Investigating anti-evasion malware triggers using automated sandbox reconfiguration techniques

Alan Mills [1] and Phil Legg [2],* iD

[1]   Computer Science Research Centre, University of the West of England, UK.; Alan2.Mills@live.uwe.ac.uk
[2]   Computer Science Research Centre, University of the West of England, UK.; Phil.Legg@uwe.ac.uk

Abstract: Malware analysis is fundamental for defending against prevalent cyber security threats, and requires a means to deploy and study behavioural software traits as more sophisticated malware is developed. Traditionally, virtual machines are used to provide an environment that is isolated from production systems so as to not cause any adverse impact on existing infrastructure. Malware developers are fully aware of this and so will often develop evasion techniques to avoid detection within sandbox environments. In this paper, we conduct an investigation into anti-evasion malware triggers for uncovering malware behaviours that may act benign when they detect a traditional sandbox environment. To facilitate our investigation, we developed a dynamic sandbox reconfiguration tool called MORRIGU that couples together both automated and human-driven analysis for anti-evasion configuration testing, along with a visual analytics view for examining system behaviours and performing comparative analysis. Our study reveals a variety of anti-evasion traits that are shared amongst different malware families, such as sandbox 'wear-and-tear', and Reverse Turing Tests (RTT), as well as more sophisticated malware samples that require multiple anti-evasion checks to be deployed. Using a systematic testing approach such as MORRIGU enables test coverage of anti-evasion methods, whilst also offering flexibility for further human-driven analysis of additional evasion methods. We also perform a comparative study against automated analysis using Cuckoo sandbox to show that automated scoring alone can not reliably inform on the presence of evasive malware, hence requiring a more sophisticated anti-evasive testing approach. With a greater understanding of anti-evasion malware triggers and with appropriate tools to explore these in an effective and efficient manner, this study helps to advance research on how evasive malware is being utilised to evade analysis so that we can better defend against future attacks.

Keywords: Malware analysis; Context-aware malware; Anti-evasion malware detection

0. Introduction

Malware analysis aims to study the traits and characteristics of malicious software, so that those tasked with defending computer systems can better understand the nature of an intended malicious attack and defend against future attacks. Common techniques include static analysis that examines software source code and dynamic analysis that observes system behaviours when the malware executes [1,2]. Moser et al. discussed the limits to adopting static analysis methods [3] such as code obfuscation techniques and it has long been recognised that dynamic analysis allows for greater understanding of how malware behaves in a given environment. Virtual Machines (VMs) are widely used as sandbox environments for dynamic malware analysis [2], so that activity can be safely contained in a virtual environment during malware execution so as to not cause damage to genuine production systems or users. All the while, malware development has become more sophisticated with greater measures being taken by malware developers to check the authenticity of the operating environment before executing an attack, often described as *context-aware* malware or *sandbox evasion*. This serves to evade sandbox environments that may be deployed purely for malware analysis. It also serves to ensure that target machines are worthy of compromise - such as machines that hold valuable data assets or that control valuable software services. Therefore, a fundamental challenge for analysts

is to curate testing environments that are sufficiently realistic such that the malware will believe it to be genuine and will not try to evade attack execution. Bulazel et al. survey evasion techniques deployed by malware [4], which can range from analysing system properties, checking for the presence of particular software being installed, through to observing and monitoring how the user interacts with the system to determine if there is a genuine user of the system. Being able to rapidly deploy and configure suitable virtualised environments is a time-consuming and repetitive process, such as installing additional software, incorporating user activity, and testing different permutations of these evasion techniques to observe whether the malware behaviours are triggered.

There are a variety of existing open-source and commercial malware analysis platforms available, including Cuckoo Sandbox [5], Joe Sandbox [6] and HookMe [7] that have been used widely as part of the pipeline for malware research e.g., [8–11]. However, these tools often require extensive configuration and setup to be able to utilise them, contributing towards the time overheads of deploying different testing permutations of a virtualised environment. Often, such tools are incorporated as part of an analysis pipeline, where the output results are used to classify or cluster behaviours using machine learning techniques [12]. Mills et al. [13] demonstrated an approach for interpretable machine learning to examine the relationship between features when classifying malware characteristics. Chumachenko [14] reports using Cuckoo Sandbox as part of an analytics process, resulting in 70,518 possible features that then had to be reduced down to 306 features through multiple feature selections methods. Again, this contributes towards the time and effort overheads required, whereas better integration of feature engineering can help reduce unnecessary computational and human effort required, which becomes significant when considering multiple permutations of the environment configuration.

In this paper, we study the properties of evasive malware and we identify triggers that cause evasive malware to execute within a virtualised testing environment. Existing tools such as Cuckoo have limited capability for anti-evasion, such as simulating random mouse movements to convince the malware that the system is being utilised by a human user. Malware development is becoming much more sophisticated for circumventing trivial randomised checks and so a more systematic and structured approach is required. Instead, we propose a flexible malware analytics system specifically for anti-evasion testing called MORRIGU[1]. MORRIGU is capable of automated rapid reconfiguration and deployment of virtualised environments for systematic testing of anti-evasion methods to identify malware characteristics. Compared to other analysis tools, MORRIGU is designed to minimise complicated configuration and provides full customisation and interaction from the end-user including profiling visualisations for understanding malware characteristics, as well as allowing user configuration for automated analysis. Using MORRIGU, we conduct a investigatory study on 251 malware samples, across 10 different types of malware and 49 malware families, to identify their evasive properties and their anti-evasion triggers under different environment conditions as curated by the testing system. We show the variety of different triggers for different malware variants, including how malware from the same family may respond differently under different configurations (e.g, Tinba). Given the natural arms race between malware development and malware detection, malware development will continually seek new evasive methods. Nevertheless, having a systematic testing framework that can help distinguish and identify such characteristics will significantly reduce the time and effort requirements of an analyst. In addition, having an intuitive visual analytics tools to investigate testing environment parameters in real-time can help to better understand the process execution and the resulting impact on the system. Our paper contributes towards research in this field, to provide an open-source system for configuring environments, and by conducting a research investigation on the effects of various malware samples under different virtualised environments, to

---

[1] Named after the celtic war goddess, also known as the "Phantom Queen" and a shape-shifter, a reference to the system's ability to become adaptable and undetectable to malware samples.

demonstrate how the systematic approach can identify characteristics of interest, whilst maintaining efficient runtime in an automated manner.

The main contributions of this paper are:

1. We propose a dynamic sandbox reconfiguration tool called MORRIGU that couples together both automated and human-driven analysis for configurable anti-evasion testing that is available for the wider research community.
2. Using MORRIGU, we conduct a systematic investigation into the nature of malware evasion techniques and show the identification of single and multiple anti-evasion methods utilised by evasive malware samples.
3. We show how different malware variants respond to different sandbox reconfigurations, including within the same malware families, and we also show how automated analysis methods alone fail to identify malicious signatures of evasive malware samples.
4. We provide a curated evasive malware dataset as a result of our findings that is made available to the community to support further research in this area (available from http://www.plegg.me.uk).

1. Related Works

We position our work within the context of existing research in the areas of malware analysis platforms, evasion and anti-evasion methods, and multi-system and transparent analysis techniques.

1.1. Malware Analysis Platforms

There are a number of open-source and commercial malware analysis platforms available, including Cuckoo Sandbox [5], PyREBox [15], Joe Sandbox [6] and HookMe [7]. Such tools have been used in broader malware analysis for academic research, however they are not without their limitations, including complicated configuration requirements [5], lack of GUI [15], and subscription costs [6]. Rhode et al. [16] used Cuckoo Sandbox to gather features for a recurrent neural network model for malware detection. Such malware analysis platforms can be very effective for extracting useful properties about malware process execution, and related system processes, however there is often substantial pre-processing of data required before the outputs from the analysis platform are suitable for machine learning systems, as described by Chumachenko [14]. In their work, a total of 70,518 possible features were extracted, that were then reduced down to 306 features through a process of feature selection methods, resulting in additional time and effort required for the analyst. This suggests that there is scope for further tools to reduce the time and effort requirements of feature engineering for malware analysis. Wagner et al. [17] also describe the challenge of reproducability between malware analysis platforms, since reporting may varying in terms of the inputs, analysis, output formats or options used. This further supports the argument that there is a need for better tooling that can offer greater consistency and sharing for analysts, reducing time and effort requirements.

A key requirement of a malware analysis platform is the ability to hide from the testing environment, and to be able to analyse malware samples that may be *context aware*. Context-aware malware will check the environment that it is situated within, and will only execute if a specific set of conditions are satisfied [4]. There are multiple analysis "fingerprints" that malware may use to determine whether it is executing inside a test environment, such as a virtual machine. These can include environmental artefacts, process introspection artefacts and reverse Turing tests amongst others [4]. This level of sophistication has become more common over the years as malware developers have continued the arms race against malware analysts to evade detection. As stated in [4] "differences unrelated to the evasion techniques being tested should be minimised (e.g., file system contents, time, settings, hardware resources, etc., should all be the same) so that spurious differences in execution are not conflated with evasion". Quite clearly, it is crucial that the analysis process needs to be able to assess different environment configurations in a efficient and timely manner to satisfy that malware has not evaded detection through any context-aware checks. Our work addresses this need so that

virtual environments can be quickly reconfigured, or tested under different anti-evasion techniques, to determine the malware triggers effectively.

1.2. Evasion and Anti-Evasion Methods

Context-aware malware will use numerous evasion techniques to avoid detection, with a technical report by LastLine reporting over 500 evasion techniques [18]. For example, a malware sample may wait for a specific time before executing, or for a specific number of keyboard or mouse events to have occurred. Whilst seemingly trivial, these evasion techniques are increasingly used to determine whether the environment is a genuine target (i.e., a real system with a real user). Common evasion techniques may include system artefact checks, including the use of registry edits and virtual environment processes ([2,4,19–26]), trigger-based behaviour ([4,19,20,26–28]) and checks for human interaction ([4,19,20,26,29–31]).

Anti-evasion methods are how the test environment can force, break or otherwise counter the evasion method. For example, the test environment may alter the system time to defeat a wait period, may install particular software to create a realistic system environment, or may simulate keyboard and mouse events to impersonate end user interaction. Our proposed system allows the user to examine the characteristics of malware when deploying different anti-evasion methods.

In evasion detection, a system will attempt to identify the use of evasive malware behaviours in the program execution. This can be accomplished through the use of debugging tools or other execution tracers, such as DSD-Tracer [21], or multi-system analysis. Debugging tools (or execution tracers) monitor malware behaviour and look for suspicious instructions or known detection techniques, such as checks against system artefacts. However such methods can be targeted using tools such as RDG Tejon Crypter, which offers analysis environment detection, with specific options for anti-debugging [32], through the use of opcodes such as *PUSHF* [2] or API calls [25]. Chen et al. [33] found that 40% of malware samples tested exhibited less malicious behaviour within a debugging environment.

Brumley et al. [27] investigate automated malware trigger analysis that aims to identify all possible trigger conditions so that an input can be given that will cause malware to execute as intended. However, one limitaion to this approach is that it is unfeasible to follow all possible execution paths and so potentially some triggers are likely be missed. Similarly, Pektaş and Acarman [24] use a pin tool that attaches to a process to intercept system calls to mislead malware, to check whether the malware is executed within a virtualised environment. For example, if the response to a check contains the string "VMware" then a replacement can be substituted instead (i.e. "Microsoft"). In this approach, it may be possible for the malware to identify the pin tool and therefore trigger a context-aware malware sample to act in a benign fashion [32]. Lindorfer et al. [34] adopt a multi-system approach for analysing malware with evasion detection by using 4 sandbox environments, each with a different system setup (3 Windows XP virtual machines and 1 Anubis sandbox). Behavioural comparisons are made across all 4 environments when executing the malware sample to identifying any significantly different behavioural patterns may made indicate particular configuration triggers. However, this is computational expensive and is still susceptible to samples that can evade analysis under all environments, for example using network environment checks.

2. Malware Dataset Curation

To inform this research, we gathered malware samples from the VirusShare service [35], an open source malware repository linked to VirusTotal reporting. Malware samples were identified using a combination of open source reporting, including the Centre for Internet Security (CIS) [36]. We collated samples from the top 10 malware threats as stated by the CIS for the 6-month period of August 2019 to January 2020. Adopting this approach meant that we could obtain high diversity of samples whilst ensuring that the samples were also relevant to the current challenges being observed in real systems today rather than working with earlier malware datasets that may be considered outdated. Interesting observations for our data collection revealed that 5 malware families were recorded in the top 10 every
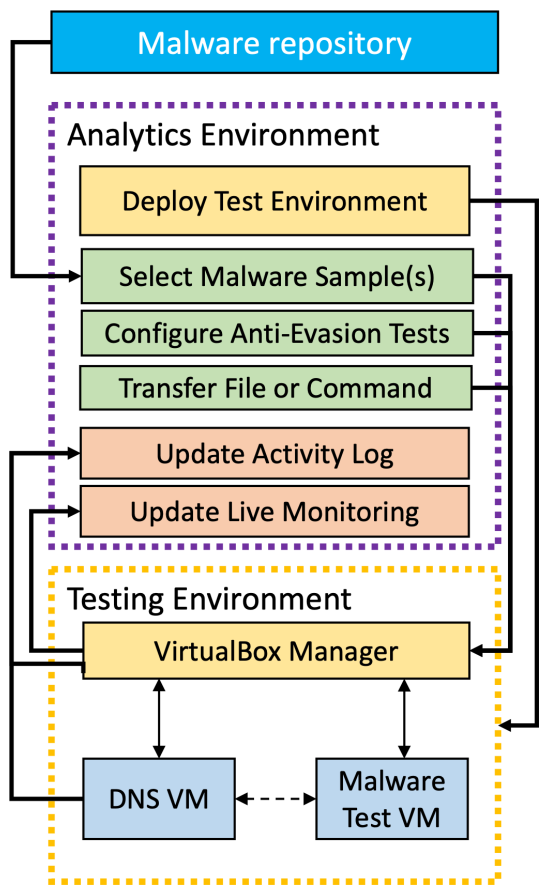
Figure 1. System Architecture to show the interactions between the analytics and testing environments

month during this period (Dridex, Gh0st, Kovter, Nanocore and Zeus). The reports also highlight the use of multiple malware variants within the same classification, for example, the top 10 malware samples in January 2020 consisted of 2 Banking Trojans and 3 Remote Access Trojans (RATs).

The resulting dataset consists of 251 malware samples across 49 families (and 10 malware classifications). The 10 malware classifications used in this research are: Adware, Banking Trojan, Bot Net, Cryptocurrency Miner, Disk Wiper, Domain Generating Algorithm (DGA), Ransomware, Remote Access Trojan (RAT), Trojan Downloader, and Trojan Infostealer. All malware samples are Windows portable executable files. As is customary, samples are downloaded from the VirusShare service without file extension and are contained within a password-protected zip file. For ease of use, MORRIGU is integrated to work with VirusShare such that samples can be downloaded and extracted automatically from their data storage, so that malware is not inadvertently accessed. To encourage further reproducible studies in the research community, the full listing of the malware dataset is available from the authors on request.

3. Analytics Environment and Methodology

Here we describe the MORRIGU analytics environment and how this can hook into a virtualised test machine for deploying malware samples. We also provide our methodology for using MORRIGU to assess the variants of context-aware malware triggers adopted by different malware samples.

3.1. Analytics Environment

The analytics environment consists of a Python-based client/server application that incorporates a Django server and a web-based GUI. The server is able to communicate to and from the virtualised environment using VBoxManager (for VirtualBox) or VMrun (for VMware). For the purpose of the
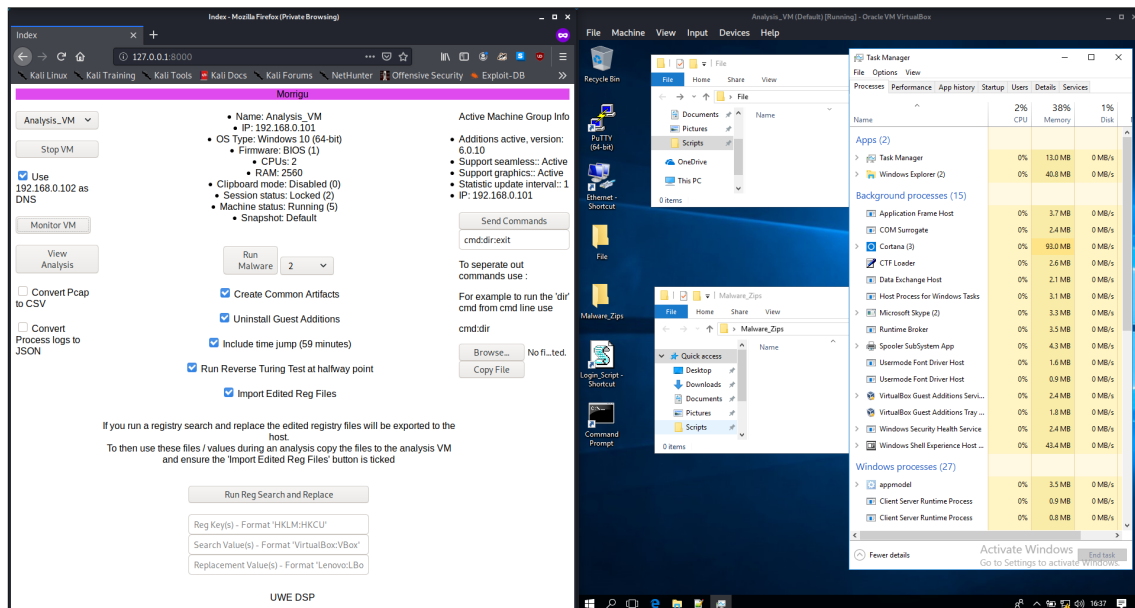
Figure 2. Example to show the MORRIGU analytics environment and configuration options (left), and the associated malware testing environment using a Windows 10 virtual machine (right).

paper, we will describe the usage of VBoxManager. Dynamic configuration of the test environment is achieved using Powershell and VBS scripting, where the scripting parameters can be easily tuned by the user using the main analytics GUI, and then deployed to the testing environment using the VBoxManager. Figure 1 shows the system architecture in further detail. In particular, the VBoxManager provides access to the virtualised testing environment, which is the machine that will be used for deploying the malware, whilst our analytics environment is responsible for accessing the protected malware repository, transferring the selected sample into the testing environment, configuring the set of anti-evasion techniques that should be tested against as determined by the user (either automatically using a batch process, or manually on an single-instance basis), capturing the resulting data attributes of the malware execution, and visualising the output.

The MORRIGU GUI (Figure 2) allows the user to configure and control the complete testing process, from creating and initialising the virtualised testing environment, through to configuration and deployment of anti-evasion methods, transfer of malware from the repository to the testing environment, and the analysis and visualisation of data attributes post-execution of the malware. Additional functionality includes uploading files to the testing environment, and sending direct commands for execution (through the Windows Command Line). Multiple malware samples can be selected from the repository for batch processing, which are then executed in sequential order based on the user configuration.

For the purpose of the study, we use Windows 10 as the virtualised testing environment that malware will be deployed within. We use an Alpine Linux VM[2] to serve as a DNS server. The majority of communication between the host machine and virtualised testing environment is managed through the use of VirtualBox sessions as they can be dynamically setup and torn down, helping to limit the risk of cross contamination caused by malware activity trying to escape from the sandboxed virtualised environment whilst also maintaining access control with our analytics environment. Some anti-evasion configurations will effect the ability to transfer files between the host machine and test environment through VirtualBox sessions (i.e., the removal of Guest Additions). To overcome this, we also support file transfer between the test environment and the host machine via the DNS server. This design choice

---

2   https://www.alpinelinux.org/

was intentional to avoid direct communication between the host machine and the virtualised testing environment in cases where safe, direct communication was not possible.

The baseline for the Windows 10 testing environment was set up for a "best use" case, with no consideration regarding context-aware malware or virtual transparency. VirtualBox Guest Additions were installed and running as default, along with associated programs and services, to represent typical usage of a Windows 10 environment. All malware samples were tested against this baseline using MORRIGU (which we refer to as 'no change' to the testing environment). The analytics environment allows for the following anti-evasion methods to be applied dynamically to the testing environment through the execution of customised PowerShell scripts:

- **Guest Additions Uninstalled (GAU):** This script will perform uninstallation of the Guest Additions software on the virtualised environment. It will also check the Windows Registry for the phrase 'Guest Additions' to ensure that all traces are removed. Due to practicalities with the testing setup, it was more efficient to remove the installed tools rather than install when required.
- **Trigger Based Activation / Time Jump (TJ):** This script will perform a 'time jump' by increasing the system clock by 59 minutes (as a default value). 59 minutes was chosen as the default so that both hour and minute would be updated.
- **Common Artefact Creation (CAC):** This script will create new files within the user directories (Documents, Downloads, Favourites, Desktop and Pictures) to simulate user activity. For the Documents, Downloads and Desktop folders, the script creates files in these directories based on 4 commonly-used file types: TXT, PDF, ZIP and CSV. For the pictures folder, the contents of pre-existing Windows Themes Cached Files location are copied across and the files are re-named simply to a random number with a .jpg extension. For the Favourites folder, a URI is currently created to a single website (which could be extended to a set of websites as defined by the user).
- **Registry Edit (RegEdit):** This script will search name-value pairs in the Windows Registry, and will perform find and replace on any given phrases. For our study, we replace "VirtualBox", "VBox" and "VMWare", with "Lenovo", however a user could provide any text entry.
- **Reverse Turing Test (RTT):** This script will imitate end user interaction, which is well-recognised as a key malware evasion technique [4,20,26,29]. This is in part due to the ability to measure the reaction and input speeds, allowing malware to remain inactive or benign when the speeds are believed to be too fast, indicating a machine user rather than human user [4,29]. As default the script will move the mouse around the screen either using a randomised array or preset positions. Unlike Cuckoo, the script will interpolate between points to provide realistic movement. The script will also vary the mouse speed whilst moving, and will randomise a sleep period at each point between 0.6 to 1.5 seconds, so as to provide some variation in motion to avoid a consistent appearance. Using the MORRIGU configuration, automated keyboard entry and mouse movement patterns can be further customised. Furthermore, the script will also open various applications on the system, and receive inputs from the keyboard (e.g., typing in notepad), so as to simulate further user interactions. As with mouse movements, keyboard entry also utilises randomised sleep patterns.

As these dynamic modifications are independent, there are 32 possible configurations that can be tested by our system using default configurations (31 permutations ranging from 1 to 5 of the possible methods being set as true, plus the state of 'no change'). Each method can either be deployed using default or user-defined parameters, and so naturally there exist many more possible combinations of how a user may test different anti-evasion methods. As an example, for testing against Registry Edits, a user may want to check how the testing environment performs when only the value "VirtualBox" is replaced with "Microsoft", or when the values "VirtualBox" and "VBox" are both replaced. Clearly testing for all possible text replacement is unfeasible and is considered out of scope for this paper, although our approach would make it easier to explore such modifications. We describe each method in greater detail within the relevant sections that follow.
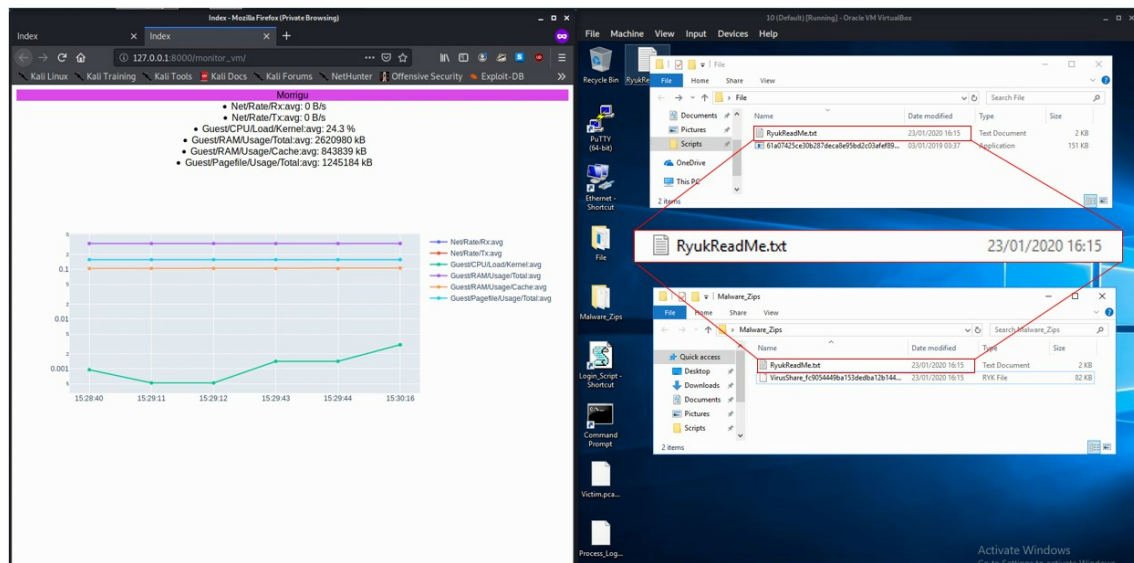
Figure 3. Example to show live monitoring in the MORRIGU GUI (left) when the Ryuk Ransomware has been deployed using MORRIGU within the Windows 10 testing environment (right).

The GUI also provides visual analytic tools for live monitoring and analysis of the testing environment. This can be used to identify malware activity on a real-time basis, by profiling changes across system attributes. Users can also explore historical records post-analysis, allowing for comparison between different testing configurations. The following time-series data attributes can be visualised in the analysis tool as either raw data or as averaged values (based on a user-defined sliding time window): Pagefile Usage (Total), RAM Usage (Total), CPU Kernel Load, RAM Usage (Cache), Network Receiving Rate, and Network Transferring Rate. Figure 3 shows an example of analysing a Ryuk Ransomware attack. It can be seen that there is an increased CPU load that corresponds with execution of the malware, where the ransom encrypts files within the testing environment and creates ReadMe files to notify the user of the attack[3]. Ransomware by it's very nature will make itself known to the target (since a payment is often demanded to retrieve a decryption key), however other forms of malware such as Infostealers, Downloaders, and Remote Access Trojans may intentionally be less obvious to the user, however, their behaviours on the system can be identified using the live monitoring tools.

MORRIGU also supports visual analytics for post-analysis to provide details on the network and process activity that occurred during the analysis period. These are key in identifying both the activity levels of a sample and the type of activity, and for comparative analysis against the baseline execution, different anti-evasion configurations, and between different malware variants. Plots can be configured to show all activity or a subset of the data. This could be extended to show only relative difference in activity between a pre-existing execution of the testing environment (e.g., a common case would be to show the differences against the baseline execution of the malware when 'no changes' have been made to the testing environment). Figures 4 and 5 show a comparative analysis of the network and process activity for a Bayrob Infostealer malware sample when there is 'no change' to the testing environment (Figure 4), versus when the VirtualBox guest additions are uninstalled (Figure 5). With the guest additions uninstalled, it can be seen that there is a significant variation in network activity when performing the same execution task, including new IP addresses and a pattern of increased frequency

---

[3]   Note that the time difference between the testing environment and analytics environment is due to the independent time management of the testing environment, required for some of the anti-evasion configurations such as Time Jump.
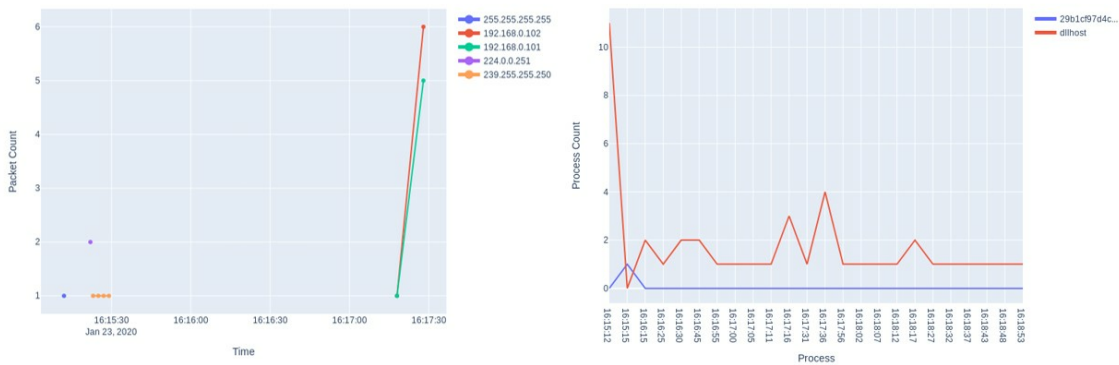
Figure 4. Network and process activity for Bayrob - No Change Environment
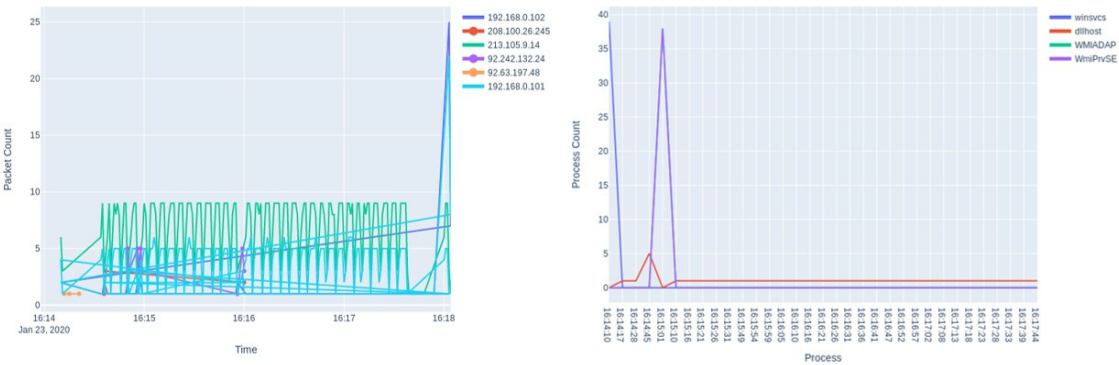
Figure 5. Network and process activity for Bayrob - Guest Additions Uninstalled

for communicating with these IPs. New processes are also observed, in particular, there is a significant increase for the WmiPRvSE process. The use of WmiPRvSE and other WMI (Windows Management Instrumentation) associated services is a common methodology for malware development, often considered to be "living off the land" by exploiting existing Windows services. By using these services, malware can gather information about the system, execute scripts and control other processes through what appears to be a legitimate means, and in a way that can potentially avoid detection [37]. The use of simple and intuitive visual analytics for malware investigation can provide quick summarisation of complex and detailed activity logs. Since the visual analytics are interactive, the user can hide or highlight different attributes. The use of line and scatter plots are highly configurable to yield further investigation, such as manipulating axes to show IPs by conversation, or to view process details by specific attributes such as HandleCount[4] or Processor Time. As an example, an analyst could quickly identify malware that uses multiple hardcoded IP addresses such as Kovter and WannaCry, and filter their investigation to identify more complex malware triggers and behaviours. Data from the visual analytics display can also be exported in multiple formats. Traffic captures can be in PCAPNG or CSV format and process logs can be in JSON or CSV. This would allow the system to be integrated as part of a broader machine learning pipeline, and would help to alleviate many of the issues related to feature engineering and data pre-processing for cleaning inputs for performing classification or regression tasks.

3.2. Testing Methodology

Malware samples were accessed by MORRIGU from our malware repository using batch processing, to be able to reconfigure the testing environment, deploy the malware sample, and capture

---

4 A data attribute that records how many objects or files a process has interacted with.

the output results, for each of the malware samples being testing and for each of the configuration permutations as described previously. Each test was classified automatically into the following 3 groups:

- No Activity that deviates from the observed baseline system behaviours.
- Activity that results in new communication being made to a single IP address.
- Activity that results in new communication being made to multiple IP addresses.

The presence of new single and multiple IP activity compared against the observed baseline is a strong indicator of malware-related activity. Expected network activity or IP addresses which are commonly seen, such as communication between the Windows 10 testing environment and the official Microsoft IP addresses, the ISP provider[5] or generic Content Delivery Networks are excluded using rule-based filtering.

The output from the testing environment includes the network traffic (as capture from both the testing environment and the DNS VM), and the process logs for the malware test VM, which are then used to update the visual analytics monitoring view. Results are also saved directly for future comparative analysis.

The system will test all 32 permutations of anti-evasion methods (including the baseline 'no changes' stage as the initial testing stage), from 1 through to 5 methods being utilised. After each test, samples that revealed significant activity (such as communicating with multiple IP addresses or malicious actions on the analysis VM itself such as the encryption or destruction of data), were then deemed as successfully executed and so were removed from further evasion testing. If the malware still did not deviate from the baseline having tested all 32 permutations, and did not exhibit malicious behaviour, it is deemed to have evaded the detection system. Where a malware sample yields limited activity (e.g., attempting to access a single IP address) we would continue to test against further permutations, since the sample may be conducting an external check which could easily be circumvented under different conditions.

Each malware sample was tested in isolation to avoid interference from other samples and the malware test VM was automatically reset to a clean state between tests, as managed by the MORRIGU environment. Malware samples were retrieved from the repository, and transferred to the testing environment through the use of VirtualBox sessions. Sessions were then ended once the file transfer was complete to reduce the risk of cross contamination.

4. Results

For our study, a total of 251 malware samples were tested using the MORRIGU environment, consisting of 10 different malware classifications and 49 different malware families. As discussed previously, samples were drawn from recent CIS reports, to ensure that we are considering current threats that are being reported in real-world operations. Within our testing environment, 173 of the 251 samples (68%) were deemed as 'active', i.e., they would trigger some malicious response within the testing environment such as connecting to an IP address or causing significant activity within the VM such as encryption or destruction of data. Samples were only confirmed as being 'active' if the behaviour was observed at least 3 times under repeatable testing conditions. Of the active samples, 133 samples were found to perform malicious behaviours when used in a typical virtualised environment (i.e., where 'no changes' are made to the testing environment). These samples therefore do not exhibit any evasive capabilities. There may well be good reason for this - for example, malware that is designed to corrupt any data may not necessarily be concerned about being evasive and may wish to simply execute whenever possible, rather than being targeted to specific machine configurations. We

---

[5]   With the exception of addresses within the 81.96.0.0/12 and 213.105.0.0/17 ranges, as these are registered to the ISP under IP blocking addresses for known malicious IPs

Table 1. Results from 40 evasive malware samples to show the minimum anti-evasion configuration required to trigger a malicious response.

Malware Sample	Configuration Options				
	Guest Additions Uninstalled	Time Jump	Reverse Turing Test	Registry Edit	Common Artefacts Creation
Adposhel_60de (Adware)				✓	✓
Andromeda_240a (Downloader)	✓				
Andromeda_928a (Downloader)	✓				
Andromeda_ea1e (Downloader)	✓				
Azorult_c4e2 (Infostealer)	✓				
Bayrob_8972 (Infostealer)	✓				
Bayrob_ab9b (Infostealer)	✓				
Bayrob_b12e (Infostealer)	✓				
Bayrob_b6c7 (Infostealer)	✓				
BitCoinMiner_8554 (Cryptocurrency Miner)	✓				
Blocker_2c1c (Ransomware)	✓				
Blocker_7fe1 (Ransomware)			✓		
Dridex_58fb (Banking Trojan)			✓		
Dridex_d684 (Banking Trojan)			✓		
Dridex_f236 (Banking Trojan)			✓		
Gamarue_f7d2 (Downloader)	✓				
Gozi_3130 (DGA) *				✓	
Gozi_8e8b (DGA)			✓		
IcedID_04fb (Banking Trojan)			✓	✓	
Lokibot_95df (Infostealer)		✓	✓		
Neutrino_4929 (Bot Net) *				✓	
Neutrino_4975 (Bot Net)			✓		
Nymaim_c003 (Ransomware / Downloader)		✓			
Qakbot_2f63 (Infostealer)		✓			
Qakbot_8279 (Infostealer)		✓			
QuantLoader_acac (Downloader)			✓		
Panda_244f (Banking Trojan)			✓		
Panda_b455 (Banking Trojan)			✓		
Panda_ca8b (Banking Trojan)			✓		
Pushdo_cd63 (Bot Net)	✓		✓		
Strictor_5e9c (Downloader) *				✓	
Tinba_3132 (Banking Trojan)				✓	
Tinba_9120 (Banking Trojan)			✓		
Tinba_b377 (Banking Trojan)				✓	
Tinba_ba58 (Banking Trojan)					✓
Ursnif_0e82 (Infostealer)			✓		
Ursnif_393a (Infostealer)			✓		
Ursnif_ad1d (Infostealer)			✓		
Ursnif_ee9f (Infostealer)			✓		
WannaCry_410f (Ransomware)				✓	

found that all 10 malware classifications (Adware, Banking Trojan, Bot Net, Cryptocurrency Miner, Disk Wiper, Domain Generating Algorithm (DGA), Ransomware, Remote Access Trojan (RAT), Trojan Downloader, and Trojan Infostealer) included samples that were non-evasive.

As a result of this initial testing phase, 40 samples were identified using MORRIGU that exhibit no malicious behaviour when used in a typical virtualised environment (i.e., where 'no changes' are made), however they do perform malicious behaviours when anti-evasion configuration changes are made. Whilst this represents only 16% of our original malware sample dataset, it provides an initial step towards curating evasive malware datasets for further research. To the best of our knowledge,

Table 2. Results from 40 evasive malware samples to show the anti-evasion configurations required to trigger a malicious response against each of the 8 malware classification types that adopt evasion techniques.

Malware Classification	Configuration Options					
	Guest Additions Uninstalled	Time Jump	Reverse Turing Test	Registry Edit	Common Artefacts Creation	Sample Count
Adware				1	1	1
Banking Trojan			8	4	1	11
Bot Net	1		2	1		3
Cryptocurrency Miner	1					1
DGA			1	1		2
Ransomware	1	1	1			4
Trojan Downloader	4		1	1		6
Trojan Infostealer	5	3	5			12
Total	**12**	**4**	**18**	**8**	**2**	**40**

there are currently no publicly available dataset for evasive malware, and so we see this as a valuable contribution to the research community.

4.1. Anti-Evasion Results based on Individual Malware Samples

Table 1 shows each of the 40 evasive malware samples, highlighting the anti-evasive methods that caused the malware to be triggered. Where different samples from the same malware family respond under differing conditions (i.e. different samples of the Tiny Banker, aka Tinba malware family), we append the last 4 digits of the MD5 hash to identify the sample. 8 of the 10 malware classifications tested are listed in the evasive malware results. The remaining two classification (Disk Wiper and RAT) were deemed as 'active' when 'no changes' to the environment were deployed. Given the nature of a disk wiper attack, the attack is essentially to cause disruption rather than to evade detection, and so it stands to reason that this would execute as intended under any environment configuration. Looking at the results, 12 of the samples triggered with GAU, 4 triggered with TJ, 18 triggered with RTT, 8 triggered with RegEdit, and 2 triggered with CAC. Whilst samples may be triggered with further anti-evasive methods used, we show here the minimal required set of methods. Furthermore, there are 4 samples (Pushdo, Lokibot, IcedID, and Adposhel) that required at least two evasive methods to be deployed in order for the malware to be triggered. This illustrates the level of sophistication increasing, since no response was observed when only a single method was utilised. Further research will seek to identify samples that require a combination of multiple anti-evasive checks. Three samples highlighted by the asterisk (Neutrino_4929, Gozi_3130 and Strictor_5e9c) also triggered when GAU as well as RegEdit. However, given this finding, we can identify this as being RegEdit since the GAU also performs a RegEdit as part of its process. Further findings reveal that different malware samples that are part of the same malware family may respond under different anti-evasion methods. For example, the Tinba malware family shows that different samples of the same malware family are conducting different evasion checks. All of the Tinba samples tested were attempting to contact multiple FQDNs (Fully Qualified Domain Names), with each matching the same pattern of 12 seemingly-random alphabetic characters with one of three domain names: 216.218.185.162 (registered to Hurricane Electric), 92.242.132.24 (error handling for Barefruit), and 81.99.162.48 (Virgin Media IP Blocking). However, it should be noted that two of these cases were the result of malicious traffic being redirected or blocked. However, this illustrates the evolution of malware capabilities to naturally extend upon existing examples and require more sophisticated evasion checks.

4.2. Anti-Evasion Results based on Malware Families

Table 2 shows the 8 malware classifications that we found respond to anti-evasion checks using the MORRIGU environment. Since some samples required multiple anti-evasion methods, the number of samples may be lower than the sum of anti-evasion methods listed (e.g., Adposhel, an Adware malware, required both RegEdit and CAC). The results show that the majority of samples responded to the RTT, although a number of these were part of the Banking Trojan family, revealing that this is a common anti-evasion method amongst this malware family. Less cases were observed using TJ and CAC. This could be due to the approach of how common artefacts are created by our system, where they could be deemed as not truly representative of how a genuine user would react. Clearly, more sophisticated methods of modelling user creation of files could be developed. However, it could also be that the malware samples available simply do not require this evasion method, and so it is challenging to account for this since no ground truth data is available on the malware samples utilised. Nevertheless, our system offers the capability for testing this domain should more malware samples adopt CAC in the future. This provides further motivation for the curation of evasive malware datasets, of which the MORRIGU environment would help researchers to gather.

Other findings from our study indicate that there was correlation between network activity and malware families. For example, the network behaviour of most ransomware samples during testing was readily identifiable, e.g., one sample of WannaCry (Ransomware) attempted to connect to over 10,000 IPs within 10 minutes. However, this was not the case for all samples which used hardcoded IPs, as one sample of Dridex (Banking Trojan) connected to only 2 malicious IPs, spread out over 4 minutes, with the largest amount of data (13k bytes) being sent in less than 6 seconds. The type of malware and risk appetite, therefore, seems to be key factors in the network activity, specifically when using hardcoded IPs. A Banking Trojan that attempts to exfiltrate data without being noticed is clearly more concerned with remaining discrete in its use of hardcoded IPs, compared to a Ransomware attack that would be highly visible on a machine or network. The MORRIGU system enables researchers to understand such characteristics between malware families much clearer through the use of interactive visualisations, as was previously illustrated in Figures 4 and 5.

Our results have highlighted the importance of examining different anti-evasion techniques for triggering malware responses in a virtualised testing environment. In particular, whilst traditionally researchers may have had to adopt ad-hoc methods to tackle this problem, here we show how a systematic approach using the MORRIGU environment can help trial various permutations of configuration setups, offering an easier, and also more complete, analysis capability. Using MORRIGU, we have been able to identify samples that only respond when anti-evasion methods are deployed, and we have also been able to identify cases where malware families may vary in their evasion methods, revealing evolution of the malware capabilities, and also where multiple anti-evasion methods are required to trigger the malware sample. Both our evasive malware dataset, and our MORRIGU system, are made available as open-source to serve the research community.

4.3. Comparison between MORRIGU and Cuckoo

To further support the utilisation of MORRIGU, we perform a comparison against existing malware analysis platforms. Cuckoo Sandbox is widely-used by researchers for malware analysis, and allows for monitoring of virtualised environments using an agent plug-in. As a key comparison between the two environments, Cuckoo is recognised to require extensive configuration in order to make this usable for analysis, and so there is a steep learning curve required before analysts can utilised the environment. There is significant documentation available on Cuckoo, however to quote the documentation: *'Cuckoo is not meant to be a point-and-click tool: it's designed to be a highly customizable and configurable solution for somewhat experienced users and malware analysts.'*. To be able to fully utilise Cuckoo Sandbox an end-user needs to go through setup for both the host machine, the guest machine (which will be used to run the malware samples) and the virtual network setup that links the two. A further issue is that as of yet Cuckoo required Python 2.7 for the use of the agent that is installed on

the virtualised environment. Python 2.7 was due to be phased out in 2020. Whilst this will likely be updated in an upcoming release, it is another contribution towards Cuckoo not being easily accessible for beginners. By comparison MORRIGU is designed as a point and click tool. Once the server has been started the relevant network setup is carried out by the system itself and all configuration options can be managed through the GUI. Cuckoo is highly customisable however it often requires command line interface (CLI) and manually configuration through the use of configuration files. Therefore, MORRIGU does serve to offer a more user-friendly approach for malware analyst which could be utilised by those looking to enter the industry.

Regarding analysis, Cuckoo is designed to execute a malware sample against the environment for a given period of time, and then terminate this environment, which is also the same approach adopted by MORRIGU. However, MORRIGU will then perform multiple iterations of the same test with different configuration parameters, and the results are visualised using the visual analytics tools to observe whether there is deviation in behaviours. Cuckoo is designed to provide a score between 0 and 10 for each sample, based on a pre-defined signature rule-based system. Cuckoo documentation suggests that these can be treated as 3 classifications: low (0-4), medium (4-7), or high (7-10). However, as reported by Walker et al. [38] this scoring classification can be misleading due to the underlying methodology of severity classification. They suggest that greater emphasis is required on analyzing the behavior of the malware, which is the objective of MORRIGU. Using the generated visual analytics, system metrics can be visualised and contextualised to highlight key network and process activity in a manner that Cuckoo does not currently provide. Furthermore, Cuckoo currently only offers limited anti-evasion methods, such as randomised mouse movement, which could easily be identified by evasive malware. MORRIGU is specifically designed to focus on the analysis of anti-evasion methods, and provides a flexible manner for extending this through the use of PowerShell scripting given the continually evolving nature of malware analysis and malware development.

Table 3 shows the results obtained from the Cuckoo Sandbox malware analysis tool, when testing each of the 40 evasive malware samples as identified using MORRIGU. In particular, the purpose of this study is to assess the reliability of the scoring methods against our evasive samples since previous works have suggested that the Cuckoo scoring metrics can be problematic [38]. Whilst Cuckoo does provide detailed reports on malware execution, many analysts will rely on the scoring obtained to provide summarisation of a large malware dataset. We perform our experimentation using the same Windows 10 environment as deployed with MORRIGU, and we also report on testing with a Windows 7 environment. What we aim to highlight here is the challenge of automated analysis, and how a configurable environment like MORRIGU can be used in conjunction with Cuckoo to improve anti-evasion malware analysis. Firstly, we can see that the Windows 10 results are significantly lower than when using Windows 7. Importantly to note, all anti-virus and firewall measures are disabled on both operating systems. Also, all samples had successfully executed using MORRIGU. We will therefore focus more on the Windows 7 results for the majority of our study, recognising that many samples would have effectively evaded any form of detection using the Cuckoo scoring thresholds. Looking at the Windows 7 results, they show significant variation in scoring, from 0.0 (Bayrob_b6c7 InfoStealer) to 14.0 (Qakbot_2f63 InfoStealer). Scoring is based on a weighted function of the number of signatures identified and their severity. Whilst this is supposedly between 0 and 10, 2 cases generated scores above this. Using the Cuckoo scoring bands, the majority of samples (25 out of 40 on Windows 7, 35 out of 40 on Windows 10) would be considered as 'low' and could well go unnoticed in a large scale analysis exercise. Of course, we may expect evasive malware samples to evade simple forms of detection, however this highlights the significance of performing deeper anti-evasion checks. Whilst Cuckoo offers a lot of customisation, it is quite limited in terms of specific anti-malware evasion checks. The module "human" is used by default, which randomly simulates mouse movement and button clicks. In MORRIGU, our RTT offers much richer ability such as opening programs and simulating text entry. With the "human" module disabled, 11 scores deviated under Windows 7 (with only 3 in Windows 10) as shown in brackets. Surprisingly, some cases of Pushdo, Dridex and Wannacry actually

Table 3. Cuckoo Sandbox scoring results for 40 evasive malware samples, using the default cuckoo settings (Windows 10: Mean=1.2, Median=0.6; Windows 7: Mean=3.66, Median=2.2). Scores in brackets show where results differ when the 'human' module is disabled.

Malware Sample	Cuckoo Score (Win10)	Cuckoo Score (Win7)
Adposhel_60de (Adware)	0.2	3.4
Andromeda_240a (Downloader)	0.4	6.4
Andromeda_928a (Downloader)	0.0	9.8 (9.2)
Andromeda_ea1e (Downloader)	0.4	6.4
Azorult_c4e2 (Infostealer)	1.8	1.8
Bayrob_8972 (Infostealer)	0.6	0.6
Bayrob_ab9b (Infostealer)	0.6	0.8
Bayrob_b12e (Infostealer)	0.4	0.4
Bayrob_b6c7 (Infostealer)	0.4	0.0
BitCoinMiner_8554 (Cryptocurrency Miner)	0.6	0.6
Blocker_2c1c (Ransomware)	0.0	0.6
Blocker_7fe1 (Ransomware)	0.0	4.2 (1.0)
Dridex_58fb (Banking Trojan)	4.4	3.6
Dridex_d684 (Banking Trojan)	4.8 (5.2)	4.4
Dridex_f236 (Banking Trojan)	4.0 (4.4)	3.6
Gamarue_f7d2 (Downloader)	0.8	1.8
Gozi_3130 (DGA) *	0.2	1.0
Gozi_8e8b (DGA)	0.8	2.0 (1.2)
IcedID_04fb (Banking Trojan)	2.2	2.2
Lokibot_95df (Infostealer)	0.8	6.6
Neutrino_4929 (Bot Net) *	0.6	0.6
Neutrino_4975 (Bot Net)	0.6	0.6
Nymaim_c003 (Ransomware / Downloader)	1.2	0.8
Qakbot_2f63 (Infostealer)	0.8	14.0
Qakbot_8279 (Infostealer)	0.4	13.6 (12.6)
QuantLoader_acac (Downloader)	0.2	1.6
Panda_244f (Banking Trojan)	0.4	5.8
Panda_b455 (Banking Trojan)	6.0	5.4
Panda_ca8b (Banking Trojan)	6.4	5.8
Pushdo_cd63 (Bot Net)	0.8	2.2 (2.8)
Strictor_5e9c (Downloader) *	0.2	1.8 (1.2)
Tinba_3132 (Banking Trojan)	0.8	6.8
Tinba_9120 (Banking Trojan)	0.0	5.6
Tinba_b377 (Banking Trojan)	0.0	2.2
Tinba_ba58 (Banking Trojan)	1.0	7.0
Ursnif_0e82 (Infostealer)	0.6	1.8 (1.0)
Ursnif_393a (Infostealer)	2.8 (2.0)	2.8 (2.0)
Ursnif_ad1d (Infostealer)	0.6	1.8 (1.0)
Ursnif_ee9f (Infostealer)	0.6	1.8 (1.0)
WannaCry_410f (Ransomware)	0.4	4.2 (5.0)

scored higher when no human interaction was simulated. Figure 6 shows the scoring distribution for the 40 samples (Windows 10 (Orange): Mean=**1.2**, Median=**0.6**; Windows 7 (Blue): Mean=**3.66**, Median=**2.2**). Given that these samples have been identified as evasive in their nature, a suitable testing framework needs to be able to trial various anti-evasive configurations. The default practice of Cuckoo is limited to the "human" module and does not support other forms of anti-evasion variants that we tested using MORRIGU (e.g., System Time Jump, RegEdits, or Common Artefacts Creation). As mentioned, the default RTT in Cuckoo is relatively limited, and produces randomised mouse movement that could easily be identified. This provides further justification for developing more sophisticated anti-evasion techniques such as those in the MORRIGU toolkit, that could be utilised in conjunction with more advanced analysis platforms such as Cuckoo.
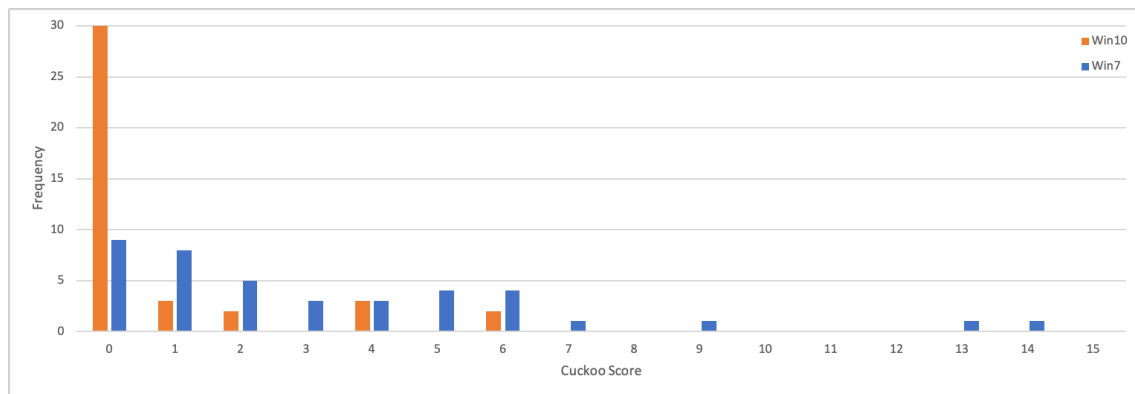
Figure 6. Distribution of Cuckoo scoring for the 40 evasive malware samples (Windows 10 (Orange): Mean=**1.2**, Median=**0.6**; Windows 7 (Blue): Mean=**3.66**, Median=**2.2**).

5. Discussion

We have presented a study of assessing anti-evasive malware triggers based on a systematic testing process using the MORRIGU environment. This provides a highly configurable virtualised environment with anti-evasion configurations for deploying and testing evasive malware samples. From a set of 251 malware samples, we show how the MORRIGU environment is able to identify samples based on evasive characteristics, and we found at least 40 of the current malware threats as described by CIS utlise malware evasion methods. Since 173 of the 251 were deemed to be 'active', it is feasible that the remaining samples may have responded to other anti-evasion methods that are outside of the scope of this current study. Nevertheless, the MORRIGU environment was capable of identifying 40 samples that would not have necessarily been identified within a 'traditional' testing environment, since no change in behaviour was observed in without anti-evasion methods being deployed. Given the inherent nature of malware development and malware detection, evasion techniques will undoubtedly be developed further as time goes on, and become more and more sophisticated in their nature. MORRIGU provides a flexible analysis environment with PowerShell scripting for environment configuration that would enable more systematic deployment and testing of further anti-evasion techniques that are adopted in practice today [18]. Miramirkhani et al. [23] describe the use of 'wear-and-tear' artefacts, in line with how we utilise Common Artefacts Creation, however there would be further scope to examine the impact of more sophisticated 'wear-and-tear' artefacts as part of the MORRIGU environment. Key examples include system files and logs that span a greater time period (e.g., years) so that the system appears to have existed and been active for a significant period of time, making it a valuable target for malware execution. We currently focus our investigations on the Windows operating system (Windows 10 and Windows 7), and so further research would seek to look at evasive malware as deployed on other operating systems.

Whilst behavioural analysis is widely used in malware analysis, one of the limitations in this current study is that none of the evasive samples have been statically analysed to identify ground truth attributes of how the malware should behave. Whilst network and process activity are good metrics for malware analysis, it is quite possible that some samples within the context-aware testing were using evasion techniques not identified within this study. Other common evasion methods could include stalling evasion that makes use of NOP loops or other techniques for significantly delaying the execution time of activities within virtualised environments [4]. This effectively would slow the execution down in such a way that the malware would not deploy within the time period that the system is active for during analysis. While static analysis could have potentially identified such behaviours, the success of this can be hindered by code obfuscation techniques which are commonly used in malware creation [39]. Further work would explore the use of combined dynamic and static analysis methods for studying evasive characteristics of malware, and further developing a corpus of evasive malware samples for the wider research community.

6. Conclusion

Our investigation has developed further understanding related to evasion techniques deployed by malware developers and has presented systematic methods for testing anti-evasion malware triggers using the MORRIGU analysis environment, such as Reverse Turing Tests, Common Artefacts Creation and System Time Jumps. Naturally, malware developers continue to develop evasion techniques and so MORRIGU is specifically designed to support extension for further anti-evasion methods using PowerShell scripting tools. As malware samples increase in size, there is the need to deploy automated analysis platforms such as Cuckoo sandbox. However, it is important to note as shown by our study that evasive malware may not necessarily be noticeable in summary reports and scores, and so combined interactive methods of investigation need to be treated as complimentary. Our study identifies 40 current evasive malware samples that have been reported by the CIS top 10 [36] between August 2019 to January 2020, showing that they are current and genuine threats being observed 'in the wild'. Whilst our current sample is relatively small, it is larger than other previous works whilst also establishing ground truth, a recognised limitation of prior work [4]. Future work will extend MORRIGU and our data sample to explore further evasion strategies.

Funding: This research received no external funding.

Author Contributions: Conceptualization, Alan Mills and Phil Legg; Data curation, Alan Mills; Investigation, Alan Mills and Phil Legg; Methodology, Alan Mills and Phil Legg; Project administration, Phil Legg; Software, Alan Mills; Supervision, Phil Legg; Writing – original draft, Alan Mills and Phil Legg; Writing – review & editing, Alan Mills and Phil Legg.

References

1. Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. A Survey on Automated Dynamic Malware-Analysis Techniques and Tools. *ACM Comput. Surv.* **2008**, *44*. doi:10.1145/2089125.2089126.
2. Or-Meir, O.; Nissim, N.; Elovici, Y.; Rokach, L. Dynamic malware analysis in the modern era—A state of the art survey. *ACM Computing Surveys (CSUR)* **2019**, *52*, 1–48.
3. Moser, A.; Kruegel, C.; Kirda, E. Limits of static analysis for malware detection. Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). IEEE, 2007, pp. 421–430.
4. Bulazel, A.; Yener, B. A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web. Proceedings of the 1st Reversing and Offensive-Oriented Trends Symposium; Association for Computing Machinery: New York, NY, USA, 2017; ROOTS. doi:10.1145/3150376.3150378.
5. Cuckoo Sandbox. https://cuckoosandbox.org. Accessed: 2020-04-23.
6. JOE Security. https://www.joesecurity.org. Accessed: 2020-04-23.
7. HookMe. https://code.google.com/archive/p/hookme/. Accessed: 2020-04-23.
8. Firdausi, I.; Lim, C.; Erwin, A.; Anto, S.N. Analysis of machine learning techniques used in behavior-based malware detection. 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, 2010, pp. 201–203.
9. Tian, R.; Islam, R.; Batten, L.; Versteeg, S. Differentiating malware from cleanware using behavioural analysis. 2010 5th International Conference on Malicious and Unwanted Software, 2010, pp. 23–30.
10. Hansen, S.S.; Larsen, T.M.T.; Stevanovic, M.; Pedersen, J.M. An approach for detection and family classification of malware based on behavioral analysis. 2016 International Conference on Computing, Networking and Communications (ICNC), Workshop on Computing, Networking and Communications (CNC), 2016, pp. 1–5.
11. Tobiyama, S.; Yamaguchi, Y.; Shimada, H.; Ikuse, T.; Yagiup, T. Malware Detection with Deep Neural Network using Process Behavior. 2016 IEEE 40th Annual Computer Software and Applications Conference, 2016, pp. 577–582.
12. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications* **2020**, *153*, 102526. doi:https://doi.org/10.1016/j.jnca.2019.102526.

13.   Mills, A.; Spyridopoulos, T.; Legg, P. Efficient and Interpretable Real-Time Malware Detection Using Random-Forest. 2019 International Conference on Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA), 2019, pp. 1–8.

14.   Chumachenko, K. Machine Learning Methods for Malware Detection and Classification **2017**.

15.   Python-scriptable Reverse Engineering Sandbox. https://github.com/Cisco-Talos/pyrebox. Accessed: 2020-04-23.

16.   Rhode, M.; Burnap, P.; Jones, K. Early-stage malware prediction using recurrent neural networks. *Computers and Security* **2018**, *77*, 578 – 594. doi:https://doi.org/10.1016/j.cose.2018.05.010.

17.   Wagner, M.; Fischer, F.; Luh, R.; Haberson, A.; Rind, A.; Keim, D.A.; Aigner, W.; Borgo, R.; Ganovelli, F.; Viola, I. A survey of visualization systems for malware analysis. EG conference on visualization (EuroVis)-STARs. The EGA, 2015, pp. 105–125.

18.   LastLine. Labs Report at RSA: Evasive Malware's Gone Mainstream. https://www.lastline.com/labsblog/labs-report-at-rsa-evasive-malwares-gone-mainstream/. Accessed: 2019-06-23.

19.   Afianian, A.; Niksefat, S.; Sadeghiyan, B.; Baptiste, D. Malware Dynamic Analysis Evasion Techniques: A Survey. *arXiv preprint arXiv:1811.01190* **2018**.

20.   Keragala, D. Detecting malware and sandbox evasion techniques. *SANS Institute InfoSec Reading Room* **2016**, *16*.

21.   Lau, B.; Svajcer, V. Measuring virtual machine detection in malware using DSD tracer. *Journal in Computer Virology* **2010**, *6*, 181–195.

22.   Liston, T.; Skoudis, E. On the cutting edge: Thwarting virtual machine detection (2006), 2009.

23.   Miramirkhani, N.; Appini, M.P.; Nikiforakis, N.; Polychronakis, M. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017, pp. 1009–1024.

24.   Pektaş, A.; Acarman, T. A dynamic malware analyzer against virtual machine aware malicious software. *Security and Communication Networks* **2014**, *7*, 2245–2257.

25.   Singh, J.; Singh, J. Challenges of Malware Analysis: Obfuscation Techniques. *International Journal of Information Security Science* **2018**, *7*, 100.

26.   Wueest, C. Threats to virtual environments. *Symantec Security Response. Version* **2014**, *1*.

27.   Brumley, D.; Hartwig, C.; Liang, Z.; Newsome, J.; Song, D.; Yin, H. Automatically identifying trigger-based behavior in malware. In *Botnet Detection*; Springer, 2008; pp. 65–88.

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

29.   Ehteshamifar, S.; Barresi, A.; Gross, T.R.; Pradel, M. Easy to Fool? Testing the Anti-evasion Capabilities of PDF Malware Scanners. *arXiv preprint arXiv:1901.05674* **2019**.

30.   Noor, M.; Abbas, H.; Shahid, W.B. Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis. *Journal of Network and Computer Applications* **2018**, *103*, 249–261.

31.   Veerappan, C.S.; Keong, P.L.K.; Tang, Z.; Tan, F. Taxonomy on malware evasion countermeasures techniques. 2018 IEEE 4th World Forum on Internet of Things (WF-IoT). IEEE, 2018, pp. 558–563.

32.   Royal, P. Entrapment: Tricking malware with transparent, scalable malware analysis. *Black Hat* **2012**.

33.   Chen, P.; Huygens, C.; Desmet, L.; Joosen, W. Advanced or not? A comparative study of the use of anti-debugging and anti-VM techniques in generic and targeted malware. IFIP International Conference on ICT Systems Security and Privacy Protection. Springer, 2016, pp. 323–336.

34.   Lindorfer, M.; Kolbitsch, C.; Comparetti, P.M. Detecting environment-sensitive malware. International Workshop on Recent Advances in Intrusion Detection, 2011, pp. 338–357.

35.   VirusShare.com. https://virusshare.com/. Accessed: 2020-03-29.

36.   CIS Centre for Internet Security. https://www.cisecurity.org/. Accessed: 2020-05-08.

37.   Maduranga, K. Investigate Windows Management Instrumentation (WMI) Attacks in Windows Operating Systems. PhD thesis, 2017.

38.   Walker, A.; Amjad, M.F.; Sengupta, S. Cuckoo's Malware Threat Scoring and Classification: Friend or Foe? 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), 2019, pp. 0678–0684.

39.   Moser, A.; Kruegel, C.; Kirda, E. Limits of static analysis for malware detection. Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). IEEE, 2007, pp. 421–430.