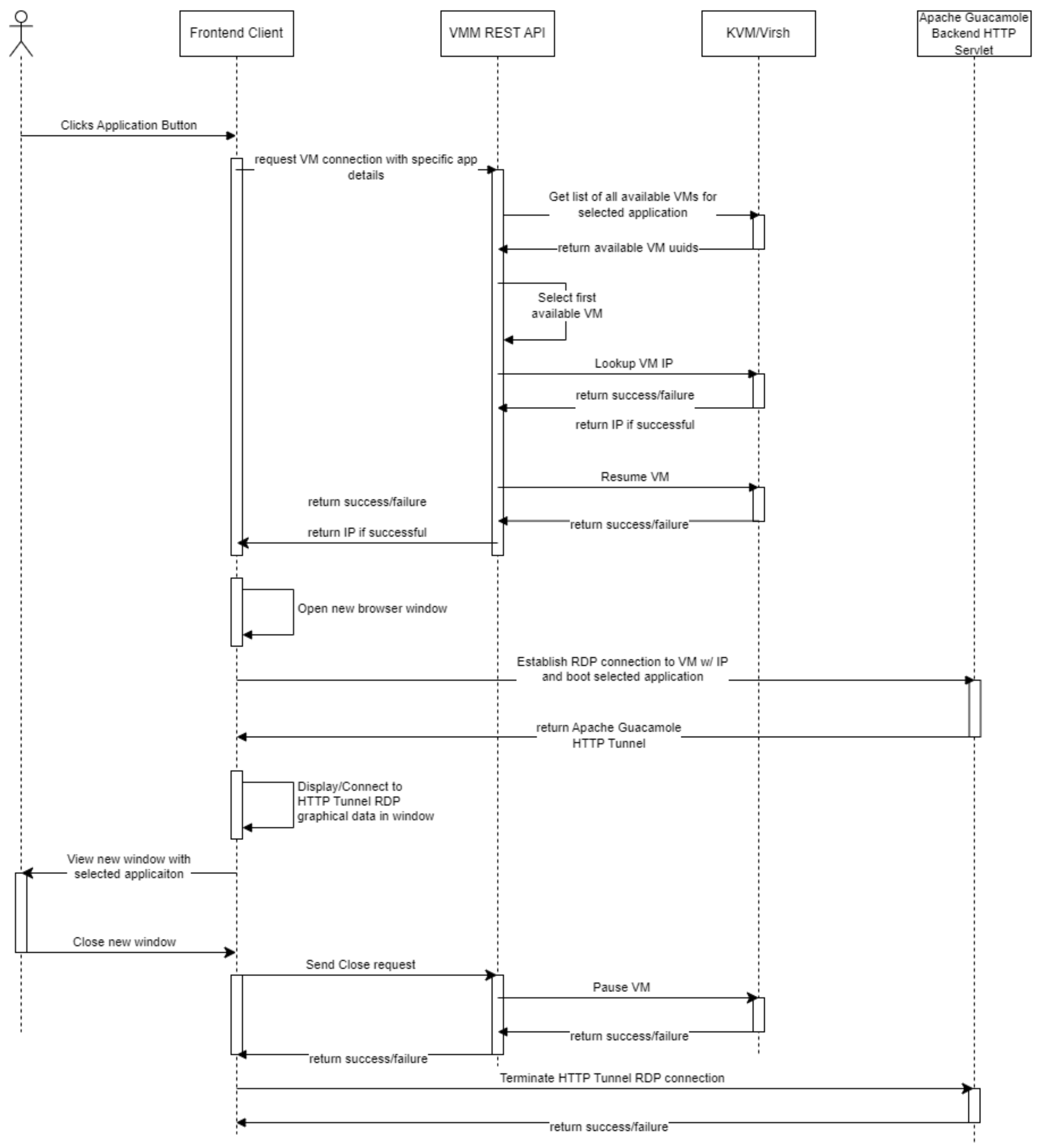
Frontend Client
VMM REST API
KVM/Virsh
Apache Guacamole Backend HTTP Servlet
Clicks Application Button
request VM connection with specific app details
Get list of all available VMs for selected application
return available VM uuids
Select first available VM
Lookup VM IP
return success/failure
return IP if successful
Resume VM
return success/failure
return IP if successful
return success/failure
Open new browser window
Establish RDP connection to VM w/ IP and boot selected application
return Apache Guacamole HTTP Tunnel
Display/Connect to HTTP Tunnel RDP graphical data in window
View new window with selected applicaiton
Close new window
Send Close request
Pause VM
return success/failure
return success/failure
Terminate HTTP Tunnel RDP connection
return success/failure

Figure 1: Flow diagram illustrating the life cycle of an application in our system.

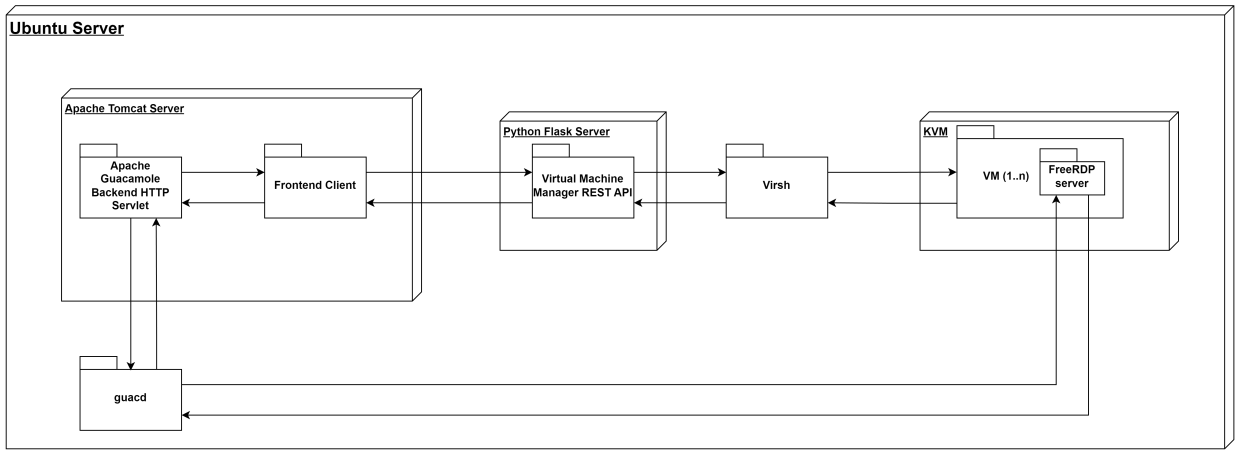

Figure 2: System architecture diagram. It demonstrates all the components in the system and how they interact.

User Interface. When implementing the user interface, we wanted to make the interface as similar to a traditional desktop environment as possible. Our solution to this was to create a webpage that look like a windows desktop. In making the webpage, we utilized HTML and CSS to create a webpage that was almost identical to a traditional desktop. We included buttons on the webpage that used the same images as desktop icons, as shown in Figure 3. This would let the user open their apps just like if they were on a regular desktop, but our system would provide additional security measures behind the scenes.

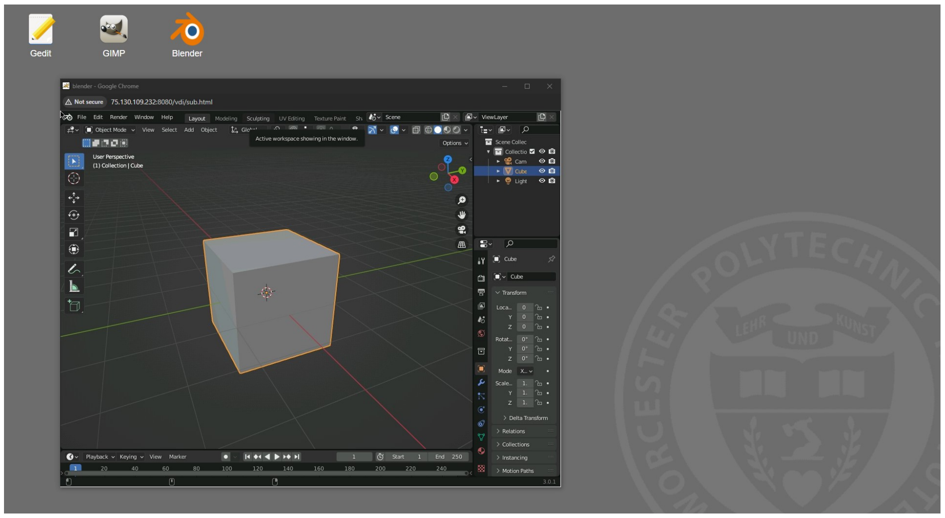

Figure 3: Screenshot of a user utilizing the web page client to access a remote Blender VM

To add functionality to the UI, we utilized Javascript to add logic to the HTML elements. This consisted of creating HTTP requests to the Virtual Machine Manager REST API (more details in Section 3.1.2) in order to start a VM when the application icon buttons are clicked. Additionally, we utilized the guacamole-common-js [89] library to establish an RDP HTTP tunnel connection with the Apache Guacamole HTTP Servlet [6]. The RDP tunnel was then embedded in an HTML element inside the new window that was opened for the application. This functionality can be seen in the system flow diagram in Figure 1.

Virtual Machine Management REST API. In order to control the VM lifecycle from the user interface, we needed to create a way to interface with our hypervisor. We chose KVM as our hypervisor because it was most available to us and fit our system needs. We created a REST API utilizing Python Flask [90] that acted as an interface between the front end client and the KVM hypervisor. A high level overview of the API flow can be seen in Figure 1.

This REST API contained the following endpoints:

- *GET* : `/get-connection/application-name`
- *POST* : `/close-connection`

The API contains a dictionary that maps application names to a list of VM uniform unique identifiers (UUIDs). This is manually set and used to determine which VMs can be used for specific applications. `Virsh`, the VMM CLI tool for KVM allows VMs to be maintained in 3 states (Running, Shutoff, and Paused). In our system, a VM is considered available if it is in the paused state.

The `get-connection` endpoint takes in an application name as an argument and returns the IP address and UUID of an available VM that supports that application. The first available VM that supports that application is chosen and a subprocess command is run using *virsh start uuid*, to start the VM before returning the connection details.

The `close-connection` endpoint takes in the UUID in the body of the POST request uuid: $uuid-value$. This endpoint is used to reset the VMs after use. A subprocess command is run using *virsh suspend uuid*, to pause the VM, successfully marking it as available in the context of our system.

We implemented "hot spare loading" by pausing the VMs after use, so that they would not need to be rebooted every time they are accessed, potentially worsening the user experience by delaying response time. There are, however, some drawbacks to this approach that we detail in Section 5.1.

Apache Guacamole HTTP Servlet & Client. To establish remote connections to our VMs, we decided to integrate the Apache Guacamole API [6] with our web application. The reason for choosing Apache Guacamole was that the API does not require any software to be downloaded from the client side because the Guacamole connections can be accessed through a normal browser. Not requiring client-side software reduced the amount of space taken up on each VM, and using a browser provided familiarity by allowing the user to work with a browser.

In the front-end client, we utilized the custom guacamole protocol available in guacamole-common-js over HTTP to connect to our Apache Guacamole HTTP servlet. The front-end client passed the application path, VM IP, VM RDP server port, username, and password to the servlet over the guacamole HTTP protocol. This servlet has the job of creating an HTTP tunnel through the guacamole daemon process, *guacd* and connecting to the VMs based on the connection details we set. Additionally, once the connection has been established we passed user input through the HTTP tunnel using the guacamole protocol. A high level overview of the flow mentioned here can be seen in Figure 1.

We utilized RDP to establish remote connection to our virtual machines. This was due to the initial application parameter, that allows a specified application to be booted when connecting via RDP protocol. This greatly simplified the application booting process; however, it did introduce some limitations which we mention in Section 5.1. Each VM utilized the FreeRDP RDP server [23] for establishing RDP connections.

3.1.3 Experiments

Our experiments looked into specific aspects of our overarching questions. The first research question focused on the various aspects of our system's usability compared to traditional and VM systems. The second question focused on our system's performance compared to traditional and VM systems. In our experiments, "traditional desktop" refers to a computer running Windows natively, and "VM system" refers to a computer running the OracleVM VirtualBox hypervisor on top of native Windows.

Usability Experiment 1: How does VM hot loading impact the startup times of applications in isolated systems?

Our first experiment was focused on comparing the startup times of applications across various systems to examine the impact of having VMs hot-loaded in our system. We compared the startup times of apps in a traditional desktop environment, a traditional VM environment (VirtualBox), and our system. We defined startup time as the elapsed time from when the user clicks on the first icon to start the application startup process until the application is presented to the user and in a ready to use state.

This experiment used three different scenarios. Each system was tested with three different applications: GEdit (notepad app), GIMP (photo editor), and Blender (3-dimensional creation suite). Each application was chosen to study the environments with varying levels of resource load.

We chose to measure from the first click to get a better representation of the holistic time costs for each systems. For the traditional desktop system and our system, this was clicking on the icon and waiting for the app to load. For the traditional virtual machine environment, this also included opening the VMM and starting the VM. We chose to do this because we wanted to get the app startup times in relation to their isolation. A user cannot access an app within a virtual machine without first going through the VM startup process, so it was included in this experiment. To reduce the error associated with the number of steps, the traditional virtual machine was split into three steps whose times were added together. Those steps were opening the VMM, starting the VM, and launching the app within the VM.

To measure startup times, we created a series of Python scripts. Before the script is run, the user has to take a screenshot of the app that is being tested in the desired state. For example, if the user wanted to test Microsoft Excel, they would need to take a screenshot of a fully loaded Excel application window. This image is used as a target for the script to look for. When the script is started, the first step is to start the app and simultaneously start the timer. For tests that run on the traditional desktop, launching the VMM and launching the apps within it was done using Python subprocesses. Subprocesses allowed us to pass in application paths and the command "open" and have the app started via the script. To test our system and launch the VM from the VMM, we utilized automated mouse inputs.

We utilized the `pynput` tool [91] to create a controller for the mouse, which we can use to give it commands. With the controller, we were able to feed it coordinates on where the applications and buttons are located on our screen and the number of clicks. As a result, whenever we needed the application opened or the button pressed, we told the controller to click at coordinate (x,y). However, in order to feed the coordinates to the controller, we needed to find the coordinates of each application and button. As a result, we used `pynput` [91] to create a script that listens to the mouse, and on the left click, it will return the x- and y-coordinate of the click. In our case, we found the coordinates of the applications in our system and the start button in the VMM. Then, we used those coordinates to open the applications and start the VM for our experiment.

Once the app launching process had started, the script started to look for the desired end state. This state would be achieved and the timer would be stopped when the app window that was launched was identical to the app window that loaded as the target screenshot. To check if the current app state matched the target state, the script continually took screenshots. First, the `pywinctl` tool [92] was used to get the border locations of the app window. The script then took a screenshot of the region specified by the app borders using the `pyautogui` tool [93]. The screenshot process takes a few seconds, so at the start of that process the time was recorded so it would be representative of the app state in the screenshot and not a few seconds after the screenshot was taken. Once a screenshot was obtained, the tool `imageio` [94] was used to load both images.

Those images were passed into a function that calculated the percent similarity between the images. Percent similarity was calculated by the ratio between the number of similar pixels to the number of total pixels. The images had to be of the same shape and number of pixels or the script threw an error. When counting the number of similar pixels, a tolerance had to be included. We found that even with images indistinguishable to the human eye, the percent similarity could be as low as 60 percent without tolerance. With the tolerance set, the script checked if the images were more than 90 percent similar. We set a constant tolerance level of 10 and an acceptance level of 90 percent because this is what our tests found to be a reasonable number that allowed similar images to pass while still rejecting images that did not match. If the images were above that threshold, then the elapsed time would be returned. If not, the script would start the percent similarity calculations of the next screenshot that was taken. This continues until an image above the set threshold was taken.

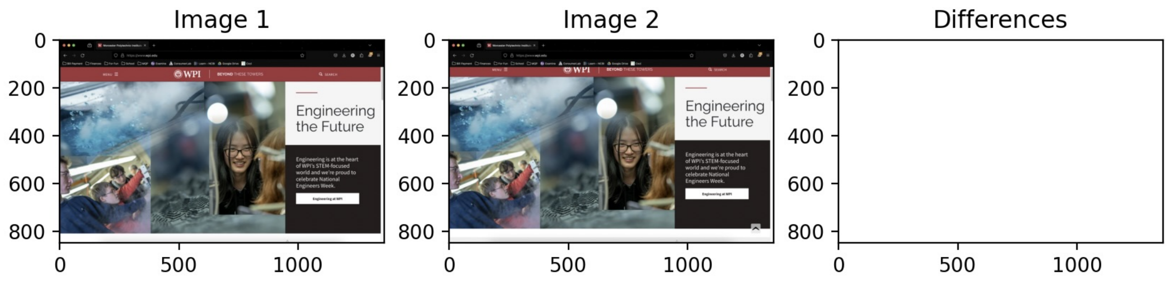

Figure 4: Image Difference Visualization Using www.wpi.edu Homepage

Tolerances were included to allow the images to be seen as “the same”. Figure 4 shows the results of testing the image similarity of the www.wpi.edu homepage within the same Firefox browser. All bookmarks and extensions were the same across runs. The target image seen on the left and labeled “Image 1” was taken using the screenshot function built and described above. It was then renamed to avoid being overwritten, and it ran a second time to ensure that the images were as identical as possible. “Image 2” in the middle was the result of the second run of the script. The box on the right labeled “Differences” is the plot of the absolute pixel-wise differences using the `numpy` function `numpy.abs(image1, image2)`. That function was used to show the areas within two images of the same shape that are different. Because there are no visible differences, the plot of the differences is left blank. However, when the percent similarity for these images was calculated using the `numpy` function `numpy.isclose()` on the pixels in the photos, the resulting value was 50.03%. To verify that this was not uniquely an issue with `numpy`, an additional function was made using the scikit-images library. This function required the images to be grayscaled, and a similarity index was formed using the scikit-images function `ssim()`. This returned a percent similarity of 61.68%, and the increase was likely due to the grayscale. Neither method of determining the image similarity was achieving the desired result of matching the human observed empty plot of differences.

While the plot of image differences showed as empty, that does not mean that there were truly no

Tolerance	Percentage Similarity of WPI Homepage
0	50.03%
1	53.36%
2	55.39%
3	57.19%
10	66.96%
20	77.74%
30	82.85%

Table 2: Tolerance Values and Percent Similarities for WPI Homepage Images

differences. We observed that the array of absolute differences used to form the differences plot was far from empty. To make up for these differences, tolerances were introduced. After testing with our images in the test environments, we found that when using a tolerance value of 10, we could regularly record percent similarity values above 90%. For reference, Table 2 shows the tolerance values and corresponding percent similarities using the same images from image difference visualization. Even in these matching images, the highest percent similarity achieved was 82.25% with a tolerance of 30. From this, we felt comfortable that a tolerance of 10 and percent similarity of 90% would accept our matching images in the test environments while failing images that did not actually match.

We hypothesized that the traditional desktop would have the lowest application startup times, followed by our system and then VirtualBox. The traditional desktop only needs to open the app. We thought that our system would be second because, in addition to loading the app, it also needed to establish the Apache guacamole tunnel across VMs. Lastly, we thought that the VirtualBox system would be the slowest because it had to incorporate the VM startup process in addition to the app startup process.

Usability Experiment 2: How much wasted screen space is incurred as a cost of using nested windows?

Our second experiment looked at wasted screen space resulting from artifacts introduced by an application loading in a virtual machine window. One of the drawbacks of using a VM is that some screen space is wasted due to nested borders; when an application is opened in a virtual environment, the virtual environment renders the application dimensions within the virtual window, which results on average a larger window size being produced. We defined wasted space as the pixels on the screen that were not related to the application being rendered within a window.

To investigate this, we decided to compare the percentage of wasted pixels running *Blender* across our system, a traditional desktop system, and a VirtualBox system. In this experiment, the traditional desktop system is assumed to have no wasted pixels when rendering an application; because of this, the traditional environment was used as a baseline. We used one scenario for this experiment due to the focus not being on the contents of the applications but rather the borders they were nested in, which would be similar for any application.

For this experiment, we first developed a Python script that would automatically resize our windows across all three systems to be the same dimensions. The script prints out a list of available window names on the user's system and asks the user to associate which window name correlates to which environment; this information is then used to obtain the handle information on the selected windows, allowing the script to automatically resize the windows. To perform this, we used `pygetwindows`, an application window module, to store information on our windows [95]; we then used this information with Win32GUI, a Python module for the Windows32 API, to resize the windows to the same dimensions [96]. We also used Microsoft's Accessibility Insight, a UI analysis tool, to measure the dimensions of various UI elements within the windows that would help calculate the wasted space [97].

The experiment was performed following the steps below:

1. Opened Blender locally on the computer, through a virtual machine, and through our system
2. Ran the script and selected the from a list of window names which window name correlates to which system
3. Opened Accessibility Insight
4. For each window, measured the dimensions of any UI element that does not contribute to rendering the application or that is not the window title bar
5. Divide the dimensions of the selected UI elements (SL, SW) by the total dimensions of the window (TL, TW) to calculate the wasted screen space percentage:

$$\left(\frac{\text{SL} \times \text{SW}}{\text{TL} \times \text{TW}}\right) \times 100$$

.

After a rough visual analysis of each of the windows, we hypothesized that VirtualBox would have a wasted screen space percentage of 10% when rendering an application, while the same application in our system would have a wasted screen space percentage of 5%.

Usability Experiment 3: What are the number of steps to access a VM when compared to other VMMs?

The last aspect of usability that we wanted to investigate was the process of accessing a VM. The traditional method of accessing a VM requires many intermediate steps: 1) launch the virtual machine manager, 2) select the VM, and 3) start the VM. These steps must be repeated for each VM the user wants to launch, resulting in many extra steps during a normal workflow. Additionally, this process may seem tedious and annoying for many users. We wanted to see how our system can minimize the extra VM access steps compared to the traditional VM system. Our measurement for this would be using the Keystroke-Level Model (KLM).

KLM is a usability tool to predict the time an expert user will take to complete a specific task with no errors [98]. KLM does not account for any learning required for the given task. KLM breaks down a given task into atomic actions: keystrokes (K), pointing the mouse to the target (P), mouse clicks (B), and switching between devices (H), such as keyboard and mouse. KLM has two additional operators or atomic actions, "M" and "R." "M" indicates the mental preparation required for a step, such as making a decision. "R" is the system response time. KLM links each atomic action to an interaction time and would produce a total task time. Table 3 illustrates the interaction time for each atomic action for our experiment. Since

KLM Atomic Action	Interaction Time (seconds)
Keystroke	0.28
Button Press	0.10
Pointing with a mouse	1.1
Switching hands from one device to another	0.4
Manually Drawing	N/A
Mental Preparation Time	0
System Response Time	0

Table 3: KLM atomic actions and their interaction time

each atomic action is linked to a specific interaction time, our KLM depends on assumptions:

1. The user is an expert in the domain and system.
2. The user is familiar with the hardware.
3. There are no interruptions during the task.
4. The user is a good typist (0.28 seconds per key press).

5. The user starts with their hands on the immediate device (the user does not need to switch devices at the start of the given task).

6. The user's system has instantaneous response time.

7. The user knows exactly what they have to do, requiring no mental preparation.

8. The user does not need to draw in our scenario, so "manually drawing" is not available.

Since KLM only needs the atomic actions for a given task, KLM does not need a functional system or user interface to calculate the predicted time.

In our experiment, our KLM model broke down a given task into the following atomic actions: keystrokes, pointing the mouse to the target, mouse clicks, switching between devices (keyboard and mouse), and system response time. Since this experiment did not relate to a traditional desktop environment, we only compared our system to a traditional VM system. We have a KLM script that tracks the user inputs in a given task to generate the list of atomic actions performed by the user. Our experiment follows all the assumptions listed above. Assumptions #1, #2, and #3 are standard assumptions for a KLM test, since this is an expert user performing a given task in an ideal environment. Assumptions #4 determines the interaction time for key press, whether we want 0.10 seconds per key press for a good typist or 1.20 seconds per key press for a novice typist. Assumption #5 allows us to ignore whether or not the user has to switch to a keyboard or a mouse at the beginning of their task. Assumption #6 allows us to ignore calculating the response time since it varies from machine to machine depending on the specs of the machine. Usability Experiment 1 explores the response time of our system compared to the traditional VM environment and the traditional desktop environment. We assume the user is an expert and has performed this repetitive task several times, so the user knows the steps to perform and does not need to make decisions while performing his task. This will also vary from user to user. As a result, we have assumption #7. The desired result is having our system have a lower KLM time than the traditional VM system.

Our scenario is accessing a VM. We will start the script on an empty desktop for the traditional VM system. Then, it will proceed with the following steps:

1. Start KLM script

2. Open Oracle VM Virtual Box

3. Select the VM

4. Start the VM

5. Stop KLM script

For our system, we will start with our system opened to simulate having our system on a desktop. Then, we will proceed with the following steps:

1. Start KLM script
2. Click an application to launch the VM
3. Stop KLM script

We hypothesized that our system would have a lower KLM. The KLM time will be lower because our system requires fewer steps to launch a VM.

Performance Experiment 1: What computing costs are incurred due to using our virtualized system on the client and server-side?

This research question aimed to compare our system's computing resource usage (CPU & RAM) to other computing environments. In this study we investigated how our system compared to a traditional desktop environment and a VMM environment (VirtualBox) in computing resource usage when starting up different applications.

All system environments were hosted on the same base system and hardware:

1. CPU: Intel(R) Core(TM) i9-9980HK CPU @ 2.40GHz
2. RAM: 33.2 GBs
3. Operating System: Ubuntu 22.04.3 LTS

There were 4 separate system environments we ran scenarios on:

1. Traditional Desktop Environment (Ubuntu 22.04.3 LTS)
2. VMM (Ubuntu 22.04.3 LTS virtual machine running in VirtualBox on Ubuntu 22.04.3 LTS)
3. Our System (Server) (hosted on Ubuntu 22.04.3 LTS)
4. Our System (Client) (accessed using Firefox web browser version 122.0.1)

Each system was tested with 4 different scenarios:

1. Idle: No running applications, idle system
2. Light Load: Run GEdit application

3. Medium Load: Run GEdit and GIMP applications

4. Heavy Load: Run GEdit, GIMP, and Blender applications

Each scenario was designed to test the environments with varying levels of resource load. In each scenario we ran the listed applications and recorded the average and maximum CPU and RAM usage in all four environments. We created a script to record the CPU and RAM usage across the different systems to analyze the resource consumption over time when opening the apps in the various systems. Additionally we kept track of the maximum resource utilization for both metrics, to serve as an upper bound for system resource requirements.

To achieve this we utilized the `psrecord` [99] Python package to record CPU and RAM load at an interval of 1 time every second. In each system environment and scenario we captured the CPU usage % and RAM usage in MB over 60 seconds. We then averaged the results and repeated the process for 30 trials for each system environment/scenario combination (16 combinations). The reason 30 trials was chosen was that after observing the data, it seemed all variance had been captured within 30 trials. We ran additional trials to confirm this finding, and did not see any additional variance in the results. We utilized the Traditional Desktop Environment in the Idle scenario as a baseline, and subtracted that from all other system environments and scenarios to get the difference between the base system idle, and running the scenarios. By subtracting the base system idle, we effectively show the "overhead" associated with each system environment/scenario combination.

We hypothesized that the traditional desktop would have the lowest total CPU and RAM usage, followed by the virtualized system (VirtualBox), and lastly our system would be the most resource intensive. The traditional desktop doesn't rely on any additional abstractions like our system or the virtualized system, which result in additional resource consumption. We expected the server-side portion of our system to be the most resource intensive, because it creates multiple virtual machines, one per application. However, we theorized that the client-side portion of our system would have the lowest CPU and RAM usage out of all environments. We theorized this because our system offloads all computing to the server, and only renders applications graphically client side.

3.2 Modified Harbinger for Improved Accuracy and Policy Scalability

We present a thorough overview of Harbinger, our justification for using Harbinger, our modifications to Harbinger, and real world scenarios to test network access control. Through this section we aim to answer the following question about network access control and policy scalability:

RQ1. To what extent does incorporating VM contextual labels and user intent data through UI monitoring benefit firewall accuracy and policy scalability?

3.2.1 Rationale for Harbinger Use

The team decided to use Harbinger as a base tool for a variety of reasons. Harbinger was proven to have significantly better detection of malicious packets than traditional IP/DNS packet filtering, especially in situations involving user intent. Harbinger detected 99.5% of malicious packets while a traditional approach, using packet headers alone, was only able to stop 70% of malicious packets [81]. Harbinger was created under the guidance of faculty at Worcester Polytechnic Institute. This allowed the team to easily consult with the creator to resolve issues with the tool and ask questions about tool functionality. Harbinger can capture user interactions at the kernel level, allowing the team to securely incorporate user intent events like mouse clicks and keystrokes into the information received by the controller without having to re-sign a kernel-level module with Windows.

3.2.2 Harbinger Installation and Setup

To install Harbinger, the team ensured all dependencies were met and then created a local copy of Harbinger rules and the driver installer in the root Windows directory. The team then copied pre-shared encryption keys into the user AppData directory to be used for communication encryption. Another directory in the ProgramData directory of Windows was created to hold the configuration file and all tool log files. The installer was then run to add the WinSight driver Harbinger uses to communicate with its various components, especially those for user intent data collection. Harbinger was then tested for functionality by analyzing log files to see if user intent information was captured.

During the setup and installation, the team found communication with the controller was nonfunctional, so a transition to local rule creation and policy management was initially implemented. Through reverting to a previous version of Harbinger with minimal functionality lose, the team was able to modify a working Harbinger tool. After confirmation of user intent data capturing and policy creation based on UI-based information, the team proceeded to the modification of the tool.

3.2.3 Modifying Harbinger

The team modified Harbinger to incorporate context data about machines into the information given to the controller to make decisions on whether to allow or deny network flows from certain machines with certain users. This new version is named Harbinger+. These contextual labels showed the associated functional group for VM. Functional groups were to be defined by system administrators, for example defining a machine as a finance machine or a research machine based on what type of work was to be conducted.

The team designed a text file input and parsing system to read in data and make decisions based on the new information. Harbinger had several components that needed to be adjusted to allow for the machine context labels to be stored: the SDN controller, the controller's MySQL database, and the UI receiver. Harbinger was designed to be placed on all machines in an enterprise system, with each instance of Harbinger connecting back to a main SDN controller. With this in mind, each machine contained Harbinger, its core dependencies, and, with the addition of the team, a text file called VMCreationData that contained all machine context data.

Each Harbinger+ instance was then designed to read the information from the text file and incorporate it into data flows that were sent to the controller. This required updating the UI receiver. The receiver, when given a process ID by the controller, originally sent back UI data associated with the process. The team modified this by adding the context label data to the set of information sent out by the receiver, so the controller could also understand what type of machine the process was associated with. The controller used a SQL database to store all flow information, so the team also modified this database by adding extra fields to allow for machine context labels. The team then modified the controller itself to accept this new information and make decisions on it based on policies created by system administrators. The team added logic statements to check certain characteristics of machine context labels.

3.2.4 Modified Harbinger Testing Suite

To understand the team's implementation of Harbinger+ relative to traditional Harbinger-integrated systems, the team developed a series of situations to simulate different events that could occur in an organization. The team created three scenarios based on real-world security threats that imagined a college with multiple departments using Microsoft products to manage operations. The team then analyzed the effectiveness of traditional network security, Harbinger, and Harbinger+ in each situation to estimate the impact of VM context labels and user intent data in firewall accuracy and policy scalability.

The team created testing infrastructure and policies to mimic each network security system, then developed scripts to automate the testing process for each testing scenario. The general infrastructure included a server, a controller, and three client-side machines. The server was built using XAMPP, with an Apache HTTP server and database backend. The controller in the testing infrastructure was the Harbinger+ controller, and the three client-side machines had Harbinger+ installed on each of them. Wireshark was also set up on both the client machines and the server to capture packet traffic and connections. As the testing environment is based in Windows 7, the team was limited to Wireshark 3.2 [100].

To simulate real world threat models, we created a set of scenarios to help evaluate and compare traditional networking, Harbinger, and Harbinger+.

3.2.5 Scenario 1: Sensitive Financial Information Protection

The financial department of the educational institution works with sensitive finance information about students, college earnings, and more. The college wants to restrict the network access to financial department data by those not working in the that department. For example, a server owned by the financial department should only be accessed by other finance-related machines and should reject communications from other machines.

This scenario looked at accidental or malicious data exfiltration of financial information and other sensitive information. More than half of all breaches are caused by insider attacks, either accidental or with malicious intent, but they are difficult to defend against and often require a layered security approach [101]. Guarding against exfiltration as a whole can assist in preventing data leaks from one organizational space to another, or to outside the network [102].

On top of the base infrastructure, the team added context labels to each machine. Two machines were labeled finance and one machine was labeled as research to compare the scalability of each system's policies. The policies were configured so that the server would only accept connections from finance machines with each system requiring different sets of policies to be created. To simulate connections with the server, the team created a script to make a cURL request to a website the server was hosting 10 times, with `sleep()` commands in-between to ensure one communication is completed before another begins.

3.2.6 Scenario 2: Automatic Scripts from Malicious Files

As the college uses the Microsoft Office suite, students use Microsoft Outlook to communicate through email and use Microsoft Word to open .docx document files. On any machine, malicious .docx files can hold scripts that are enabled by opening the file and clicking "Enable." The college wants to restrict network flows from malicious files characterized by processes acting without user input.

This scenario looked at detection and prevention of flows from scripts not initialized by the user. The team defended against malicious automated processes running on systems. Cryptocurrency miners, data exfiltration attacks, and botnet commands could all run automatically on an infected system, so protecting against scripts not automated by the user or the OS itself is important to the overall security of a system.

On top of the base infrastructure, the team created a Microsoft Word .docx file that contained an automatically-executed macro and added it to one of the machines. The document contained a social engineering prompt encouraging individuals to enable macros, starting the attack. The macro itself executed a `ncat` command to open a command prompt on the target machine. The document was added to one of the machines and the macro was executed automatically as the document was opened. To simulate connections, the macro was opened 10 times while UI-related traffic was run in the background.

3.2.7 Scenario 3: Role-Based Access Control

To protect student grades from being changed by students, the college implements VLANs to separate professor and student access so only professors can access the space in the network used to modify student scores. Students involved in Teaching Assistant programs need permissions to access and modify student scores, but cannot with the network isolation from the VLAN implementation.

This scenario looked at users having multiple permissions across roles and how access is granted to users with multiple permissions. With VLAN separation, special access rules can be created for individuals or matched across regular expressions, but updating permissions lists can be tedious. This also could introduce a human error component, so creating a human-adjacent but not human-directed solution will help to avoid this.

On top of the base infrastructure, the team added labels to the client-side machines. Two machines were labeled as students, and the other machine was labeled as a teacher. The first machine was designated as a teaching assistant, the second machine was designated as an ordinary student, and the last machine was designated as a teacher. Each role had different access permissions: students could create successful cURL requests, teaching assistants have all of the student permissions while also being able to make SQL database requests, and teachers had full access, allowing them to make ping requests. The team implemented policies on the controller to show these permissions for different roles, and the team automated the testing process using a script that tested all three (cURL, ping, and database access) request types for each role. We executed this script 10 times to simulate traffic.

3.3 Accessibility of Understanding UI Influence over Time Series Data

We present a comprehensive overview of the sandboxing methodology employed in our research, focusing on the utilization of Cuckoo Sandbox and associated tools. This sandboxing framework serves as a pivotal element in our investigation into the identification of suspicious activities. The primary objective is to assess the effectiveness of specific indicators, encompassing network, system, static, and behavioral aspects, in capturing events reflective of of attacks within sandboxed environments. We then evaluate whether our report can distinguish aligned and non-aligned UI activity with background micro-activity during a high-level event. The research questions are:

RQ1. With the sensors available in the sandbox, what amount of information (from time or data points) is needed to distinguish the different scenarios? Which sensors are the ones collecting that information?

RQ2. Can our report distinguish aligned and non-aligned UI activity and background micro-activity during a high-level (macro) event?

3.3.1 Research Design

Cuckoo Sandbox Integration. The foundation of our tool relies on Cuckoo Sandbox, an open-source, automated malware analysis engine. Cuckoo enables the opening of files or URLs (samples) in a pre-configured virtual environment. While users interact with the sample, Cuckoo logs network traffic, API calls, and file behavior. Notably, we chose Cuckoo Sandbox for its compatibility with VMCloak. VMCloak facilitates virtual machines instantiation, OS installation, application installation, and environment configuration, cloaking hardware components' names and versions.

Custom Analysis and Dependency Packages. To tailor Cuckoo Sandbox to our project's needs, we leveraged its modular design to create custom analysis and dependency packages. Our contributions include custom analysis and dependency packages specifically designed to support the automatic analysis of Microsoft Word .doc, .docx, and .docm samples within the virtual machine. This customization allows for a more focused and efficient analysis of potential threats.

Streamlined Cuckoo Installation Process. Installing and configuring a Cuckoo Sandbox environment typically involves a series of complex steps, including dependency installations, user inputs, and complex configuration instructions. Recognizing the need for efficiency, we developed shell wrappers that significantly streamline the entire Cuckoo installation process. These scripts handle downloading required libraries, setting permissions, creating the Python virtual environment for Cuckoo, generating virtual machines, editing Cuckoo's configuration files, and providing global routing for all sandboxes. To optimize the VM creation process, our Bash script wrapper allows users to specify desired features before initialization, reducing the amount of setup commands the user would need to run from 71 to 4.

Custom Virtual Environments. To address end-user needs effectively, our solution supports the creation of custom and configurable virtual environments. The number of virtual environments is only limited by server hardware. These virtual machines contain a minimal working set of applications necessary to support specific operations. For instance, a virtual machine designed for Word file analysis includes only LibreOffice and additional logging tools, minimizing the risk of exposure to potential threats.

Keylogger Integration. In parallel with Cuckoo Sandbox, a specialized Keylogger played a pivotal role in enhancing the data collection process alongside Cuckoo Sandbox. Implemented in Python3, the Keylogger monitored keyboard interactions and mouse clicks, providing a detailed log file with timestamps. This log file proved instrumental in visualizing end-user UI interactions during experimental runs.

While the Keylogger installation occurred within the Ubuntu environment hosting Cuckoo Sandbox, its integration was strategically placed outside the Cuckoo Sandbox environment due to limitations in capturing GUI log files via the Cuckoo Web API. This decision ensured a comprehensive approach to data collection, capturing both sandbox and GUI interactions.

Log Report Automation and Visualization Scripts. To create visualizations, we developed Python scripts tailored to our research objectives. The script was dedicated to creating graph visualizations using time series data, offering a dynamic representation of temporal aspects within the collected data. This visualization tool enhanced our ability to interpret patterns over time, contributing to an analysis of alignment.

3.3.2 Indicators of Compromise

Subsequently, our focus initially shifted to the identification of Indicators of Compromise (IOCs) based on the insights provided by Verma et al. [103]. Their research, utilizing Cuckoo Sandbox, laid the groundwork for understanding ransomware through the analysis of IOCs. In our own efforts, we desired to collect data to identify IOCs from the 45 highlighted in their study. Our initial focus was on determining how many of these IOCs our solution could successfully extract from the logs generated during the post-sandbox activity. This approach was to enhance the effectiveness in the traceability and post-event analysis of potential ransomware incidents [103].

However, our attention pivoted towards understanding the mechanisms within the sandbox responsible for gathering the identified IOCs, alongside UI data such as keystrokes and mouse clicks logged in the generated reports. This shift aimed to ascertain the specific components within the sandbox environment that facilitate the collection of IOCs and user interaction data, crucial for distinguishing between different experimental scenarios. This shift was prompted by the necessity to understand how IOC and UI data are collected in the sandbox, and analyzing their significance in the generated reports became crucial for improving our ransomware analysis methods.

We focused on extracting and analyzing the following IOCs identified by Verma et al.:

Network Indicators of Compromise. Network IOCs play a pivotal role in proactive defenses, providing actionable insights into suspicious activities within a network environment. Within this context, data derived from reports generated by Cuckoo Sandbox, specifically extracted from the network section of the report.json file, provide support in identifying the following network IOCs.

- **Perform HTTP requests.** Monitoring HTTP requests serves as a substantial indicator of compromise for two crypto ransomware families, CryptoWall and Locky. The analysis of the HTTP message sequences and their respective content sizes is sufficient to detect threats [104]. Verma et al. had identified this as a valid network IOC used by the ransomware detection tools HitmanPro and Cryptomonitor.

- **Connect to tor2web.** Tor is a software project created to facilitate anonymous Internet browsing, while tor2web is a related initiative aimed at allowing users to access onion services without relying on the Tor Browser [105]. Verma et al. recognized a connection to Tor2web to be one of their elevent critical IOCs for the ransomware families CryptoWall, CryptoLocker, and Locky. Cryptomonitor also uses this IOC to aid in its detection.

- **Excessive DNS requests.** If a high volume of DNS requests are observed within a brief time frame, particularly directed at unconventional domain names, it indicates potential malicious activity. This increases suspicion when such queries take place during non-standard hours [106]. Cryptomonitor uses this network IOC to aid in its detection.

- **Request to high entropy domain names.** Domains with randomness and complexity in their names, such as '12.345.shsjk.com', particularly those connecting to C and C servers potentially managed by an attacker, can be deemed suspicious [106]. Verma et al. recognized a request to high entropy domain names as one of their critical IOCs for the ransomware families CryptoWall and Locky. HitmanPro and Cryptomonitor incorporated this IOC into their detection mechanisms.

System Indicators of Compromise. System IOCs highlight potential vulnerabilities created on the operating system level. By watching for changes in Windows registry files, system indicators of compromise can be identified. With the exception of the *Renames file to executable* indicator, system IOCs are observed by comparing a baseline Windows registry snapshot with one taken at the end of sample execution.

- **Disable UAC.** User Account Control (UAC) is important for Windows security as it restricts unnecessary access, prompts user approval for administrative actions, and prevents silent installations, mitigating the impact of malware attempting unauthorized system changes [107]. Disabling this feature is a system IOC that Cryptomonitor takes account for.

- **Disable Task Manager.** Disabling the Task Manager is an IOC that is monitored by Cryptomonitor. This is because it hampers the monitoring of crucial system metrics, impeding the identification of suspicious or abnormal activities related to hardware resource usage and performance [108].
- **Stops Windows Security Center service and prevents it from starting up on boot.** Windows Security Center is essential for device protection, offering threat monitoring, account security, firewall management, and customizable controls for apps and browsers [109]. This service being stopped is an IOC that Cryptomonitor, Bit Defender, and Cryptoprevent analyzes because this action can hamper the system's ability to continuously monitor and respond to security threats.
- **Stops Windows Defender service and prevents it from starting up on boot.** Microsoft Defender Antivirus for Windows is an antivirus that safeguards users' personal data and devices [110]. A disruption in this service is posed as an IOC because it may undermine the continuous protection mechanisms designed to defend against various threats. Cryptomonitor, Bit Defender, and Cryptoprevent all utilize this system IOC into their detection mechanisms.
- **Stop Windows Update service and prevents it from starting up on boot.** In Windows, updates ensure device security and performance. Disabling windows update services is a risk as it prevents crucial security enhancements, leaving the device vulnerable to threats [111]. This IOC is closely examined by Cryptomonitor, Bit Defender, and Cryptoprevent.
- **Stops error reporting service and prevents it from starting up on boot.** Windows Error Reporting is a feedback system that gathers data on detected issues within Windows, relays this information to Microsoft, and offers user potential solutions [112]. Stopping this service is identified as an IOC monitored by Cryptomonitor, Bit Defender, and Cryptoprevent because it hinders the system's ability to collect crucial information on errors when suspicious activity occurs.
- **Firewall disabled.** Windows Firewall is a security feature that safeguards the device by filtering incoming and outgoing network traffic [113]. Disabling the firewall becomes a problem as it eliminates the protective barrier, leaving the device vulnerable to unauthorized access and threats. This IOC is utilized by Cryptomonitor in its detection of suspicious behavior.
- **Stops background intelligent transfer service and prevents it from starting up from boot.** Background Intelligent Transfer Service (BITS) optimizes file transfers, considering costs and network usage [114]. Disabling BITS poses as an IOC that is monitored by Cryptomonitor, Bit Defender, and Cryptoprevent as it disrupts file transfer efficiency, potentially causing delays in critical tasks and impacting overall system functionality.

- **Renames file to executable.** File names with executable extensions (.exe, .bat, .cmd, etc.) can indicate compromise, as attackers often misguide malware by renaming files to appear benign [115]. Deceptive practices allow adversaries to trick users into executing malicious code. HitmanPro, Cryptomonitor, and Cryptoprevent analyze this IOC to aid in their ransomware detection. This indicator is identified in Cuckoo Sandbox's dropped file report that reveals any files that have been renamed.

Static Indicators of Compromise. Static IOCs are crucial in fortifying defenses by identifying potential threats through a variety of static information. The following indicators are identified through reports generated by Cuckoo Sandbox, specifically from the signature and behavior sections of the report.json file.

- **Recently downloaded.** Monitoring files that were recently downloaded and added to the system may be indicative of ransomware that has been recently delivered to the system. HitmanPro, Cryptomonitor, Cryptoprevent, and Crypto Drop all utilize this static IOC into their detection mechanisms.

- **Reported infected by YARA.** YARA is a tool designed to detect and exploit code similarities among malware samples within a family [116]. Leveraging YARA's reports can be particularly effective against the evolving threats like ransomware. This recognition is underscored by Verma et al., who identified YARA as a valuable IOC for pinpointing ransomware threats.

- **Process signature.** A process signature is a precise pattern that aids in identifying and recognizing malicious threats, facilitating the detection and mitigation of potential risks [117]. HitmanPro and Crypto Drop observe signatures to aid in its detection of ransomware threats.

Behavioral Indicators of Compromise. Behavioral IOCs are integral to revealing insights into the file activity of a system. Leveraging reports Cuckoo Sandbox generates, the following behavioral IOCs are identified using the information extracted from the behavior section of the report.json file.

- **Dropped files.** Dropped files are files written to the disk during sandbox execution [118]. Some ransomware arrives on a system as a file dropped by other malware or as a file downloaded unknowingly by a user when visiting malicious sites. Dropped files is a valid behavioral IOC that is closely examined by all five detection tools.

- **Periodic Activity.** After encrypting files, ransomware, like CryptoLocker, tends to periodically rescan the system for new drives and files to encrypt [119]. This periodic activity is considered an IOC and is observed by CryptoPrevent to detect suspicious behavior.

- **Untrusted processes spawning/injecting into target processes.** Process Injection serves as an IOC because it allows attackers to run potentially suspicious processes within the guise of seemingly harmless ones [120]. HitmanPro, Cryptomonitor, Cryptoprevent, and Crypto Drop actively scrutinize the events related to process injection to detect any signs of suspicious behavior. Ransomware follows a common tactic of placing copies of its executables into the Temp directory. In this process, the malware renames these copies with random file names and subsequently executes them from that location [121]. All five detection tools investigate activity in this directory to watch for suspicious behavior.

- **AppDataRoaming Directory.** Certain ransomware strains, such as CryptoLocker, employ a technique where they generate executable files within the AppDataRoaming directory. These files are assigned random names, contributing to the stealthy nature of the malware [121]. In this directory, all five detection tools scrutinize actions to identify and monitor potential suspicious behavior.

- **AppDataLocal Directory.** The *Temp* directory, a subdirectory of AppDataLocal, serves as a crucial location for the execution of ransomware executables. This is where the malware's malicious code is run, furthering its impact on the system [121]. All five detection tools analyze activities within this directory to detect any signs of suspicious behavior.

- **ProgramData Directory.** The *ProgramData* folder serves as another potential location where users or adversaries may choose to store data. This directory can be utilized for various purposes, both legitimate and potentially malicious [122]. The activities within this directory are closely examined by HitmanPro, Cryptomonitor, Cryptoprevent, and Crypto Drop to actively watch for any indications of suspicious behavior.

- **Encrypts files/directory.** The encryption of files and directories is identified as a critical IOC according to Verma et al. Ransomware's purpose is to employee encryption to hold a victim's information at ransom [123], so this crucial IOC is analyzed by HitmanPro, Cryptomonitor, BitDefender, and CryptoDrop to determine malicious behavior.

3.3.3 Experiment 1: Synthetic Malware

In this experiment, we present two synthetic malware scenarios that reflect a *Crypto Locker* malware attack and a *Locky* malware attack. The Synthetic *Crypto Locker* attack is characterized by four critical IOCs which include deleting a shadow copy, connecting to Tor2web, file encryption, and packed or obfuscated code. The Synthetic *Locky* attack is characterized by four critical IOCs which include deleting a shadow copy, connecting to Tor2web, request to high entropy domain name, and file encryption. This experiment is crucial in ensuring our solution can pick up any file system, network, and UI data.

Run 1.1: Synthetic Crypto Locker. In this scenario, a synthesized malware script was presented that covered three of the four critical IOCs. The Tor2web connection was implemented via the requests python library. The script used AES key algorithm to encrypt every file in a given directory with the cryptography library. Due to VM limitations, however, all files over 100MB were not encrypted.

Run 1.2: Synthetic Locky. In this scenario, we employed synthesized Microsoft Word macro-enabled document (*docm*) files, simulating the behavior of the *Locky* ransomware attack. Two out of four IOCs were successfully incorporated into the analysis. These incorporated IOCs include connecting to Tor2web services and requesting high entropy domain names. Both the Tor2web and high entropy domain connections utilized the LibreOffice SystemShellExecute library service. The macro code (for both IOCs) spawned a shell and executed an involuntary web redirection. Note that the scale of testing was constrained by the limitations of virtual machines and LibreOffice.

3.3.4 Experiment 2: Legitimate Malware

For this experiment, we executed 18 runs involving diverse and verified malware sourced online, as detailed below.

- cryptowall
- Encryptor_ps1
- l0ck
- Locky
- Ransomeware
- wannacrypt
- Alphabet
- Cerber
- eda2
- EternalRocks
- GhostCrypter
- Golden Eyes
- jigsaw
- mamba
- Matsnu
- petya2
- ShellLocker
- xorist

3.3.5 Experiment 3: Benign Operations

In this experiment two benign scenarios were completed for expected alignment of UI and other data. The first benign scenario focused on mostly general network and some filesystem usage. The second benign scenario utilized a harmless macro script within a benign *.docm* file, which focused on UI and filesystem usage.

Run 3.1: General Machine Usage. In this scenario, the end-user interacted with basic filesystem functionalities like unzipping a .zip file and general web browsing.

Run 3.2: General Docm Usage. This scenario contained a .docm with a macro designed to improve document formatting and enhance various functionalities without venturing beyond the document to make malicious changes to the filesystem. The benign macro was uploaded to the sandbox. Because of Cuckoo Sandbox's limitation with capturing events unrelated to the uploaded file, using the uploaded benign file to overwrite an existing file and even creating a new version of that file was tested. This was to see if the UI data does in fact align with the file system data.

3.3.6 Aligning IOCs with UI Activity

When running the experiments, Cuckoo sandbox generated a *report.json* with network, process, and file system information data we used to create our time series graphs. In addition to the report.json, we created a keylogger that contained UI data and put the information in a file called *ubuntuLogfile.txt.* We created a script that combined the two files and compiled the data. This script was made in Python and simply merges the information together into one file named *output.json.*

To properly visualize the data from our experiments, we created time series data graphs that would encapsulate how network, process, file system, and UI data would interact. Using the *output.json* file, we developed a Python script that would filter through the data and plot the information using matplotlib based on when the events occurred. If there was no data for a specific field then the graph for that field would not be shown. For example, if there were no Tor2web connections in an experiment then the Tor2web graph would not appear in the results. The keystrokes and mouse clicks fields were included to note the relation between UI information and all other fields. The timeline of UI activity and the timeline of identified IOCs are stacked vertically. Utilizing consistent time buckets ensures a synchronized view, facilitating a nuanced analysis of the system's behavior. This approach facilitated a detailed analysis of system behavior, highlighting any discrepancies or misalignments in UI activity and IOCs.

By correlating each IOC with its respective UI activity, we scrutinized the source origins of suspicious activities and potential threats. These visual representations allowed us to identify patterns and anomalies, enhancing our understanding of system behavior. These comprehensive graphs provided insights into temporal discrepancies between UI activities and corresponding IOCs. Visual comparisons aided in spotting patterns and irregularities, while analyzing the vertical alignment of UI activity and IOCs within the same timeframes offered valuable insights into the solution's efficacy in capturing and linking IOCs.

4 Results and Findings

For our isolation-centric system, we looked at the effect VM hot loading had on application startup time, the amount of wasted screen space that was rendered with an application, the number of steps it took to boot up a VM, and the computing costs from usage of our system on both the client and the server. With Harbinger, we tested how the introduction of user intent data and machine context labels influenced network decision-making policies in three scenarios. Finally, we experimented with our sandbox system and obtained the IOC data that was produced when it was attacked with synthetic versions of CryptoLocker and Locky ransomware.

4.1 System for Isolating Environments

The majority of our experiments pertaining to our isolation-centic system focused on its usability. Our goal was to have our system provide a similar user experience to that of a traditional system. In the following sections, we provided analyses on application startup time, wasted screen space, and the VM creation process and compared it to our competitor, VirtualBox. We also provided data regarding the computing costs our system sustained when operating on the client and server level.

4.1.1 Usability Experiment 1: What effect do VM hot loading techniques have on startup time?

Overview and Hypothesis. This study is designed to examine the comparison between startup times for applications across various systems. We investigate how a traditional desktop system, our created system, and VirtualBox compare when starting up different apps. Each system was tested with 3 different apps: gedit for a light load, GIMP for a medium load, and Blender for a heavy load.

Our hypothesis is that the traditional desktop would have the lowest startup times, followed by our system, and the VM system would be the slowest. We thought that the traditional desktop would be fastest because all it needs to do is to load the app. We expect our system to be the second fastest because in addition to loading the app, it also needs to send the RDP commands and create the Guacamole tunnel across VMs. Lastly, we expected the VM trials to be the slowest because the VM will need to be booted in addition to the app, which we think should take significantly more time.

Results and Analysis. The data results from our tests are present in Table 4, and the corresponding bar chart is shown as Figure 5. In the Table 4, we show the time taken in seconds to load the app in each environment, as well as the percent of startup time of the traditional desktop system. We include the

App	**Traditional Desktop, s (%)**	**Our System, s (%)**	**VirtualBox, s (%)**
gedit	1.04 (100%)	2.70 (259.62%)	21.27 (2045.19%)
GIMP	2.58 (100%)	3.72 (144.19%)	22.81 (856.59%)
Blender	1.84 (100%)	2.51 (136.41%)	22.07 (1199.46%)

Table 4: App Startup Time Across User Systems Results

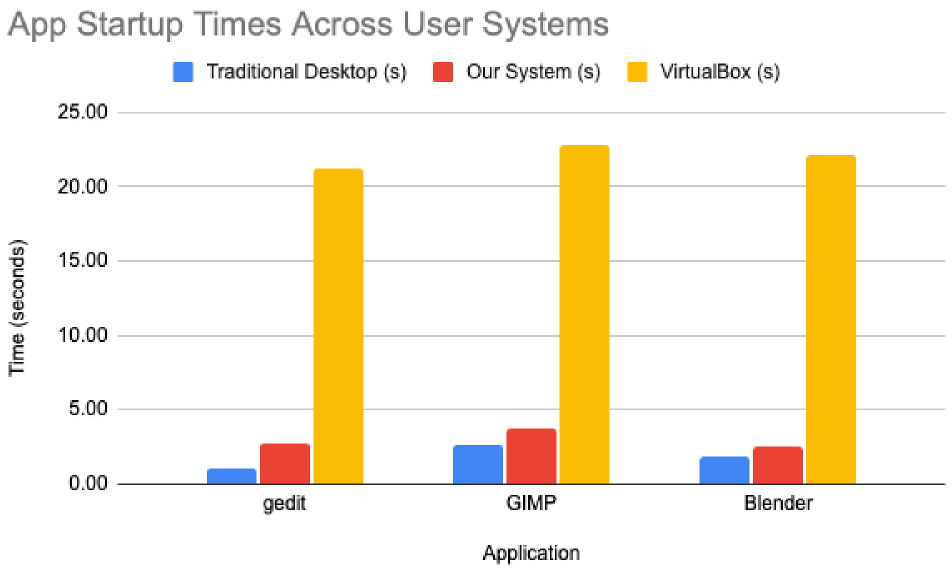

Figure 5: Comparison of Application Loading Times Across Systems

percentages to show the traditional desktop system as a point of reference that most people are familiar with.

The data supports our hypothesis about how the systems' speed would compare to each other. First, we correctly predicted that the VirtualBox tests would be the slowest. Because it has to boot the VM every time, it is unable to perform as fast as the other two systems. Across the three apps, the startup times were 21.27 seconds, 22.81 seconds, and 22.07 seconds. These values correlate to 2045.19%, 856.59%, and 1199.46% of the corresponding tests in the traditional desktop environment.

Our hypothesis is also correct when we predicted that the traditional desktop system would be faster than our system in all scenarios. Our system is slower than the traditional system for all apps. Our system took 2.70 seconds (259.62%), 3.72 seconds (144.19%), and 2.51 seconds (136.41%) for gedit, GIMP, and Blender respectively. These times are close to that of the traditional desktop experience, as all of them were within 2 seconds of the desktop times.

The VirtualBox tests took the longest because those trials included opening the VMM, starting a VM, and then launching the app in it. While the typical VM workflow will not include opening a VM for every application, that is what was required to achieve the same level of isolation as our system, and thus the most similar end result of a single app in a single VM. Because of this, the VM trials always took significantly longer than either of the other systems. Our system is able to remain competitive with the traditional desktop system. This is attributable to offsetting some of the time needed for RDP commands and Guacamole tunnels with having the apps on the VMs are pre-loaded. Because the apps were already launched and only needed to be resumed, the limiting factor for our system was handling the RDP and Guacamole components. The largest difference between our system and the traditional desktop is 1.66 seconds, showing that we were able to maintain most of the speed of the traditional desktop system. The times for our system and the system using VMs to achieve isolation show a staggering difference. The VM trials took 787.78%, 613.17%, and 879.28% of the app startup times in our system. These differences show the effectiveness of our system, as we achieve the same levels of isolation for a fraction of the time that using VMs takes. We consider the increase in time of less than 2 seconds across all apps to be acceptable as it is a minimal difference in a standard user workflow and is worthwhile considering the increased resistance to malware attacks.

4.1.2 Usability Experiment 2: How much wasted screen space is incurred as a cost of using nested windows?

Overview and Hypothesis. This experiment focused on the amount of unnecessary screen space from window artifacts created from Blender running through VirtualBox and our system in comparison to a traditional system environment. We determined this by implementing a Python script that resized our desired windows to a standard dimension size as well as Microsoft's Accessibility Insight to measure the dimension sizes of UI elements in the windows. For this experiment, each window was set to a screen ratio of 4:3 with a resolution of 1515 by 1095 pixels. For the case of VirtualBox, the VM was resized to 1600 by 1200 pixels within the Linux environment as this fills up the window and minimizes any unnecessary wasted space; likewise, our system VM was resized to 1024 by 768 pixels due to similar reasons.

We conducted a visual analysis across both systems, and predicted that Blender running VirtualBox would have a wasted screen space percentage of 10% while the same application in our system would have a wasted screen space percentage of 5% when compared to a traditional system environment. In this hypothesis, we assumed that the traditional environment contains no wasted screen space in their applications.

Results and Analysis. Our results show that, on average, VirtualBox had a wasted screen space percentage of 7.80%, while the same application in our system had a wasted screen space percentage of 4.85% when compared to a traditional system environment. This resulted in a deviation from our original hypothesis

Screen Ratio	**Traditional (px)**	**VirtualBox Wasted Space (px)**	**VirtualBox Ratio (% of wasted space)**	**Our System Wasted Space (px)**	**Our System Ratio (% of wasted space)**
4:3	1515 x 1095; or 1,592,265	1515 x 82; or 124,230	7.80%	1515 x 51; or 77,265	4.85%

Table 5: Wasted Screen Space comparing Traditional Environment, VirtualBox, and Our System

of 10% and 5% by 22% and 3% respectively; the margin of error that occurred in our hypothesis was most likely due to human error from the original visual analysis.

4.1.3 Usability Experiment 3: What are the number of steps to create a new VM when compared to other VMMs?

Overview and Hypothesis. This experiment compared accessing a VM in the traditional VM system and our system. We determined this by using a KLM script to produce a KLM time and track the execution time of our scenario: accessing a VM.

We hypothesized that our system would have a lower KLM. The KLM time will be lower because our system requires fewer steps to launch a VM. Our system requires fewer steps by abstracting the process of creating a VM behind a single button, which also launches the desired application. This removed the intermediate steps of accessing a VM.

Results and Analysis. The KLM script only tracks the user's mouse clicks and keyboard inputs. Using mouse clicks and keyboard inputs, the KLM script generates a KLM string to determine how long it would take a user to perform these steps. Table 6 provides the key to read the KLM strings in Table 7. There are faults in the KLM script, specifically its inability to register double clicks properly. Since the KLM script only tracks mouse clicks without mouse movement, it adds a P for mouse movement before each B mouse click. As a result, we have to manually remove a "P" from the string for each double mouse click. Additionally, we have to perform an extra mouse click to exit the VM to stop the script, which results in having to remove the extra mouse click. Table 7 presents the results of our KLM experiment.

The results supported our hypothesis that our system will have a lower KLM time than the traditional VM system, where our KLM time is 32.4% of the traditional VM system, respectively. The KLM time was faster for our system since we extrapolated the steps the user has to perform to access a VM into a single button. We can see the steps if we break down the simplified KLM string for VirtualBox, "PBBPBPB". To open Oracle VM Virtual Box Manager, it took "PBB", which was to move the mouse to the desktop

KLM Operator	Definition
K	keystroke
B	button press
P	pointing with a mouse (moving the mouse)
H	switching hands from one device to another
D	manually drawing
M	mental preparation time
R	system response time

Table 6: KLM Key

	VirtualBox	Our System
KLM String:	[‘B(mouse click)’, ‘B(mouse click)’, ‘B(mouse click)’, ‘B(mouse click)’]	[‘B(mouse click)’]
Simplified KLM String:	PBBPBPB	PB
KLM Time	3.7	1.2

Table 7: KLM Results

application and double-click to launch the application. Then, we had to select the VM, which was “PB.” Lastly, we had to click “start”, which was “PB”. For our system, simply clicking any application that we want will allow us to access a VM with the application already opened. As a result, our system’s KLM string was “PB”, a single click.

4.1.4 Performance Experiment 1: What computing costs are incurred due to using our virtualized system on the client and server-side?

Overview and Hypothesis. This study was designed to compare our system’s computing resource usage (CPU & RAM) to other computing environments. In this study we investigated how our system (both the client and server components) compared to a traditional desktop environment and a VMM environment (VirtualBox) in computing resource usage when starting different applications. Each system was tested with 4 different scenarios: Running GEdit for a light load, GEdit and GIMP for a medium load, and GEdit, GIMP and Blender for a heavy load. In each scenario we ran the listed applications, recording the average and maximum CPU and RAM usage in all four environments.

Our hypothesis for this experiment was that the traditional desktop would have the lowest total CPU and RAM usage, followed by the client side portion of our system, then the virtualized system (VirtualBox), and lastly the server side portion of our system would be the most resource intensive. The traditional desktop does not rely on any additional abstractions like our system or the virtualized system, which result in additional resource consumption. We expected the server side portion of our system to be the most resource intensive; however, we theorized that the client-side portion of our system would have the lowest CPU and RAM usage out of all environments. We theorized this because our system offloads all computing to the server, and only renders applications graphically client side.

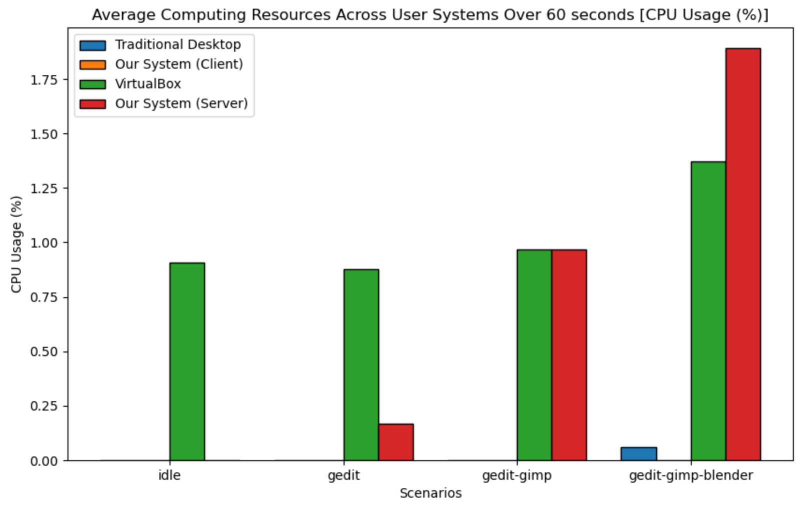

Figure 6: Average % CPU Load across all scenarios

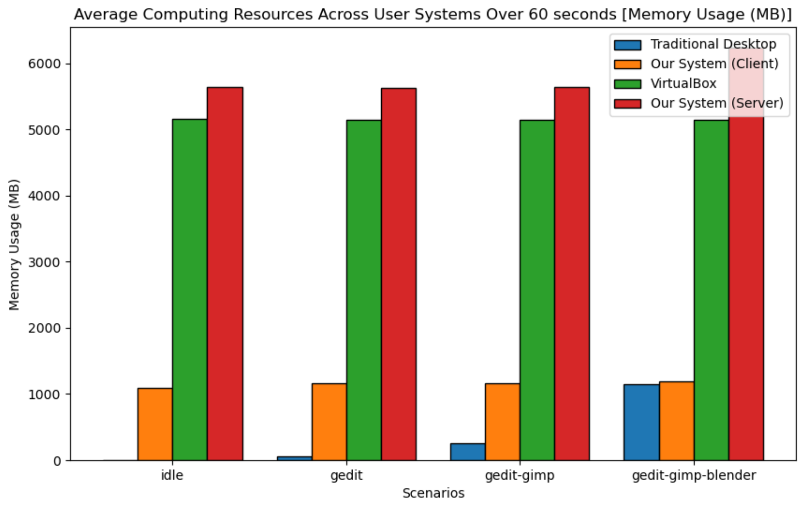

Figure 7: Average RAM (MB) across all scenarios

Scenario	Environment	Avg CPU Usage (%)	Max CPU Usage (%)	Avg Memory Usage (MB)	Max Memory Usage (MB)
Idle	Traditional Desktop	0	0	0	0
	System (Client)	0.0	0.0	1089.09	1093.12
	Virtual Box	0.91	2.9	5156.34	5350.68
	System (Server)	0.0	0	5613.18	5639.36
GEdit	Traditional Desktop	0	0	44.47	59.67
	System (Client)	0.0	0.0	1158.6	1224.96
	Virtual Box	0.88	1.33	5146.02	5171.81
	System (Server)	0.17	0.5	5590.94	5619.55
GEdit & GIMP	Traditional Desktop	0	0	207.62	255.3
	System (Client)	0.0	0.0	1156.35	1236.08
	Virtual Box	0.97	1.2	5148.86	5164.51
	System (Server)	0.97	1.7	5617.89	5644.65
GEdit & GIMP & Blender	Traditional Desktop	0.06	0.1	1146.25	1187.99
	System (Client)	0.0	0.02	1186.3	1191.67
	Virtual Box	1.37	1.94	5149.97	5161.04
	System (Server)	1.89	3.8	6234.18	6920.73

Table 8: Computing Resources used across all scenarios and environments

Results and Analysis. All graphs and tables were normalized by subtracting the values from the traditional desktop environment in the idle scenario. This was done because the traditional desktop was used as a baseline, to show the overhead associated with running systems/applications in our scenarios. This is why the traditional desktop values are 0 for the idle scenario.

As displayed from the experiments in Figures 6, 7 and Table 8, the results supported our hypothesis. The traditional desktop environment had the lowest resource usage, followed by the client-side portion of our system, then the virtualized system, and lastly, the server-side portion of our system.

The percent CPU load values (Figure 6) are low with less than two percent for all environments and scenarios. This is likely due to the nature of our scenario design. Our scenarios did not perform active tasks in the applications, the applications were just opened. This likely caused the processes/threads to sleep reducing the overall CPU consumption. Despite this, we can still see a trend with the VirtualBox environment and the server component of our system. As the number of open applications increases, so does the CPU load. The base CPU load is higher for VirtualBox, which is expected because the entire virtualized system is loaded at once. The server component of our system "hot loads" virtual machines for each application by maintaining the virtual machine in a "paused" state where processes are suspended, relinquishing CPU resources. This is why initially the CPU load for VirtualBox is higher, however as more applications are opened (more virtual machines are resumed), the CPU usage for the server component of our system out scales VirtualBox. We can also observe that the CPU usage for the client side portion of our system is low, around the same as the traditional desktop environment even as the number of open applications increases. This is likely due to the fact that the client portion of our system is accessed through the web browser. When additional applications are opened, a new web browser window is opened. The load of these windows is minimal, as they just render graphical content and send user input through an RDP connection.

Looking at the RAM load values (Figure 7), we can see a similar outcome. The disparity in the overhead between the client and traditional vs VirtualBox and server environments can be clearly seen, with a gap ranging between 4000-5000 MB. This is expected, as both the VirtualBox and server environments make extensive use of virtualization which adds system overhead. Throughout all scenarios, the average RAM usage of VirtualBox stays fairly constant at around 5100 MB, with the server at around 5600 MB. The server has high RAM utilization, despite our hotloading technique, because when a VM is "paused," RAM resources are not freed up like CPU resources. This means that the server has the equivalent RAM utilization of all 3 applications open at the same time in all scenarios. We see a jump in server RAM utilization from 5600 MB to 6200 MB in the final scenario, this is likely due to additional RAM consumption when the Blender application is fully loaded. The client results are also promising, we see that in all scenarios the client RAM usage stays around 1100 MB, even as the number of applications open increases. In contrast while the traditional desktop environment starts with low RAM utilization, as we open more intensive application like GIMP and Blender, the RAM utilization increases rapidly. In the third scenario with all 3 applications, opening Blender increased the average RAM utilization from 200 MB in the second scenario to 1150 MB. We believe that as more applications are opened, especially intensive ones that this trend will continue. These results show that the client environment is more scale-able than the traditional environment and supports the idea that by offloading running applications to a remote server, we can reduce the load on the end user.

4.2 Modified Harbinger for Improved Accuracy and Policy Scalability

The team sought to determine how incorporating VM context labels and user intent data into policy decision-making impacted firewall accuracy and policy scalability. With modifications to Harbinger allowing for the intake of VM context label information, firewall decision-making was minimal. However policy scalability was increased based on a variety of scenarios mirroring real-world security threats. The traditional network security structure, the Harbinger base system, and the modified version of Harbinger were compared using confusion matrices to better understand the accuracy of each model.

Accuracy rates were recorded for each system. True positive (TP) indicated that a flow that was malicious was correctly identified and blocked, true negative (TN) indicated that a flow was benign and was allowed, false positive (FP) indicated that a flow was benign but identified as malicious and blocked, and a false negative (FN) indicated that a flow was malicious but identified as benign and allowed. While false positives can interfere with operations, false negatives are the most dangerous inaccuracy in each scenario, as allowing malicious packets to slip through undetected could result in data exfiltration, a denial-of-service attack, or another serious issue.

4.2.1 Scenario 1: Sensitive Financial Information Protection

This scenario looked at how the additional information given to the system affected decision-making with VMs sending requests to servers with access permissions related to different VM context label groups. Table 9 presents the number of flows that were detected by each configured system, as well as the number of policies each system used.

	TP	TN	FP	FN	Policy #
Traditional	10	20	0	0	2
Harbinger	10	20	0	0	2
Harbinger+	10	20	0	0	1

Table 9: Number of flows detected and number of policies for each system in Scenario 1

While all three scenarios had the same accuracy, as shown above in Table 9, the number of policies required differs for each implementation. This is because the Traditional and Harbinger networks require hosts to be identified on a individual basis, while the Harbinger+ is able to group multiple hosts together that share policies. This highlights a potential issue in the future as the number of policies will scale according to the number of hosts, especially with the introduction of human error at each policy creation step. Additionally, hosts in a traditional network are only able to be identified by their IP address, meaning any changes in network addressing can invalidate security policies.

4.2.2 Scenario 2: Automatic Scripts from Malicious Files

This scenario examined how effective each system was in stopping malicious traffic from non-user actions. Table 10 presents the accuracy rates and number of policies for each system.

The traditional network system shows less accuracy than the Harbinger and Harbinger+ systems. This is especially concerning, as the false negative rate for the traditional system is high, meaning it was unable to correctly identify malicious traffic from the .docx macro. The false positive rate for Harbinger and Harbinger+ is higher than ideal, due to blocking system-started processes. With proper training of both

	TP	TN	FP	FN	Policy #
Traditional	0	7	0	10	1
Harbinger	10	4	3	0	2
Harbinger+	10	4	3	0	2

Table 10: Number of flows detected and number of policies for each system in Scenario 2

systems to understand normal background traffic not generated by users, the false positive rate is predicted to be significantly reduced. However, as individual policies are required for any background process, the number of policies for Harbinger and Harbinger+ will scale up while traditional remains stagnant. Once created, however, these policies can be applied to all hosts across the network, so the number of hosts does not affect the number of policies. Student policies will have to be duplicated for students with teaching assistant access, meaning that there will be double the number of policies for Harbinger when compared to Harbinger+.

4.2.3 Scenario 3: Role-Based Access Control

This scenario looked at matching user and VM security levels to allow users access to different levels of data based on their roles. This scenario involved two context groups, teachers and students, but identified specific students as teachers assistants with additional permissions. Table 11 presents the accuracy rates and number of policies for each system.

	TP	TN	FP	FN	Policy #
Traditional	30	60	0	0	4
Harbinger	30	60	0	0	4
Harbinger+	30	60	0	0	3

Table 11: Number of flows detected and number of policies for each system in Scenario 3

All three systems had the same true positive and true negative rates, even with the additional information of VM context labels and user intent data. However, this scenario faces similar challenges to Scenario 1 when it comes to scaling policies to a larger network. Both the traditional networking system and unmodified Harbinger are only able to identify singular hosts, meaning that the number of policies is proportional to the number of hosts. However, Harbinger+ doesn't allow multiple context labels or hierarchical grouping, meaning each teacher assistant must be identified individually.

4.2.4 Discussion

Through the execution of each scenario in a testing environment, it is clear that Harbinger+ had increased policy scalability compared to the traditional networking system and the base version of Harbinger. Harbinger as an access management system acts as an allow list, meaning that often system administrators need to create extensive policies for traffic management. Harbinger+ reduces the number of policies required as endpoints can be grouped together to avoid redundant policies that are hard to keep synchronized.

Across scenarios, Harbinger+ was most successful against threat models like the ones displayed in Scenario 1 and Scenario 3. In these scenarios, certain groups can access resources instead of creating individual access-granting policies for each machine or user. As predicted, Harbinger+ requires less policies on average, allowing administrators to create policies more efficiently. This success is mainly in policy scalability, as all three systems had similar accuracy in these scenarios.

Additionally, Harbinger+ outperforms traditional networking when it comes to host identification, using context labels and user identifiers instead of static IP addresses. Harbinger is able to identify hosts using a unique host ID, however policies can only be applied to one host ID, making policy scaling difficult. These differences could especially be seen in Scenario 3, which simulates an environment with users with different permission levels interacting with the same server for different tasks. In an educational or enterprise environment, as the number of users increases, the number and specificity of policies for Harbinger and traditional networking increases, while Harbinger+ grows only slightly.

4.3 Understanding UI Influence over Time Series Data

In this section, we've displayed and provided in-depth analyses of each of our synthetic malware classes as well as some real malware examples. We discuss IOC detection capabilities of our solution, behavioral analysis and IOC correlation with cuckoo sandbox and keylogger data, and temporal analysis of UI and system activities in different events.

4.3.1 IOC Detection Capabilities of Our Solution

We initially aimed to investigate the extent to which our solution's data collection mechanisms, focusing on specific network, system, static, and behavioral indicators, effectively capture events indicative of attacks in sandboxed environments. Among the critical IOCs identified by experts, encompassing eleven specific indicators, our solution successfully collects data on 4 of 11. Table 12 presents the critical IOCs, outlining the key features essential for understanding and detecting potential threats of Cryptowall, Cryptolocker, or Locky Ransomware. The table provides a comprehensive view of the 4 critical features that our solution actively collects and analyzes. These features, labeled as f1 to f11, encompass a range of malicious activities, including the deletion of shadow copies, utilization of the I2P anonymity network, connections to Tor2web, requests to high-entropy domain names, file encryption, encrypted file names, screen locking, deletion of original files from the disk, importation and linking to crypto libraries, employment of packing/obfuscation techniques, and the creation of Read-Write-Execute (RWX) memory.

Classes	Features										
	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11
Cryptowall	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
Cryptolocker	✓	✗	✓	✗	✓	✗	✗	✗	✗	✓	✗
Locky	✓	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗
Collected by our Solution	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓	✗

Table 12: Key features of malware families

4.3.2 Behavioral Analysis and IOC Correlation with Cuckoo Sandbox and Keylogger Data

The data captured by the Cuckoo Sandbox, enhanced with our integrated keylogger, provides a comprehensive outlook on the behavior of various software scenarios, enabling us to distinguish between malicious and benign activities. The analysis of the *report.json* files generated by the sandbox revealed that specific IOCs could be identified and correlated with malicious actions. For instance, a high frequency of UDP connections, a surge in the number of active processes, or an unusual pattern of file operations often signify a ransomware attack. The integrated keylogger data proved to be particularly insightful. In scenarios involving ransomware, there was a noticeable absence of consistent user-initiated keystrokes and mouse clicks, or an irregular pattern that deviated significantly from expected human behavior. This was in stark contrast to the benign scenarios, where regular and periodic patterns of user interactions were recorded, reflecting typical user engagement with the system. Some samples revealed their malicious intent almost immediately after execution through a rapid succession of IOCs, while others required a more extended observation period. Nevertheless, the combination of network behavior, file activity, and the absence or irregularity of user interaction data typically yielded enough information within the first few minutes of execution to classify the scenario accurately.

4.3.3 Temporal Analysis of UI and System Activities in Different Events

In our exploration of high-level events, we delved beyond mere quantification of IOCs to unravel the intricate dynamics between aligned and non-aligned UI activities and the underlying microactivity within the system. Graphical representations of IOCs and UI activities offered a depiction of their temporal occurrences during macro-level events.

Experiment 1: Synthetic Malware

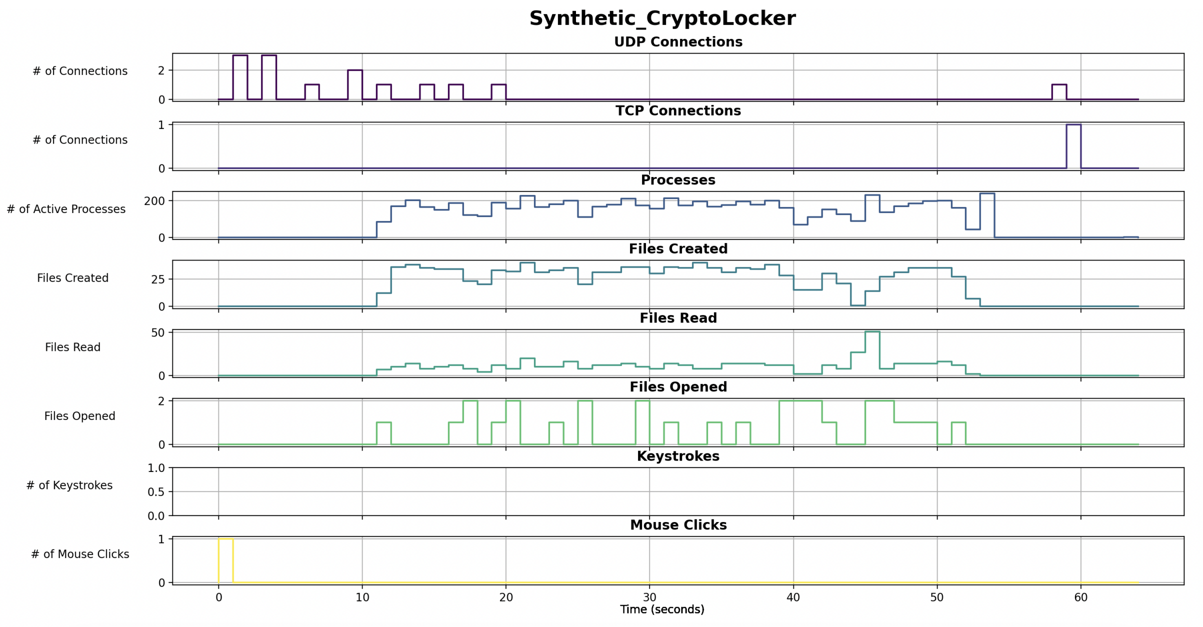

Figure 8: Run 1.1: Synthetic Crypto Locker

Run 1.1: Expected Non-Alignment during Crypto Locker Activity The graph showed heightened file modifications within the specified directory indicative of the encryption process. The graph also showed heightened network connections, indicative of the Tor2web process. Among both of these data, there was no concurrent UI activity, suggesting a non-alignment.

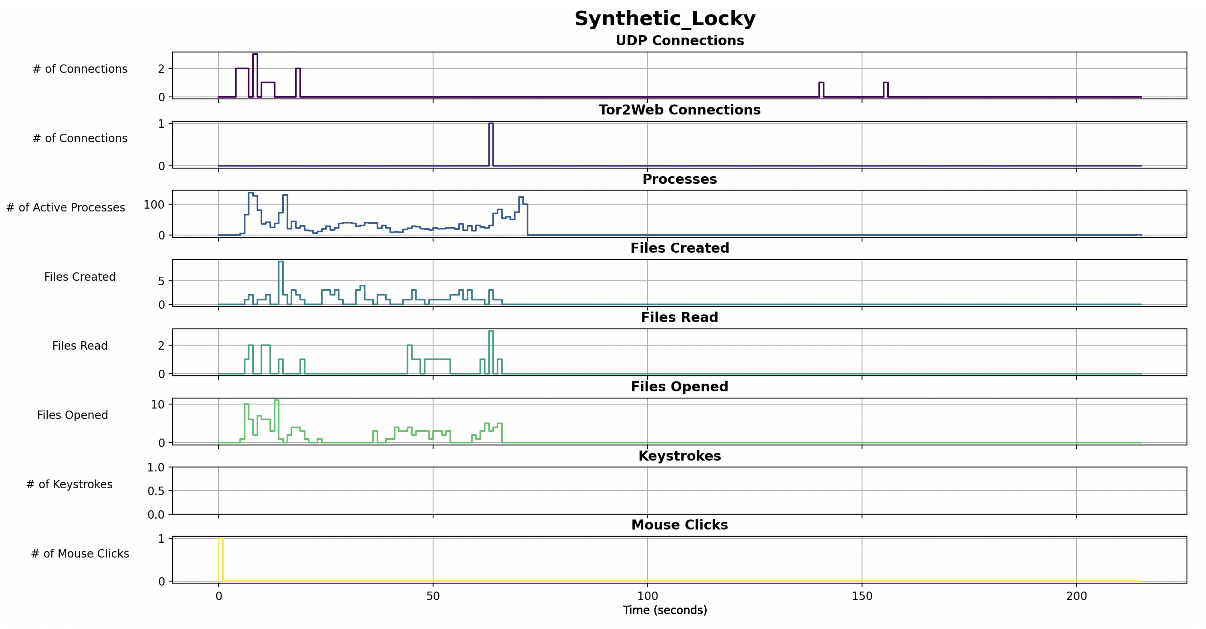

Figure 9: Run 1.2: Synthetic Locky

Run 1.2: Expected Non-Alignment during Synthetic Locky Activity The graph illustrated increased Tor2web and UDP connections upon opening of the macros file. These are indicative of the Tor2web and high entropy connections. There is little concurrent UI activity (opening of macros file), suggesting a non-alignment.

Experiment 2: Legitimate Malware

Upon examining the data visualizations for six different malware samples—Encryptorps1, Locky, CryptoLocky, Jigsaw, Xorist, and Cerber—out of a total of 18 different malware types used in our experimentation, we can draw several conclusions. These six have been selected as they are representative of the broader dataset and encompass the diversity of behaviors observed across all tested malware instances.

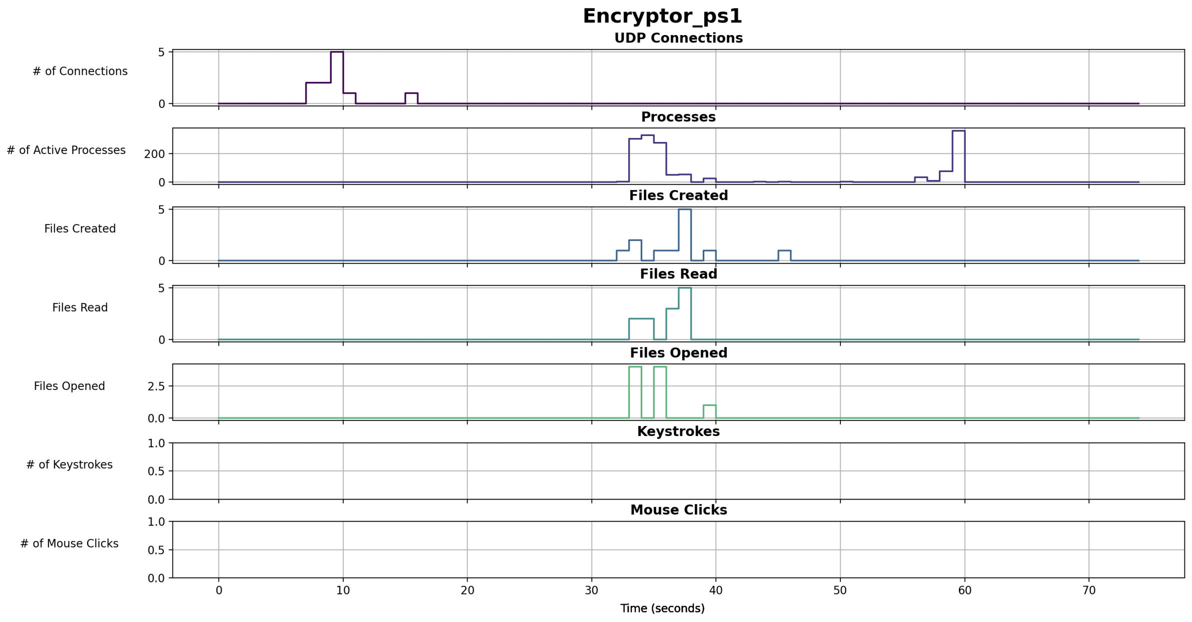

Figure 10: Run 2.1: Encryptorps1

Run 2.1: Encryptorps1 Encryptor's graph shows a similar pattern in UDP connections and active processes. However, the files created and read have a delayed start, suggesting a staged attack progression. The file operations peak and then decrease, possibly reflecting the encryption process completion. Once again, minimal UI activity occurs, reinforcing the idea of non-aligned activity during the malware operation.

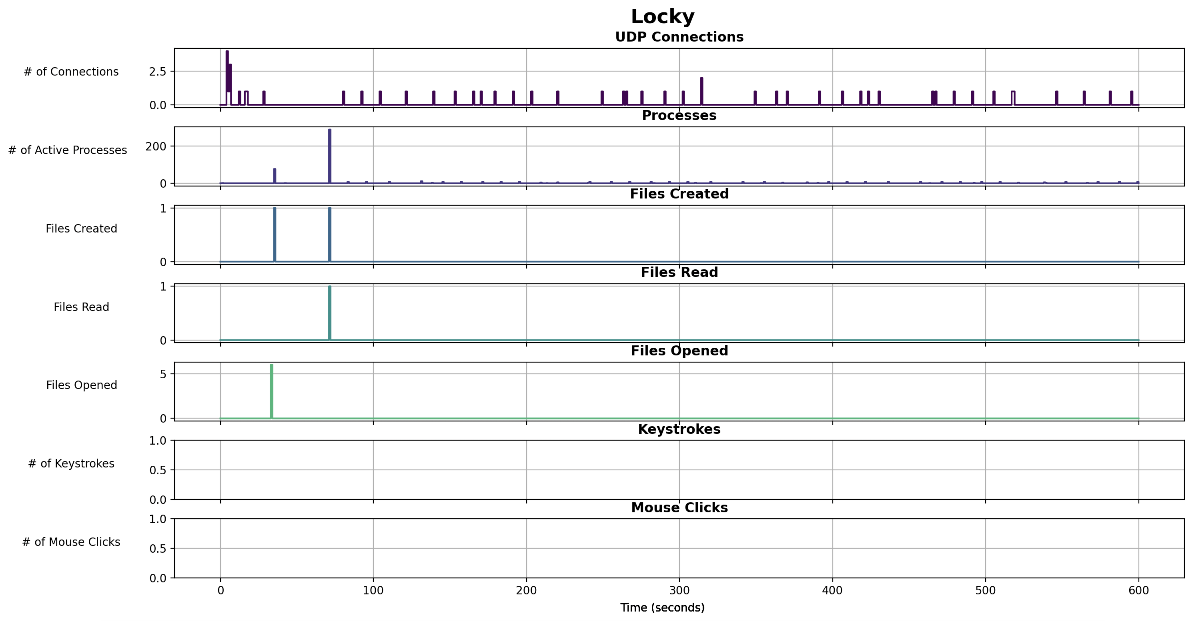

Figure 11: Run 2.2: Locky

Run 2.2: Locky Locky's graph displays a consistent level of UDP connections with periodic bursts, which may correspond to data exfiltration or synchronization with external servers. The processes show less variation, while file operations are sporadic. The UI activity is not observed and does not correlate with the IOC data, again supporting the hypothesis.

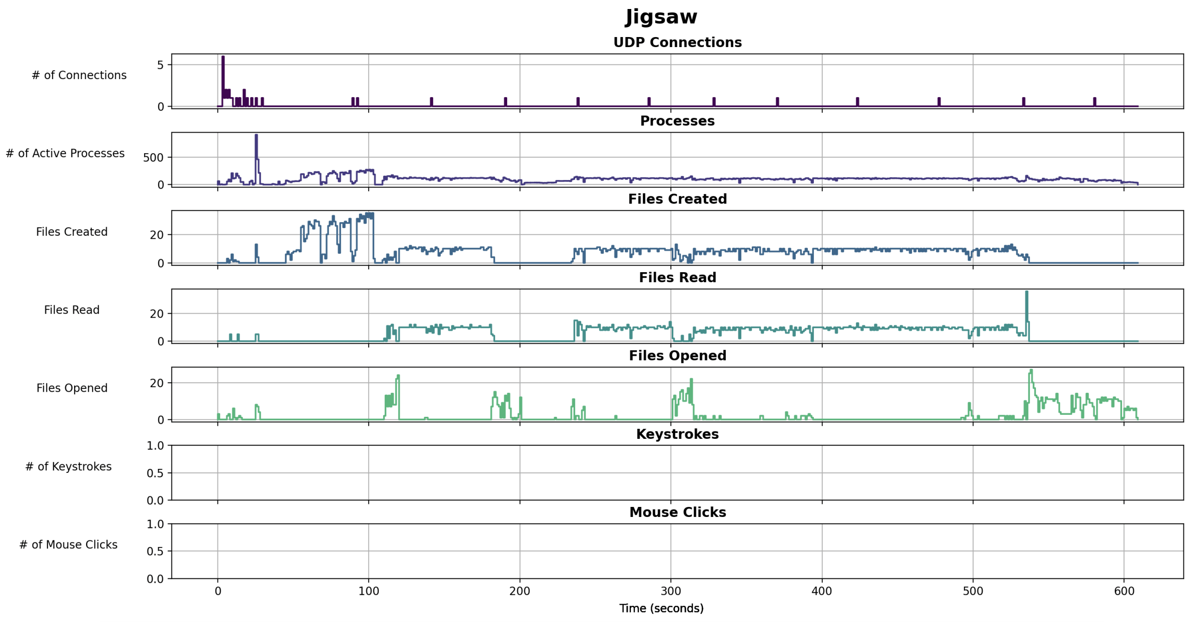

Figure 12: Run 2.3: Jigsaw

Run 2.3: Jigsaw The pattern observed in the Jigsaw graph is markedly different, with more pronounced fluctuations in active processes and file operations. The sustained high level of file reads and openings might indicate a more aggressive encryption or data manipulation strategy. The keystrokes and mouse clicks are non-existent, aligning with our expectations of non-aligned activity.

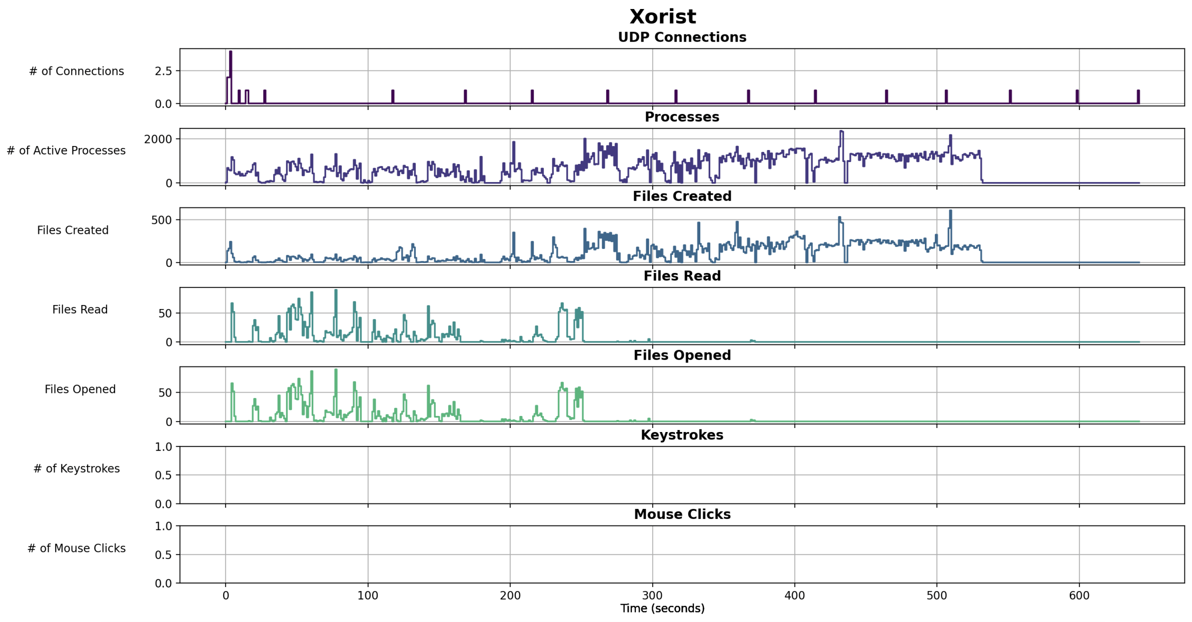

Figure 13: Run 2.4: Xorist

Run 2.4: Xorist Xorist's graph showcases a high level of activity in file operations, particularly in files being read and opened, which exhibit a jagged pattern. This could indicate the malware's active engagement in scanning and interacting with a large number of files, potentially for data gathering or searching for specific file types to encrypt. The lack of UI data supports the hypothesis of non-alignment.

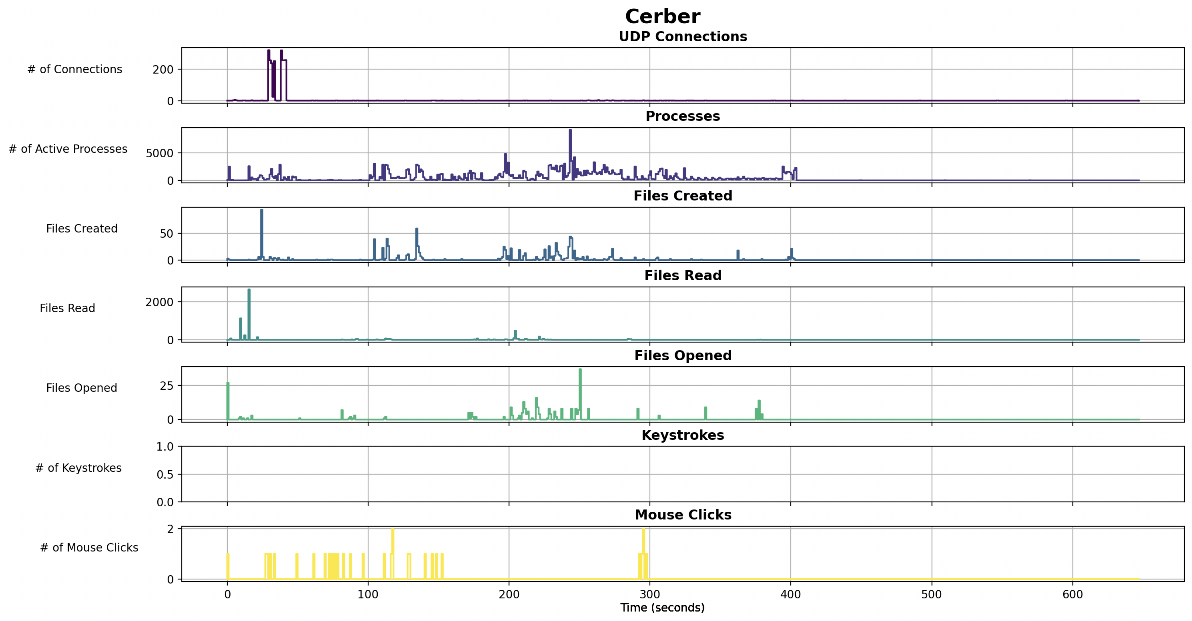

Figure 14: Run 2.5: Cerber

Run 2.5: Cerber Cerber's graph initially suggests an appearance of alignment between IOC data and

UI activity, marked by the occurrence of mouse clicks. However, these clicks were not the user interacting with the system in a typical manner, but rather the user following the instructions presented by the malware upon execution. This activity depicts the malware's method, which instructs the user to follow certain steps, potentially to facilitate its encryption process or to demand a ransom. Despite this seemingly aligned activity, the underlying IOCs such as the sporadic peaks in UDP connections and the continuous but variable background processes support the hypothesis of non-alignment. This is because the mouse clicks do not correlate with the system's micro-activities, which continue irrespective of user interaction, affirming the expected behavior of malware operating in the background.

In comparing the legitimate malware with our synthesized versions, we observe that the synthesized version of Locky demonstrates clear signs of the encryption process through heightened file modifications, similar to what was observed in the actual Crypto Locky activity. The synthesized Locky also mirrors its genuine counterpart with increased Tor2web and UDP connections when the macros file is opened. In both synthesized cases, the absence of significant UI activity alongside these system behaviors confirms the intended non-alignment between the system-level operations and user interactions.

Experiment 3: Benign Operations

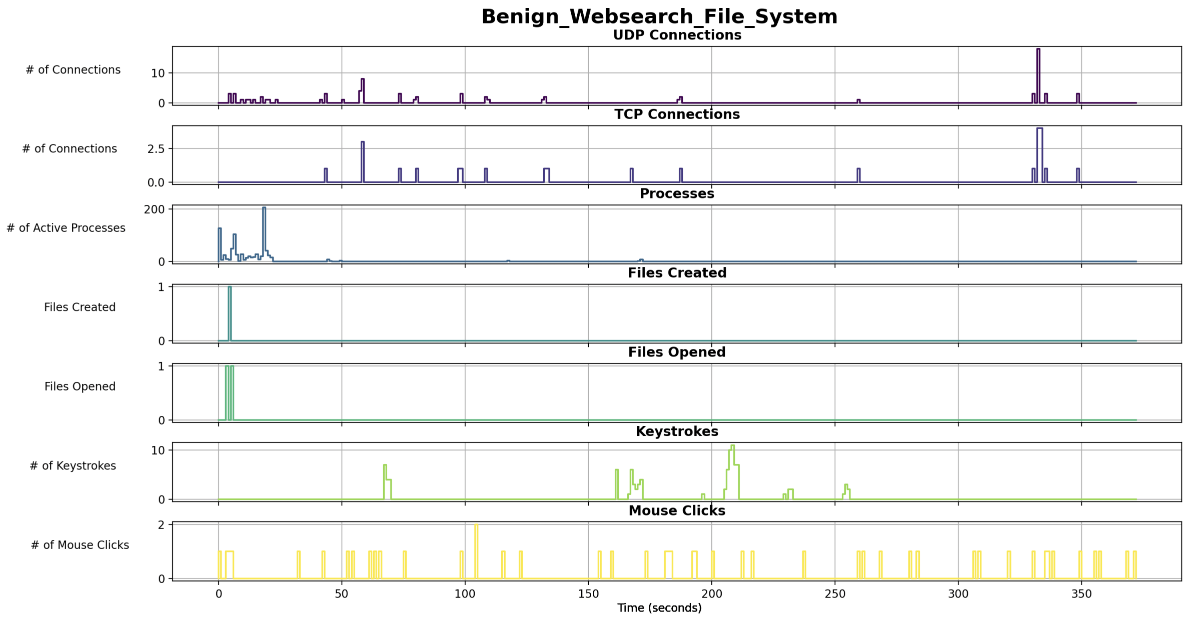

Figure 15: Run 3.1: Benign General Usage

Run 3.1: Expected Alignment during General Machine Usage The graph unveiled the anticipated alignment between UI and network data, as internet browsing coincided with concurrent mouse clicks and keylogs.

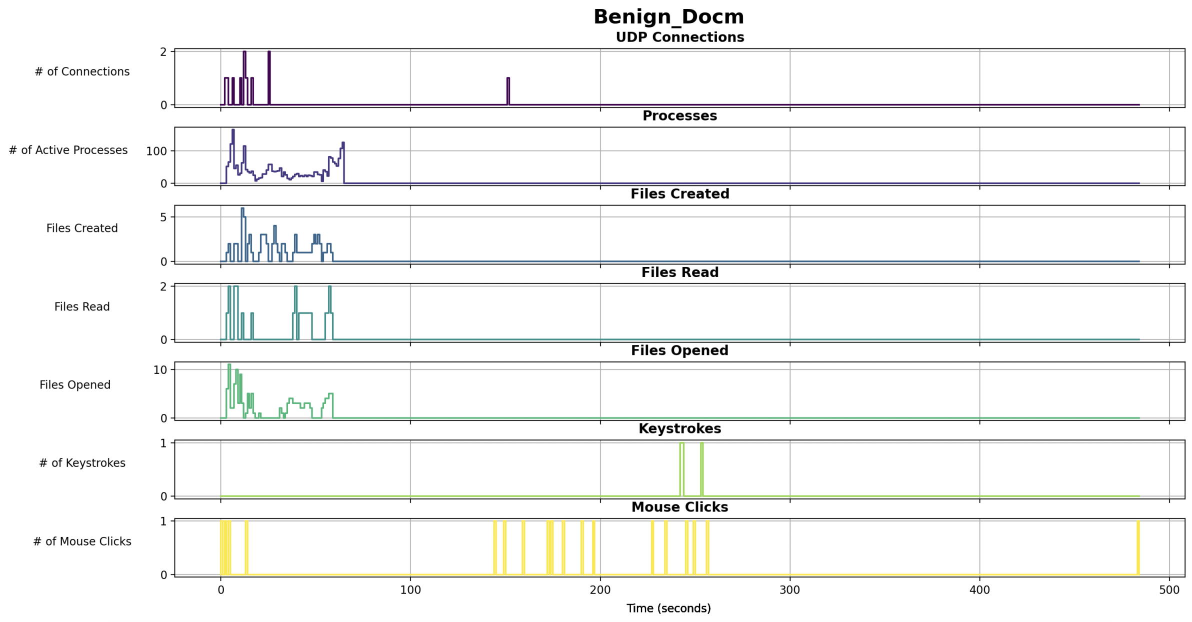

Figure 16: Run 3.2: Benign Docm Usage

Run 3.2: Expected Alignment during General Docm Usage The graph showed heightened UI and file modification data, though there was an unexpected non-alignment between the processes, files created/read, and keystroke data.

5 Limitations and Future Work Opportunities

The following section details limitations and potential future work for all components of the system. First we detail limitations and future work with the System for Isolating Environments. We then present limitations and future work with the System for Incorporating User Intent Data and VM Context Labels into Firewall Decision-Making. Lastly, we describe the limitations and future work of the Sandbox system.

5.1 System for Isolating Environments

One of the limitations with the system was with displaying/opening certain applications using Apache Guacamole and RDP. We utilized the Apache Guacamole HTTPTunnel "initial_application" field to open applications on the remote machine using the RDP protocol. This method does not work for every single application, it either fails to open the application or the RDP fails to transfer and display the single window. Further investigation is needed to determine the root cause, and a more robust custom approach to opening applications may need to be developed depending on findings.

Another limitation was the lack of window resizability. We did not implement window resizing over the Apache Guacamole RDP HTTPTunnel. Currently when the user resizes the web browser window, the actual application display inside the window is not resized, only the web browser window. Future work could explore or implement sending window size data over the Apache Guacamole RDP HTTPTunnel to resize remote applications.

An additional limitation was adding support for new applications. With current features, each application that the user needs to access needs to be manually added to the system. This consists of a lengthy process of creating a new VM image that includes the application, creating a new VM from that image in KVM, adding the VM UUID to the VMM REST API, and adding the button on the frontend webpage to trigger the request. Future work could explore streamlining/automating this process to allow system admins to easily add additional applications to the system.

Another limitation is the current "hot loading" implementation. To improve app startup time, we pre-loaded VMs with certain applications and then used the KVM manager to pause the virtual machine. This reduced startup time by keeping the entire VM's memory allocated, while saving resources by freeing CPU resources. This approach worked for a single user session, however when multiple users access the application, they share the same VM and application session for a given application. This violates user privacy, and proposes a potential security risk of users accessing other users data. Additionally, it is unknown how many VMs the system could support, while keeping all VMs allocated in memory. Future work could explore implementing user sessions, and allocating specific VMs to specific users. Additionally, future work could experiment with the resource load expected when the system is used to support multiple users.

A further limitation is with our performance and resource utilization testing. We tested a small number of scenarios, that tested running multiple applications at once. These scenarios only provided a small sample size; the nature of our scenarios resulted in performance data that was not representative of regular user workflow. Future work could explore creating a more robust set of scenarios that simulate a more realistic user workflow to stress test the system. Additionally future work could leverage more powerful hardware when stress testing that would be more representative of hardware used in industry.

There are some additional features that future work can explore adding to the system. The ability to automatically mount file systems to the VMs on application startup depending on what type of data is requested would provide support for further data isolation. A shared clipboard between application windows/VMs would provide additional usability to the system, improving the user experience.

During our background research into Qubes OS [8], we discovered a recent addition, the Qubes GUI Domain [40]. This provides the ability to use remote connection protocols to access Qubes applications. Future work could explore the feasibility of using this instead of KVM and the VMM REST API as the backend of the system.

5.2 Modified Harbinger for Improved Accuracy and Policy Scalability

The major limitation of our project was the non-functionality of the most updated version of Harbinger. The team was able to get around this limitation by reverting to an older version. In doing so we lost some functionality and optimizations that could change results. Further investigation into the bugs present in the most updated version of Harbinger, such as the driver's inability to connect to the controller, is required to merge the team's modifications with any benefits of using the most recent Harbinger update.

Another limitation of the project is limited testing against malware. The team predicts that due to the allow list structure, malware will be blocked and not be able to execute. However, there is evidence that non-malicious background process will get blocked as well. More testing, potentially with actual malware samples, will need to be done to understand the effect of blocking this traffic and ways to automatically to add these non-malicious process to the allow list.

In designing scenarios for testing the Harbinger+, the team introduced bias, especially in the designing of policies for each system. It is important for future testing to include additional scenarios that can demonstrate the strengths and the weaknesses of Harbinger+ in a variety of network setups, especially against firewalls that follow a deny list structure. The scenarios created by the team were also on a very small scale due to resource and time limitations, so testing in a larger environment with more machines, users, and realistic network traffic will create a more justifiable confirmation of policy scalability and firewall accuracy.

Policy scalability was numerical and therefore easy for the team to record and make calculations based on. The readability of policies is reliant on human experience. To capture policy readability, the perspectives of individuals with policy management experience, like network administrators, are required, meaning a user study must be performed. This comes with the difficulties of finding a sizable group of suitable individuals, study approval, and other issues that come with performing user studies. The user study is essential to seeing how Harbinger+ impacts policy readability, which is important for training and reducing human error when it comes to policy development.

5.3 Understanding UI Influence over Time Series Data

The research encountered several limitations and challenges throughout the experimentation process. One significant hurdle was the outdated and broken source code of Cuckoo. This necessitated hosting a customized version and implementing extensive fixes to ensure functionality. Additionally, the absence of a comprehensive installation guide posed a barrier to setting up the environment effectively. To overcome this, a series of Bash scripts were developed to automate the installation process, streamlining the setup procedure. Additionally, critical dependencies and analysis packages required for malware execution were missing, compelling the development of these components from scratch.

Within the testing environment, Cuckoo's restriction to analyzing only one sample at a time posed limitations on concurrent analysis scenarios. For instance, it was not possible to analyze multiple .docx files simultaneously within a Win7 VM, constraining the efficiency of experimentation. Also, the generated log reports by Cuckoo lacked comprehensive data, hindering a thorough understanding of the analyzed samples' behavior. This limitation made it difficult to obtain a comprehensive overview of observed activities. Additionally, the inability of Cuckoo to support Windows 10 or Windows 11 VMs restricted the analysis scope to other Windows versions, potentially limiting the relevance of findings. It is important to note that an overhaul of the Cuckoo project written in Python 3, known as Cuckoo 3, is currently in development at the time of writing and is designed to tackle many of the aforementioned limitations.

Our experimentation with synthetic malware was subject to inherent biases and assumptions, which may have influenced the interpretation of results. While efforts were made to emulate real-world ransomware based on expert-provided IOCs, the synthetic nature of our malware may not have fully encapsulated the intricacies and behaviors characteristic of authentic threats. We attempted to overcome this limitation by running additional experiments with legitimate malware.

In analyzing the missed critical IOCs and other challenges encountered during the creation of synthetic malware, attempts to simulate critical IOCs such as shadow copy deletion or file encryption within the synthetic locky malware were impeded by technical constraints within the LibreOffice environment. Issues with library compatibility and dependency management further heightened the complexity of replicating real-world scenarios accurately. Additionally, the absence of robustness metrics for the synthetic malware emphasizes the limitations in extrapolating findings to genuine threat landscapes.

In terms of data visualization, the absence of timestamps associated with collected data limited the analysis of temporal patterns and correlations. The inability to utilize the collected data in time graphs heightened this challenge, hindering the comprehensive examination of temporal relationships within the dataset. Incomplete tracking of file deletions due to the lack of timestamps hindered accurate reconstruction of deletion sequences, limiting insights derived from file-related activities. Additionally, the inability to visualize timestamps for HTTP and DNS requests, alongside the absence of information regarding high-entropy domain names and others, restricted the depth of analysis concerning network-related behaviors.

In addition, when filtering through data, we realized that while we are able to plot the information it is crucial to note that the graphs only plot when an event first occurs, not the full duration of an event. For example, the processes graph will only detail when a process starts, and does not show how long each process is running. Also, aligning the time buckets correctly in the graphs was another challenge, as the timestamps sometimes wouldn't match up exactly, meaning that sometimes we had to manually configure the graphs to be aligned chronologically.

6 Conclusion

This section summarizes our approaches to a variety of problems and our findings to address these problems. We summarize the work for a system for isolation environments, modified harbinger for improved accuracy and policy scalability, and understanding UI influence over time series data.

6.1 System for Isolating Environments

We addressed the risk of data breaches by creating a web interface that mirrors a traditional environment, allowing for seamless VM access and leverages remote computational resources to provide rapid availability of VMs. A current solution is running multiple VMs to separate assets, preventing contamination when one machine is compromised and other assets from being at risk. However, this requires multiple VMs to run concurrently and the user to manage multiple VMs. Our system addresses the usability and performance problems of existing solutions.

We tested our system against VirtualBox, a popular VM management application. Our experiments compared application startup time, wasted screen space, and the KLM time of VM access while using the traditional desktop as a base for comparison.

For application startup time, our system was slightly slower than the traditional desktop environment to start all applications, but was significantly faster than VirtualBox across all scenarios. Our system had 4.85% wasted screen space for screenspace, and VirtualBox contained 7.8% wasted screen space for the same application under the same circumstances. Our system's KLM time was 67.6% faster than VirtualBox, allowing users to access VMs quickly and efficiently.

We did not want to trade performance for usability; as a result, we compared the computing costs of our system to the traditional desktop and VirtualBox. Our system performed similarly to the traditional desktop environment and outperformed VirtualBox in the most intensive scenario. Additionally, our system allows greater scalability for the client than traditional desktop and VirtualBox since the client needs minimal resources per additional application. However, our server consumed the most resources in the most intensive scenario due to the overhead of single application isolation with VMs. Further work can be done in the future to optimize our system.

6.2 Modified Harbinger for Improved Accuracy and Policy Scalability

We also sought to balance policy scalability with security by incorporating user intent data and machine context labels into SDN-based decision making through the modification of the Harbinger tool. The team tested the Harbinger+ tool in three scenarios inspired by real-life security threats to confirm security was not not negatively affected by policy scalability.

Our findings show that Harbinger+ had similar accuracy to traditional network security systems and the Harbinger-integrated network security system while increasing policy scalability significantly. This was especially true in organizations in which users have multiple roles and roles with overlapping permissions because policies are created for groups based on role and machine context label, not static IP addresses associated with certain users.

Though initial results are promising, future work is needed to improve the Harbinger+ tool and justify its usability at an enterprise level. A more robust testing suite with real malware samples demonstrating functionality in a larger organization would demonstrate Harbinger+'s scalability capacity while limiting bias of the team in the creation of the original scenarios. A user study is also recommended by the team to see how Harbinger+ impacts policy readability.

6.3 Understanding UI Influence over Time Series Data

Our research combined Cuckoo Sandbox with a specialized Keylogger to enhance data collection and analysis for ransomware detection. Through thorough experimentation with synthetic malware scenarios, legitimate malware samples, and benign operations, we demonstrated the effectiveness of our methodology in identifying indicators of compromise and differentiating malicious activities from benign ones.

Our findings highlight the importance of correlating system-level activities with user interactions to gain insights into ransomware behavior. By analyzing the temporal alignment of UI and system activities, we were able to distinguish between malicious behaviors, such as file encryption and network connections, and legitimate user actions. This understanding provides valuable insights for improving ransomware detection methods against evolving cyber threats.

Moving forward, our research sets the stage for further refinement of detection mechanisms. Our proof of concept can be expanded by taking a more thorough look into time series data. The team collected time series data for 22 total experiments which may be inadequate for a comprehensive analysis. Gathering more experimental time series data at a large scale with an automation tool such as Sikuli [124] may provide more sufficient data. While definitive conclusions are difficult to draw from the limited experiments, expanding the dataset may provide more interesting trends between micro-event and macro-event alignment. Additionally, collecting more experimental data for non-malicious scenarios to test the hypothesis that there may be alignment between user activity and file, network, and process activity in benign scenarios may provide further insights into the details of system behavior under normal conditions.

Further future work may involve leveraging an intrusion classification algorithm based on recorded time series data. By training a model to identify misalignment over time series data, the efficacy of using UI activity data could be explored and compared with traditional machine learning based intrusion detection systems or log parsers.

An implementation of a partially observable Markov decision process to analyze real time data collection from within the sandbox may offer a sophisticated approach to dynamically adapt data collection strategies based on evolving system behaviors. This reinforcement model would identify misalignment by classifying the data to determine whether the system is under attack, not under attack, or if it requires more time to determine if an attack is underway, considerably cutting down on the response time to a potential security breach if one were to occur.

References

[1] SonicWall. *Annual number of malware attacks worldwide from 2015 to 2022*. Graph. Statista. Mar. 2023. URL: https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/ (visited on 02/21/2024).

[2] IBM. *Average cost of a data breach worldwide from May 2020 to March 2023, by industry*. Graph. Statista. July 2023. URL: https://www.statista.com/statistics/387861/cost-data-breach-by-industry/ (visited on 02/21/2024).

[3] *Why How Familiarity and Comfort Matter in UX Design: A Key Considerations for UX Designers*. Jan. 2023. URL: https://bootcamp.uxdesign.cc/why-how-familiarity-and-comfort-matter-in-ux-design-a-key-considerations-for-ux-designers-283fd35d4e93.

[4] Chao-Ming Wang and Ching-Hua Huang. "A study of usability principles and interface design for mobile e-books". In: *Ergonomics* 58.8 (2015). PMID: 25680001, pp. 1253–1265. DOI: 10.1080/00140139.2015.1013577. eprint: https://doi.org/10.1080/00140139.2015.1013577. URL: https://doi.org/10.1080/00140139.2015.1013577.

[5] *Estimating Memory Requirements for Virtual Machine Desktops*. 2024. URL: https://docs.vmware.com/en/VMware-Horizon/2303-and-later/horizon-architecture-planning/GUID-10ED49A9-56B6-4BD7-8425-94900F53AB02.html#:~:text=A%20good%20starting%20point%20is,and%204%20GB%20of%20RAM..

[6] Apache Guacamole. *Apache GuacamoleTM*. https://guacamole.apache.org/.

[7] URL: https://www.securityweek.com/state-firewall-report-automation-key-preventing-costly-misconfigurations/.

[8] *Qubes OS: What is Qubes OS?* Oct. 2023. URL: https://www.qubes-os.org/intro.

[9] *Qubes OS Project. Qubes Air: Generalizing the Qubes Architecture*. Jan. 2018. URL: https://www.qubes-os.org/news/2018/01/22/qubes-air/#example-hybrid-mode.

[10] *Docker Overview*. URL: https://docs.docker.com/get-started/overview/.

[11] Microsoft. *AppContainer isolation*. Microsoft. 2023. URL: https://learn.microsoft.com/en-us/windows/win32/secauthz/appcontainer-isolation.

[12] *Qubes OS Project. Frequently asked questions (FAQ)*. Aug. 2021. URL: https://www.qubes-os.org/faq/.

[13] URL: https://media.defense.gov/2013/Sep/30/2001713314/-1/-1/1/DODIG-2013-142.pdf.

[14] URL: https://www.dcsa.mil/Personnel-Security/Background-Investigations-for-Security-HR-Professionals/billing_rates/.

[15] URL: https://wiki.xenproject.org/wiki/Xen_Networking#Virtual_Network_Interfaces.

[16] URL: https://www.cisco.com/en/US/docs/security/ise/1.0/user_guide/ise10_authz_polprfls.html.

[17] URL: https://sdphelp.appgate.com/adminguide/v6.2/policies-configure.html.

[18] URL: https://www.appgate.com/blog/sdp-and-risky-devices-dynamic-controls-for-secure-access.

[19] *Deep Malware Analysis - Joe Sandbox Technology.* URL: https://www.joesecurity.org/joe-sandbox-technology# (visited on 10/13/2023).

[20] "Bromium twists chip virty circuits to secure PCs and servers". In: (). URL: https://www.theregister.com/2012/06/20/bromium_microvisor_security/.

[21] "Cost of a Data Breach Report 2023". In: (). URL: https://www.ibm.com/reports/data-breach.

[22] *KVM.* 2023. URL: https://linux-kvm.org/page/Main_Page.

[23] *freerdp.* 2023. URL: https://www.freerdp.com/.

[24] *What is a virtual machine (VM)?* May 2022. URL: https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine.

[25] *How is Qubes OS different from...* Sept. 2012. URL: https://blog.invisiblethings.org/2012/09/12/how-is-qubes-os-different-from.html#:~:text=Second%2C%20all%20mainstream,SELinux%2C%20LXC%2C%20etc.

[26] Yunfa Li, Wanqing Li, and Congfeng Jiang. "A Survey of Virtual Machine System: Current Technology and Future Trends". In: *2010 Third International Symposium on Electronic Commerce and Security.* 2010, pp. 332–336. DOI: 10.1109/ISECS.2010.80.

[27] Erick Bauman, Gbadebo Ayoade, and Zhiqiang Lin. "A Survey on Hypervisor-Based Monitoring: Approaches, Applications, and Evolutions". In: *ACM Comput. Surv.* 48.1 (Aug. 2015). ISSN: 0360-0300. DOI: 10.1145/2775111. URL: https://doi.org/10.1145/2775111.

[28] Thomas Kobber Panum et al. "Haaukins: A highly accessible and automated virtualization platform for security education". In: *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT).* Vol. 2161. IEEE. 2019, pp. 236–238.

[29] Lily Sturmann. *What is a Trusted ZComputing Base*. June 2021. URL: `https://next.redhat.com/2021/06/18/what-is-a-trusted-computing-base/`.

[30] Jason Landry. *What is Hardware Root of Trust*. July 2019. URL: `https://www.dell.com/en-us/blog/hardware-root-trust/`.

[31] Vincent Zimmer and Michael Krau. "Establishing the root of trust". In: *UEFI. org document dated August* (2016).

[32] Sofija Simic. *What is a hypervisor? types of hypervisors explained (1 & 2)*. Sept. 2023. URL: `https://phoenixnap.com/kb/what-is-hypervisor-type-1-2`.

[33] Todd Deshane et al. "Quantitative comparison of Xen and KVM". In: *Xen Summit, Boston, MA, USA* (2008), pp. 1–2.

[34] A. Binu and G. Santhosh Kumar. "Virtualization Techniques: A Methodical Review of XEN and KVM". In: *Advances in Computing and Communications*. Ed. by Ajith Abraham et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 399–410. ISBN: 978-3-642-22709-7.

[35] A. inu and G.S. Kumar. "Virtualization Techniques: A Methodical Review of XEN and KVM". In: *Advances in Computing and Communications*. Ed. by A. Abraham et al. Vol. 190. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2011. DOI: `10.1007/978-3-642-22709-7_40`. URL: `https://doi.org/10.1007/978-3-642-22709-7_40`.

[36] B. DRAGOVIC. "Xen and the Art of Virtualization". In: *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles, New York, U.S.A., October* (2003). URL: `https://cir.nii.ac.jp/crid/1570572701183474944`.

[37] *Qubes OS: a reasonably secure operating system.* Aug. 2021. URL: `https://www.qubes-os.org/`.

[38] *Architecture*. Oct. 2023. URL: `https://www.qubes-os.org/doc/architecture/`.

[39] *GUI virtualization*. Oct. 2023. URL: `https://www.qubes-os.org/doc/gui/`.

[40] *Qubes Architecture Next Steps: The GUI Domain*. Mar. 2020. URL: `https://www.qubes-os.org/news/2020/03/18/gui-domain/`.

[41] "Cuckoo Sandbox". In: (). URL: `https://cuckoosandbox.org/`.

[42] Qian Chen and Robert A. Bridges. *Automated Behavioral Analysis of Malware A Case Study of WannaCry Ransomware*. arXiv:1709.08753 [cs]. Sept. 2017. URL: `http://arxiv.org/abs/1709.08753` (visited on 09/30/2023).

[43] Aaron Walker, Muhammad Faisal Amjad, and Shamik Sengupta. “Cuckoo’s Malware Threat Scoring and Classification: Friend or Foe?” In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. Jan. 2019, pp. 0678–0684. DOI: `10.1109/CCWC.2019.8666454`. URL: `https://ieeexplore.ieee.org/abstract/document/8666454` (visited on 10/11/2023).

[44] Athina Provataki and Vasilios Katos. “Differential malware forensics”. en. In: *Digital Investigation* 10.4 (Dec. 2013), pp. 311–322. ISSN: 17422876. DOI: `10.1016/j.diin.2013.08.006`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S1742287613000923` (visited on 09/30/2023).

[45] Avira. *Cuckoo Sandbox vs. Reality*. en-US. Nov. 2014. URL: `https://www.avira.com/en/blog/cuckoo-sandbox-vs-reality-2` (visited on 10/11/2023).

[46] *Cuckoo Sandbox - Automated Malware Analysis*. URL: `https://cuckoosandbox.org/#:~:text=Cuckoo%20Sandbox%20is%20free%20software,macOS%2C%20Linux%2C%20and%20Android` (visited on 10/11/2023).

[47] *What is Cuckoo? — Cuckoo Sandbox v2.0.7 Book*. URL: `https://cuckoo.readthedocs.io/en/latest/introduction/what/` (visited on 10/11/2023).

[48] *Malware Analysis is for the (Cuckoo) Birds*. en-US. URL: `https://trustedsec.com/blog/malware-cuckoo-1` (visited on 10/11/2023).

[49] *Cuckoo Sandbox Reviews, Competitors and Pricing*. URL: `https://www.peerspot.com/products/cuckoo-sandbox-reviews` (visited on 10/11/2023).

[50] Sherban Naum. “Bromium Secure Platform Powers HP Sure Click Advanced”. In: (2019). URL: `https://threatresearch.ext.hp.com/bromium-powers-hp-sure-click-advanced/`.

[51] “Bromium: A virutalization technology to kill all malware, forever”. In: (). URL: `https://www.zdnet.com/article/bromium-a-virtualization-technology-to-kill-all-malware-forever/`.

[52] “Bromium Installation and Deployment Guide”. In: (). URL: `https://documentation.bromium.com/4_2/Deployment%20Guide/HP_SureClick_Enterprise_4.2_Deployment_Guide.pdf`.

[53] *Deep Malware Analysis - Joe Sandbox Reports*. URL: `https://www.joesecurity.org/joe-sandbox-reports#windows` (visited on 10/13/2023).

[54] *Deep Malware Analysis - Joe Sandbox Hypervisor*. URL: `https://www.joesecurity.org/joe-sandbox-hypervisor#key-features` (visited on 10/13/2023).

[55] “Joe Sandbox Cloud Likes and Dislikes”. In: (). URL: `https://www.gartner.com/reviews/market/security-solutions-others/vendor/joe-security/product/joe-sandbox-cloud/likes-dislikes`.

[56] *Deep Malware Analysis - Joe Sandbox*. URL: https://www.joesecurity.org/# (visited on 10/13/2023).

[57] No_Shift_Buckwheat. *Finally got them to . . .* Reddit Comment. Post URL: www.reddit.com/r/Malware/comments/s27 Jan. 2022. URL: www.reddit.com/r/Malware/comments/s27592/joes_sandbox_pricing/hsfdx74/ (visited on 10/12/2023).

[58] "Comparison of Various Sandboxes for Basic Dynamic Analysis". In: ().

[59] Carsten Willems, Thorsten Holz, and Felix Freiling. "Toward Automated Dynamic Malware Analysis Using CWSandbox". In: *IEEE Security and Privacy Magazine* 5.2 (Mar. 2007), pp. 32–39. ISSN: 1540-7993. DOI: 10.1109/MSP.2007.45. URL: http://ieeexplore.ieee.org/document/4140988/ (visited on 10/02/2023).

[60] "GFI Sandbox". In: (). URL: https://www.gfi.com/products-and-solutions.

[61] Cai Longzheng, Yu Shengsheng, and Zhou Jing-li. "Research and implementation of remote desktop protocol service over SSL VPN". In: *IEEE International Conference onServices Computing, 2004.(SCC 2004). Proceedings. 2004*. IEEE. 2004, pp. 502–505.

[62] Reinhard Langmann and Steffen Arts. "Application virtualization in virtual learning labs". In: *2012 9th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. IEEE. 2012, pp. 1–4.

[63] Charles Border. "The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes". In: *ACM SIGCSE Bulletin* 39.1 (2007), pp. 576–580.

[64] Tristan Richardson et al. "Virtual network computing". In: *IEEE Internet Computing* 2.1 (1998), pp. 33–38.

[65] Yun Lin et al. "Key technologies and solutions of remote distributed virtual laboratory for E-learning and E-education". In: *Mobile Networks and Applications* 24 (2019), pp. 18–24.

[66] Xiaolin Lu. "Construct collaborative distance learning environment with vnc technology". In: *2005 First International Conference on Semantics, Knowledge and Grid*. IEEE. 2005, pp. 127–127.

[67] Tae-Ho Lee et al. "Extending VNC for effective collaboration". In: *2008 Third International Forum on Strategic Technologies*. IEEE. 2008, pp. 343–346.

[68] *SSH protocol is the standard for strong authentication, secure connection, and encrypted file transfers. We developed it.* URL: https://www.ssh.com/academy/ssh/protocol#ssh-provides-strong-encryption-and-integrity-protection.

[69] Apache Guacamole. *Apache Guacamole Manual—Apache Guacamole Manual v1.5.3.* https://guacamole.apache.org/doc/gug/.

[70] Apache Guacamole. *Apache GuacamoleTM: API Documentation.* https://guacamole.apache.org/api-documentation/.

[71] Sebastián García et al. "Remote Lab Access: A Powerful Tool Beyond the Pandemic". In: *2022 Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (XV Technologies Applied to Electronics Teaching Conference).* 2022, pp. 1–5. DOI: 10.1109/TAEE54169.2022.9840672.

[72] Ismail Hassan. "Levereging Apache Guacamole, Linux LXD and Docker Containers to Deliver a Secure Online Lab for a Large Cybersecurity Course". In: *2022 IEEE Frontiers in Education Conference (FIE).* IEEE. 2022, pp. 1–9.

[73] Stefan Axelsson. "The base-rate fallacy and the difficulty of intrusion detection". In: *ACM Transactions on Information and System Security* 3.3 (Aug. 2000), pp. 186–205. ISSN: 1094-9224. DOI: 10.1145/357830.357849. URL: https://dl.acm.org/doi/10.1145/357830.357849 (visited on 02/27/2024).

[74] P. García-Teodoro et al. "Anomaly-based network intrusion detection: Techniques, systems and challenges". In: *Computers & Security* 28.1 (Feb. 2009), pp. 18–28. ISSN: 0167-4048. DOI: 10.1016/j.cose.2008.08.003. URL: https://www.sciencedirect.com/science/article/pii/S0167404808000692 (visited on 02/27/2024).

[75] *Splunk — The Key to Enterprise Resilience.* en. URL: https://www.splunk.com (visited on 02/27/2024).

[76] *Security QRadar — IBM.* en-us. URL: https://www.ibm.com/qradar (visited on 02/27/2024).

[77] Nicolas Aussel, Yohan Petetin, and Sophie Chabridon. "Improving Performances of Log Mining for Anomaly Prediction Through NLP-Based Log Parsing". In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS).* ISSN: 2375-0227. Sept. 2018, pp. 237–243. DOI: 10.1109/MASCOTS.2018.00031.

[78] Sasho Nedelkoski et al. "Self-supervised Log Parsing". en. In: *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track.* Ed. by Yuxiao Dong, Dunja Mladenić, and Craig Saunders. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 122–138. ISBN: 978-3-030-67667-4. DOI: 10.1007/978-3-030-67667-4_8.

[79] *Suricata Features.* Suricata. URL: https://suricata.io/features/ (visited on 02/15/2024).

[80] Simon Parkinson and Andrew Crampton. "A Novel Software Tool for Analysing NT File System Permissions". In: *International Journal of Advanced Computer Science and Applications* 4.6 (2013). arXiv:1312.2804 [cs]. ISSN: 2158107X, 21565570. DOI: 10.14569/IJACSA.2013.040635. URL: http://arxiv.org/abs/1312.2804 (visited on 09/17/2023).

[81] Zorigtbaatar Chuluundorj et al. "Can the User Help? Leveraging User Actions for Network Profiling". In: *Swiss Conference on Data Science* (2021). DOI: 10.1109/sds54264.2021.9732164.

[82] Gloria J. Serrao. "Network access control (NAC): An open source analysis of architectures and requirements". In: *44th Annual 2010 IEEE International Carnahan Conference on Security Technology*. 2010, pp. 94–102. DOI: 10.1109/CCST.2010.5678694.

[83] Shimon Ifrah. "Get Started with Microsoft Azure". In: *Getting Started with Containers in Azure : Deploy, Manage, and Secure Containerized Applications*. Berkeley, CA: Apress, 2020, pp. 1–26. ISBN: 978-1-4842-5753-1. DOI: 10.1007/978-1-4842-5753-1_1. URL: https://doi.org/10.1007/978-1-4842-5753-1_1.

[84] David F. Ferraiolo et al. "Proposed NIST Standard for Role-Based Access Control". In: *ACM Trans. Inf. Syst. Secur.* 4.3 (Aug. 2001), pp. 224–274. ISSN: 1094-9224. DOI: 10.1145/501978.501980. URL: https://doi.org/10.1145/501978.501980.

[85] Jeffrey Pang et al. "On the Responsiveness of DNS-Based Network Control". In: *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. IMC '04. Taormina, Sicily, Italy: Association for Computing Machinery, 2004, pp. 21–26. ISBN: 1581138210. DOI: 10.1145/1028788.1028792. URL: https://doi.org/10.1145/1028788.1028792.

[86] Kevin Townsend. *"state of the Firewall" report: Automation key to preventing costly misconfigurations*. Nov. 2019. URL: https://www.securityweek.com/state-firewall-report-automation-key-preventing-costly-misconfigurations/.

[87] Rajpreet Kaur, Adam Hils, and John Watts. *Technology insight for network security policy management*. Feb. 2019. URL: https://www.gartner.com/en/documents/3902564.

[88] Mahwish Anwar. "Virtual firewalling for migrating virtual machines in cloud computing". In: *2013 5th International Conference on Information and Communication Technologies* (2013). DOI: 10.1109/icict.2013.6732787.

[89] *guacamole-common*. 2023. URL: https://guacamole.apache.org/doc/gug/guacamole-common-js.html.

[90] *flask*. 2023. URL: https://flask.palletsprojects.com/en/3.0.x/.

[91] *pynput 1.7.6.* 2022. URL: https://pypi.org/project/pynput/.

[92] *pywinctl.* 2023. URL: https://github.com/Kalmat/PyWinCtl.

[93] *pyautogui.* 2023. URL: https://github.com/asweigart/pyautogui.

[94] *imageio.* 2023. URL: https://github.com/imageio/imageio.

[95] *PyGetWindow 0.0.9.* 2020. URL: https://pypi.org/project/PyGetWindow/.

[96] *win32gui 221.6.* 2017. URL: https://pypi.org/project/win32gui/.

[97] *Accessibility Insights for Web.* 2024. URL: https://accessibilityinsights.io/docs/web/overview/.

[98] Chauncey Wilson Ben Werner. "KLM-GOMS". In: *Usability Body of Knowledge* (2011). URL: https://www.usabilitybok.org/klm-goms.

[99] *psrecord.* 2023. URL: https://pypi.org/project/psrecord/.

[100] *Wireshark Home Page.* URL: https://www.wireshark.org/.

[101] URL: https://www.mckinsey.com/capabilities/risk-and-resilience/our-insights/insider-threat-the-human-element-of-cyberrisk.

[102] Steven M Bellovin. "The insider attack problem nature and scope". In: *Insider Attack and Cyber Security: Beyond the Hacker.* Springer, 2008, pp. 1–4.

[103] Mayank Verma et al. "Analysing Indicator of Compromises for Ransomware: Leveraging IOCs with Machine Learning Techniques". In: *2018 IEEE International Conference on Intelligence and Security Informatics (ISI).* 2018, pp. 154–159. DOI: 10.1109/ISI.2018.8587409.

[104] "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics". en-US. In: *Computers & Electrical Engineering* 66 (Feb. 2018). Publisher: Pergamon, pp. 353–368. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2017.10.012. URL: https://www.sciencedirect.com/science/article/pii/S0045790617333542 (visited on 12/15/2023).

[105] *Tor2Web — Tor Project — Support.* URL: https://support.torproject.org/glossary/tor2web/ (visited on 12/15/2023).

[106] *Threat hunting: DNS IoCs — ManageEngine.* URL: https://www.manageengine.com/products/eventlog/cyber-security/dns-iocs.html#:~:text=Abnormal%20volume%20of%20DNS%20%3A%20When,the%20querying%20systems%20are%20infected (visited on 12/15/2023).

[107] *User Account Control – Overview and Exploitation.* en-US. URL: https://www.cynet.com/attack-techniques-hands-on/user-account-control-overview-and-exploitation/ (visited on 12/15/2023).

[108] *What is Task Manager? — Definition TechTarget.* en. URL: https://www.techtarget.com/searchenterprisedesktop/definition/Microsoft-Windows-Task-Manager (visited on 12/15/2023).

[109] lenewsad. *Turn on Microsoft Defender Antivirus on enrolled device.* en-us. June 2023. URL: https://learn.microsoft.com/en-us/mem/intune/user-help/turn-on-defender-windows (visited on 12/15/2023).

[110] *Stay protected with Windows Security - Microsoft Support.* URL: https://support.microsoft.com/en-us/windows/stay-protected-with-windows-security-2ae0363d-0ada-c064-8b56-6a39afb6a963#:~:text=Windows%20Security%20is%20your%20home,Windows%2010%20in%20S%20mode (visited on 12/15/2023).

[111] *Windows Update: FAQ - Microsoft Support.* URL: https://support.microsoft.com/en-us/windows/windows-update-faq-8a903416-6f45-0718-f5c7-375e92dddeb2 (visited on 12/15/2023).

[112] Deland-Han. *Windows Error Reporting and Windows diagnostics enablement guidance - Windows Client.* en-us. Feb. 2023. URL: https://learn.microsoft.com/en-us/troubleshoot/windows-client/system-management-components/windows-error-reporting-diagnostics-enablement-guidance (visited on 12/15/2023).

[113] paolomatarazzo. *Windows Firewall overview - Windows Security.* en-us. Nov. 2023. URL: https://learn.microsoft.com/en-us/windows/security/operating-system-security/network-security/windows-firewall/ (visited on 12/15/2023).

[114] stevewhims. *Background Intelligent Transfer Service - Win32 apps.* en-us. May 2021. URL: https://learn.microsoft.com/en-us/windows/win32/bits/background-intelligent-transfer-service-portal (visited on 12/15/2023).

[115] *Threat Hunting for File Names as an IoC — Infosec.* URL: https://resources.infosecinstitute.com/topics/threat-hunting/threat-hunting-for-file-names-as-an-ioc/ (visited on 12/15/2023).

[116] “Using YARA for Malware Detection”. en. In: ().

[117] *Signature — Malwarebytes Glossary.* en. URL: https://www.malwarebytes.com/glossary/signature (visited on 12/15/2023).

[118] *dropped_files*. en. URL: `https://virustotal.readme.io/reference/dropped_files` (visited on 12/15/2023).

[119] *CryptoLocker Ransomware Threat Analysis*. en. URL: `https://www.secureworks.com/research/cryptolocker-ransomware` (visited on 12/15/2023).

[120] *Process Injection - Red Canary Threat Detection Report*. en. URL: `https://redcanary.com/threat-detection-report/techniques/process-injection/` (visited on 12/15/2023).

[121] Timothy R. McIntosh, Julian Jang-Jaccard, and Paul A. Watters. "Large Scale Behavioral Analysis of Ransomware Attacks". en. In: *Neural Information Processing*. Ed. by Long Cheng, Andrew Chi Sing Leung, and Seiichi Ozawa. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 217–229. ISBN: 978-3-030-04224-0. DOI: `10.1007/978-3-030-04224-0_19`.

[122] windows-driver-content. *ProgramData*. en-us. Jan. 2019. URL: `https://learn.microsoft.com/en-us/windows-hardware/customize/desktop/unattend/microsoft-windows-shell-setup-folderlocations-programdata` (visited on 12/15/2023).

[123] *What Is Ransomware? — Trellix*. en. URL: `https://www.trellix.com/security-awareness/ransomware/what-is-ransomware/` (visited on 12/15/2023).

[124] *GitHub - RaiMan/SikuliX1: SikuliX version 2.0.0+ (2019+)*. URL: `https://github.com/RaiMan/SikuliX1` (visited on 03/13/2024).

Appendix A

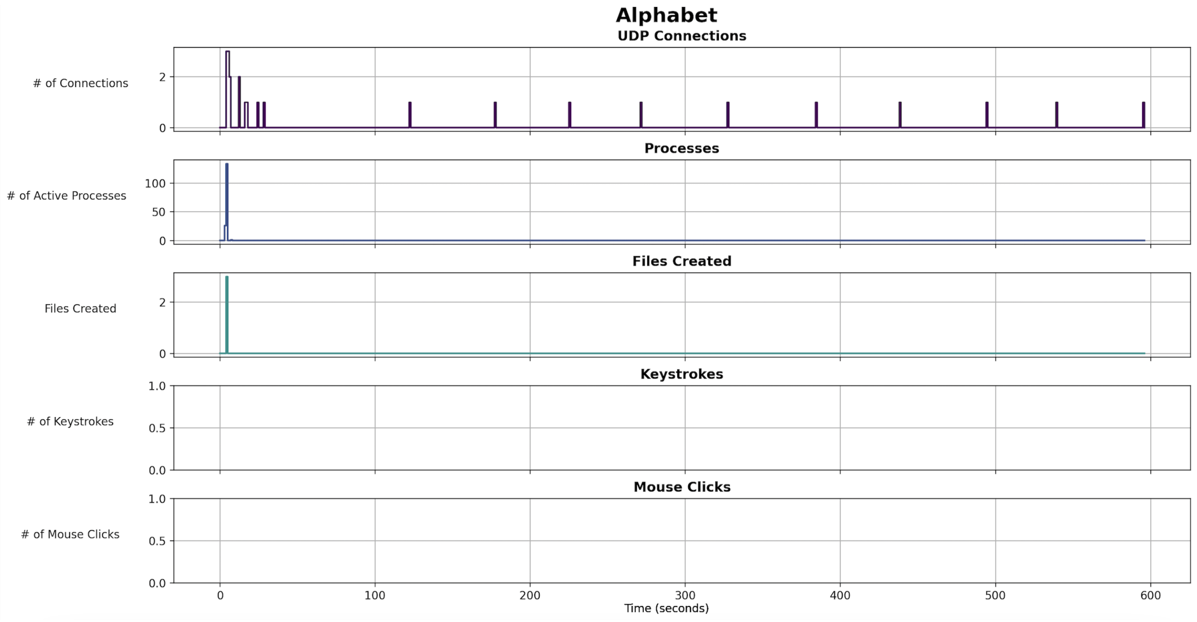

Figure A.1: Alphabet

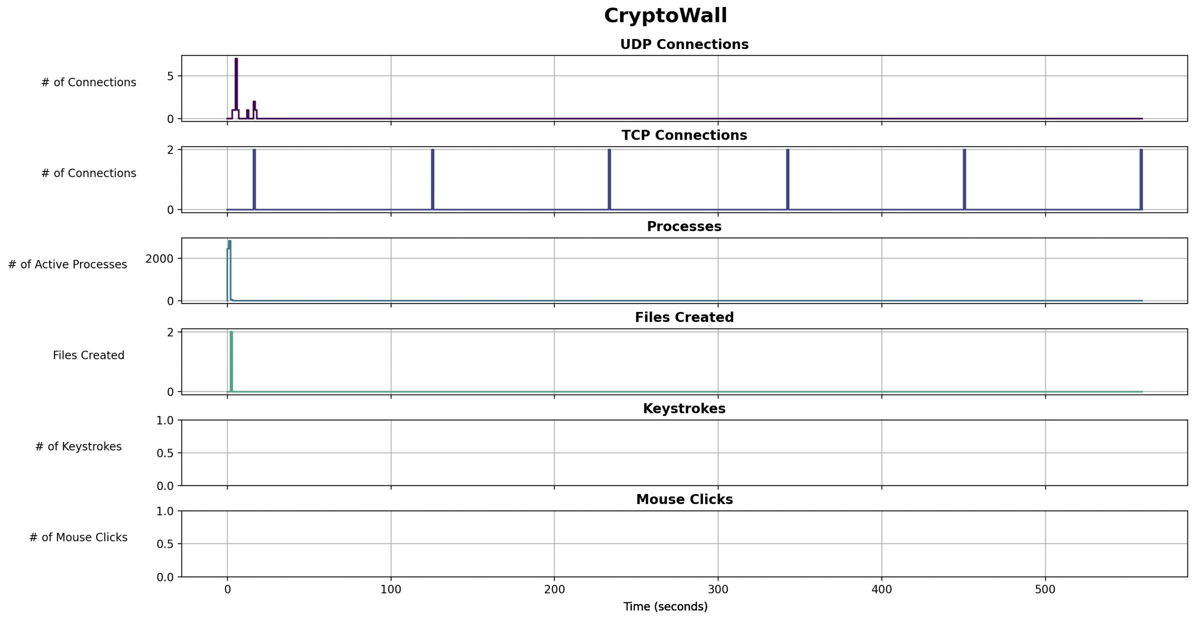

Figure A.2: CryptoWall

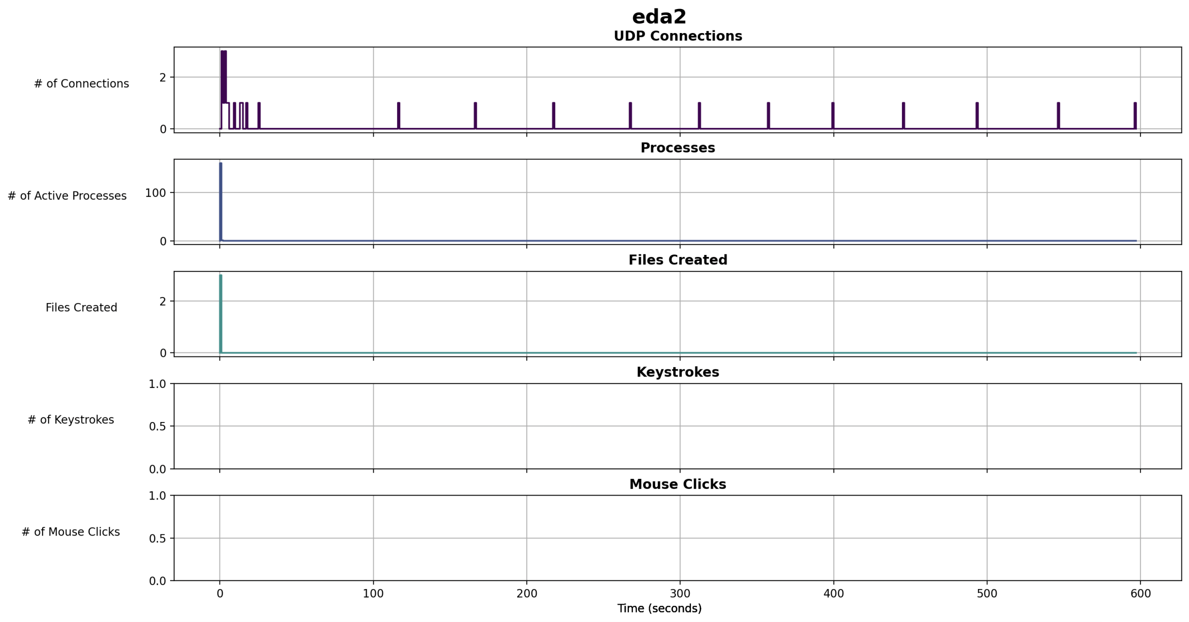

Figure A.3: EDA2

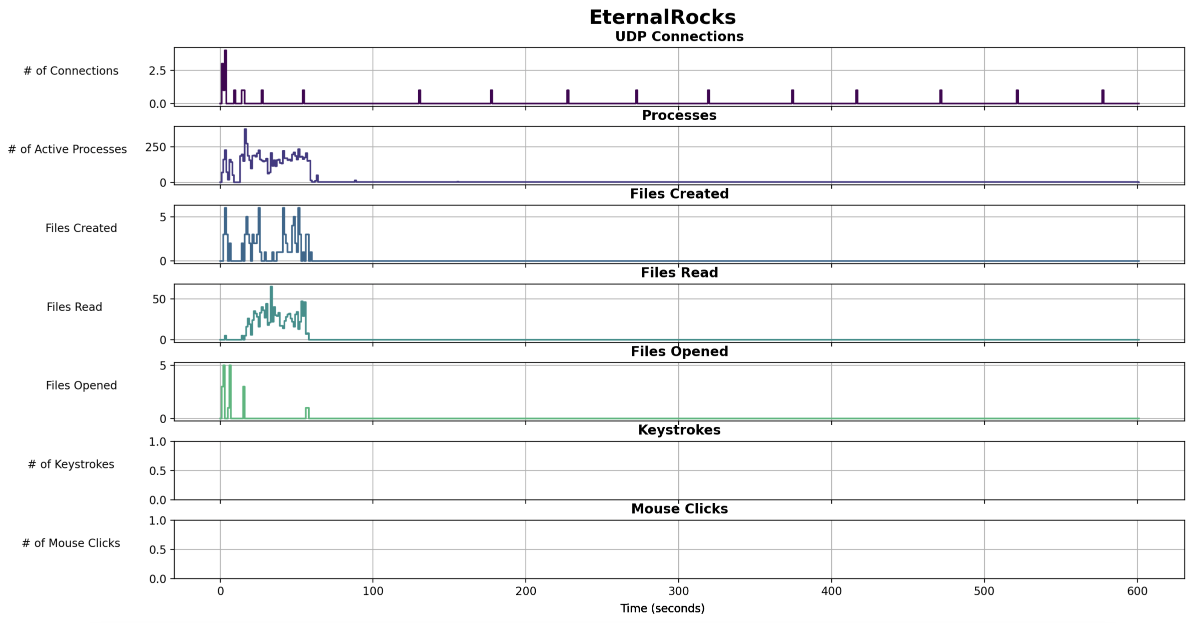

Figure A.4: EternalRocks

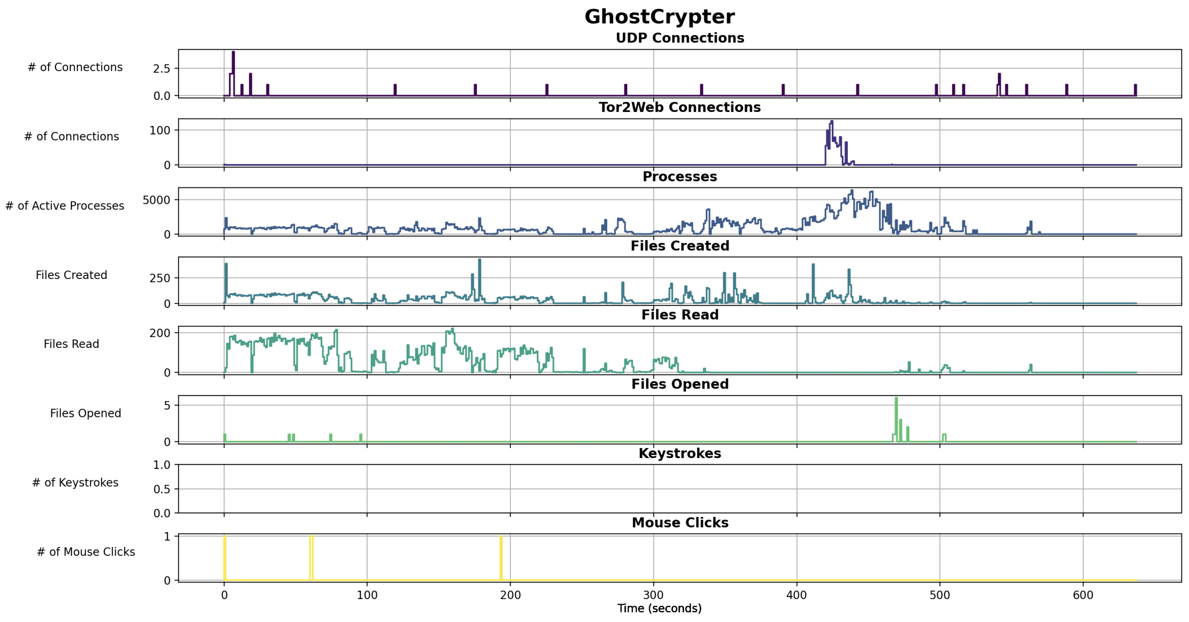

Figure A.5: GhostCrypter

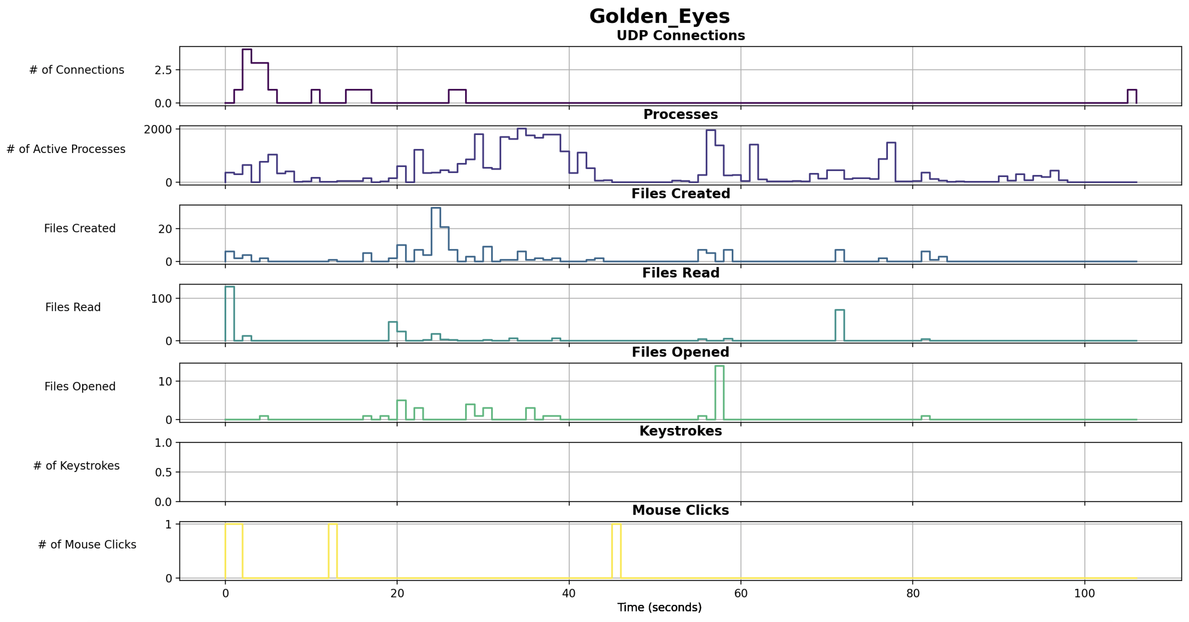

Figure A.6: Golden Eyes

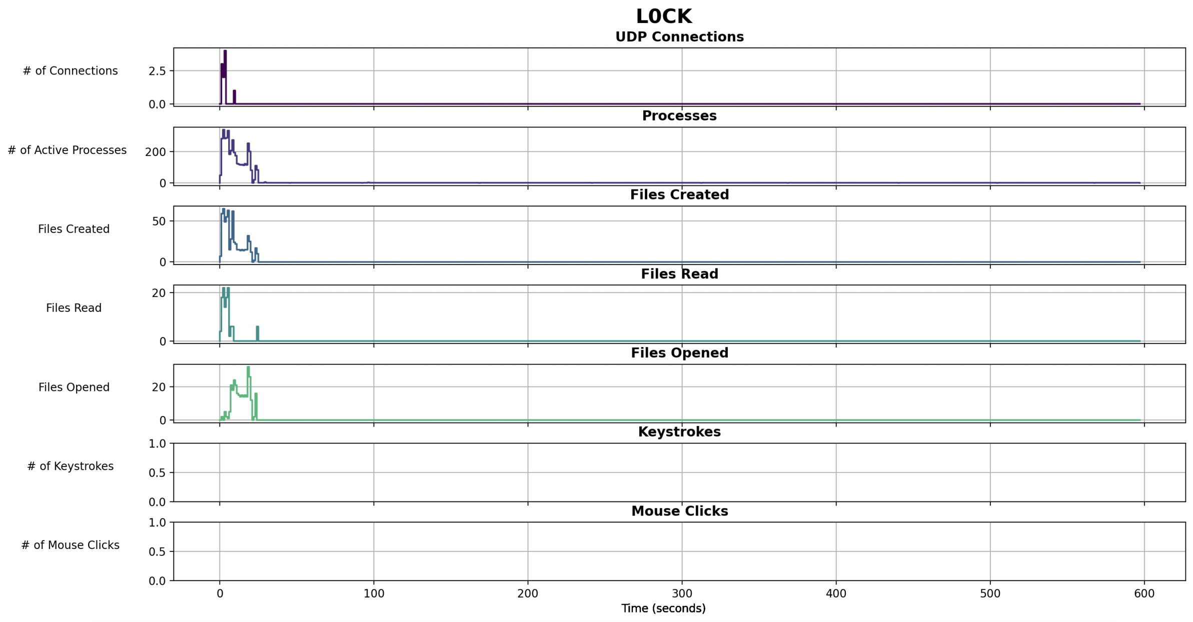

Figure A.7: L0CK

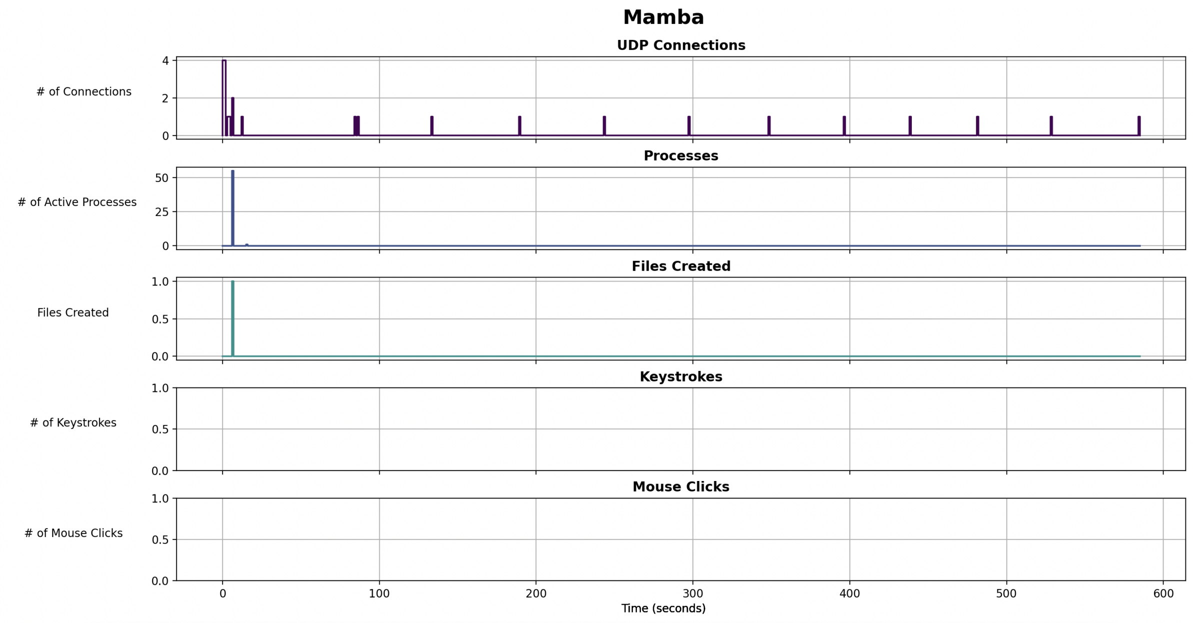

Figure A.8: Mamba

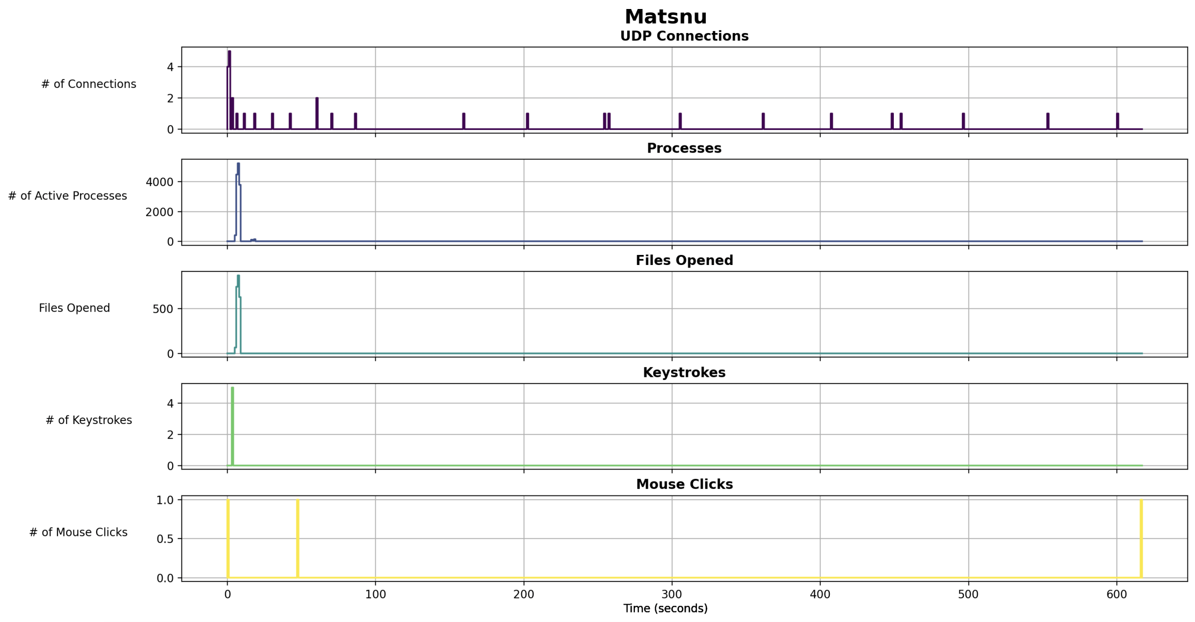

Figure A.9: Matsnu

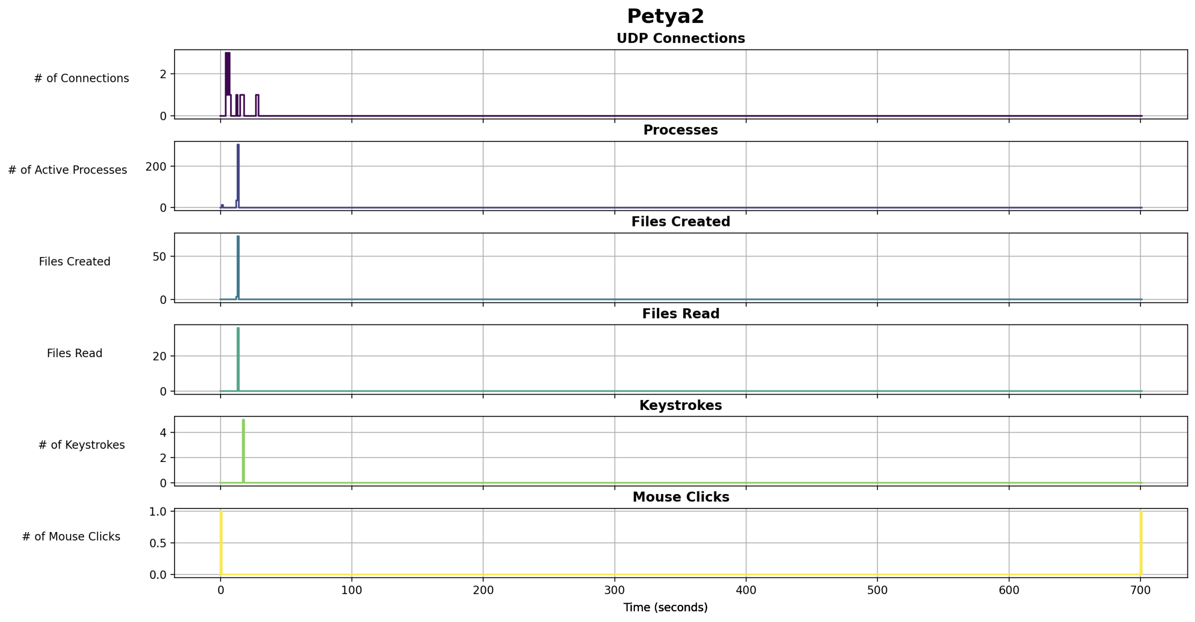

Figure A.10: Petya2

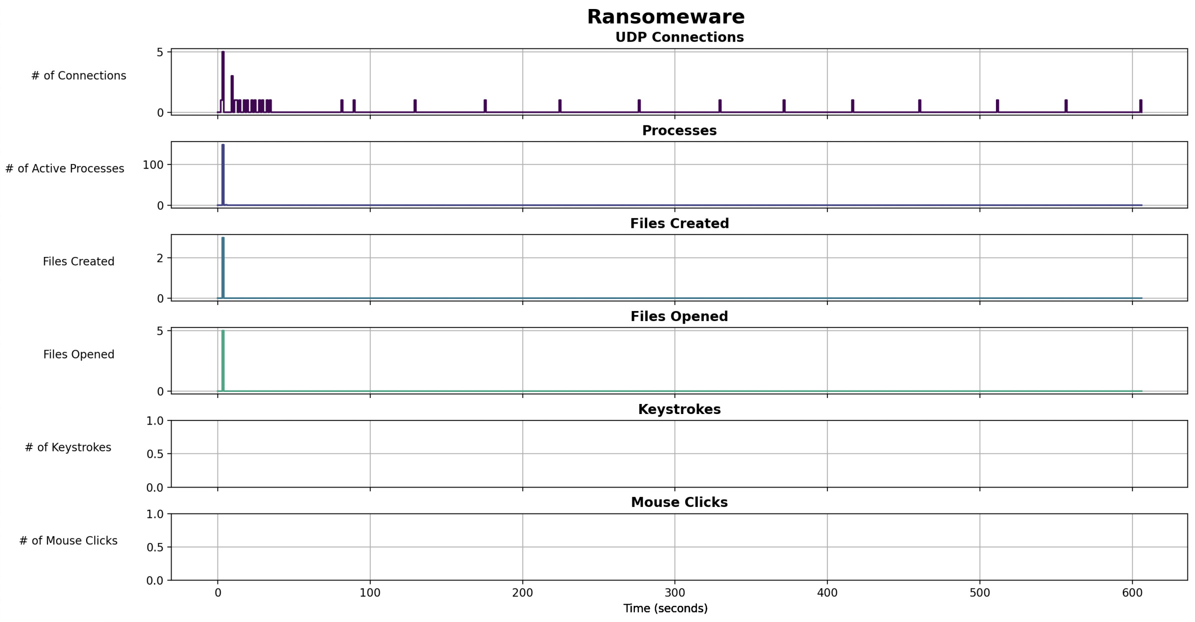

Figure A.11: Ransomeware

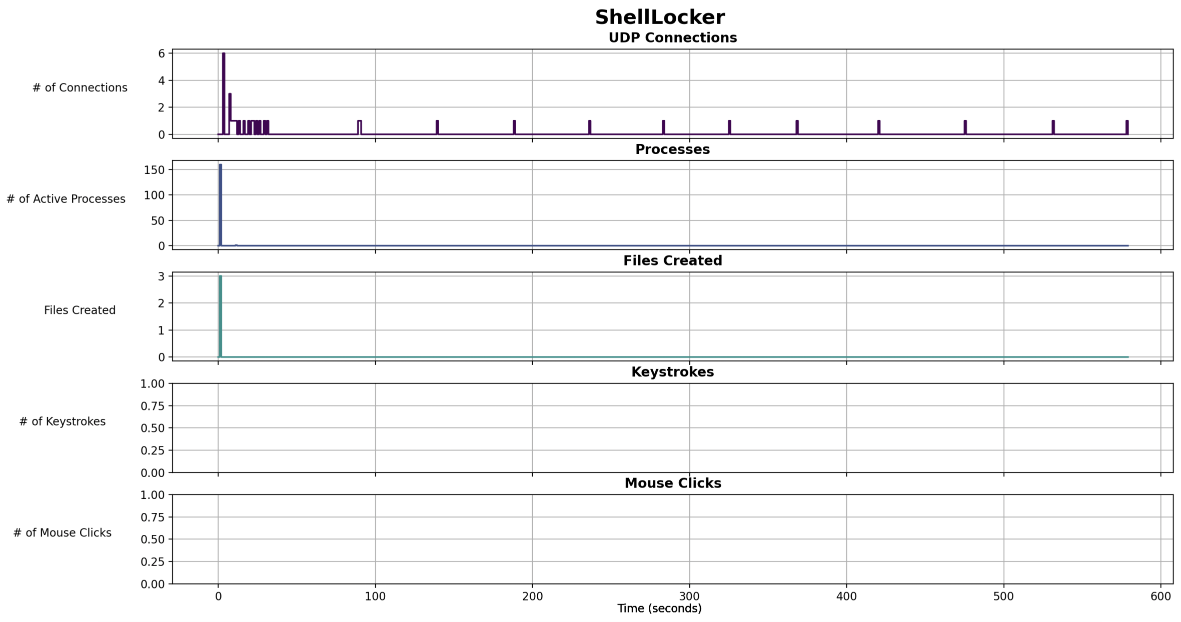

Figure A.12: ShellLocker

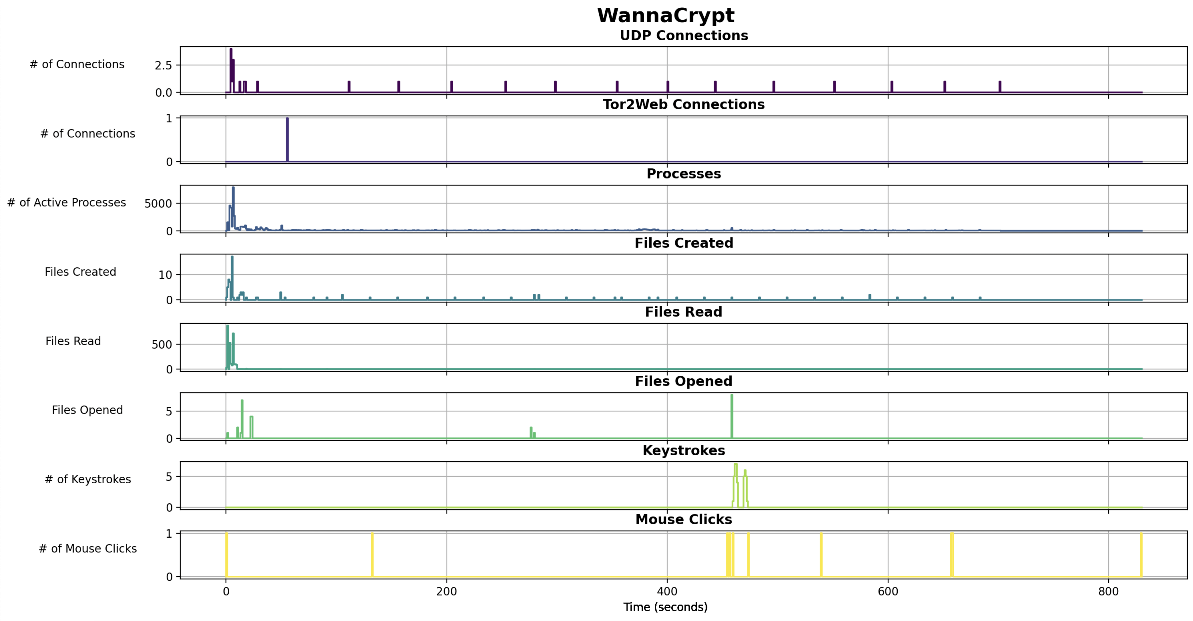

Figure A.13: WannaCrypt