


Article

Investigating Anti-Evasion Malware Triggers Using Automated Sandbox Reconfiguration Techniques

Alan Mills and Phil Legg * 

Computer Science Research Centre, University of the West of England, Bristol BS16 1QY, UK;
Alan2.Mills@live.uwe.ac.uk

* Correspondence: Phil.Legg@uwe.ac.uk

Received: 12 October 2020; Accepted: 18 November 2020; Published: 20 November 2020



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 of anti-evasion malware triggers for uncovering malware that may attempt to conceal itself when deployed in a traditional sandbox environment. To facilitate our investigation, we developed a tool called MORRIGU that couples together both automated and human-driven analysis for systematic testing of anti-evasion methods using dynamic sandbox reconfiguration techniques. This is further supported by visualisation methods for performing comparative analysis of system activity when malware is deployed under different sandbox configurations. 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. We also perform a comparative study using Cuckoo sandbox to demonstrate the limitations of adopting only automated analysis tools, to justify the exploratory analysis provided by MORRIGU. By adopting a clearer systematic process for uncovering anti-evasion malware triggers, as supported by tools like MORRIGU, this study helps to further the research of evasive malware analysis so that we can better defend against such future attacks.

Keywords: malware analysis; context-aware malware; anti-evasion malware detection

1. 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, see, e.g., in [8–11]. However, these tools often require extensive configuration and set-up 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 (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). 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 an 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 demonstrate how the systematic approach can identify characteristics of interest, while maintaining efficient runtime in an automated manner.

The main contributions of this paper are as follows.

1. We propose a dynamic sandbox reconfiguration tool called MORRIGU that couples together both automated and human-driven analysis for configurable anti-evasion testing.
2. Using MORRIGU, we conduct a systematic investigation of malware evasion techniques, and how single and multiple anti-evasion methods can identify such malware behaviours.
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 such as Cuckoo can fail to identify malicious signatures of evasive malware samples.
4. As a result of our findings, we provide both our software tools and our curated evasive malware dataset as open source to the community to support further research.

2. 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.

2.1. 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 preprocessing 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, which 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 reproducibility between malware analysis platforms, as reporting may vary 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.

2.2. 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. While 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 limitation 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 four sandbox environments, each with a different system setup (three Windows XP virtual machines and one Anubis sandbox). Behavioural comparisons are made across all four environments when executing the malware sample to identify any significantly different behavioural patterns that may indicate particular configuration triggers. However, this is a computational expensive approach and is still susceptible to samples that could potential evade analysis under all environments, for example using network environment checks.

3. 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 in terms of the type of malware deployed, while ensuring that the samples were also relevant in modern systems and being observed “in the wild”, unlike earlier malware datasets that may now be considered outdated. Interesting observations for our data collection revealed that five malware families were recorded in the top 10 every 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. At the time of collection, the researchers had no knowledge of whether samples were evasive in their nature, or whether specific anti-evasive methods would cause a malware function to trigger. All samples were downloaded using the VirusShare service. MORRIGU can be configured to download directly from VirusShare to avoid accidental execution of samples. The MORRIGU software and our datasets are available for download from the author’s home page: <http://www.plegg.me.uk>.

4. 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 then provide our methodology for using MORRIGU to assess the variants of context-aware malware triggers adopted by different malware samples.

4.1. 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 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, while 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 a 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.

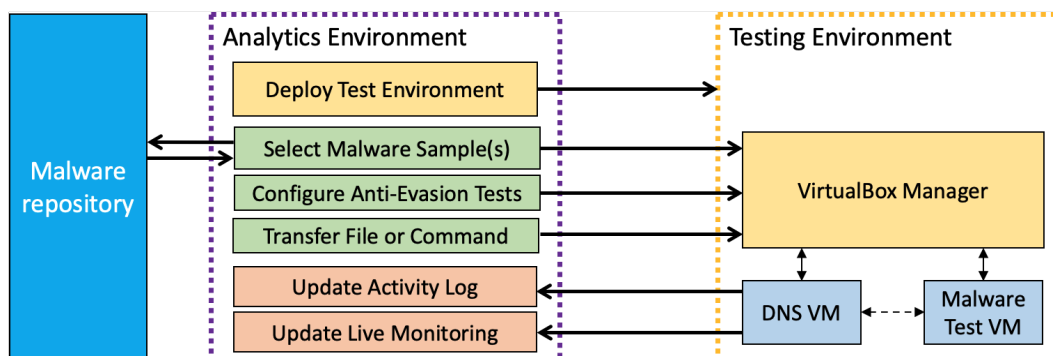


Figure 1. System Architecture to show the interactions between the analytics and testing environments.

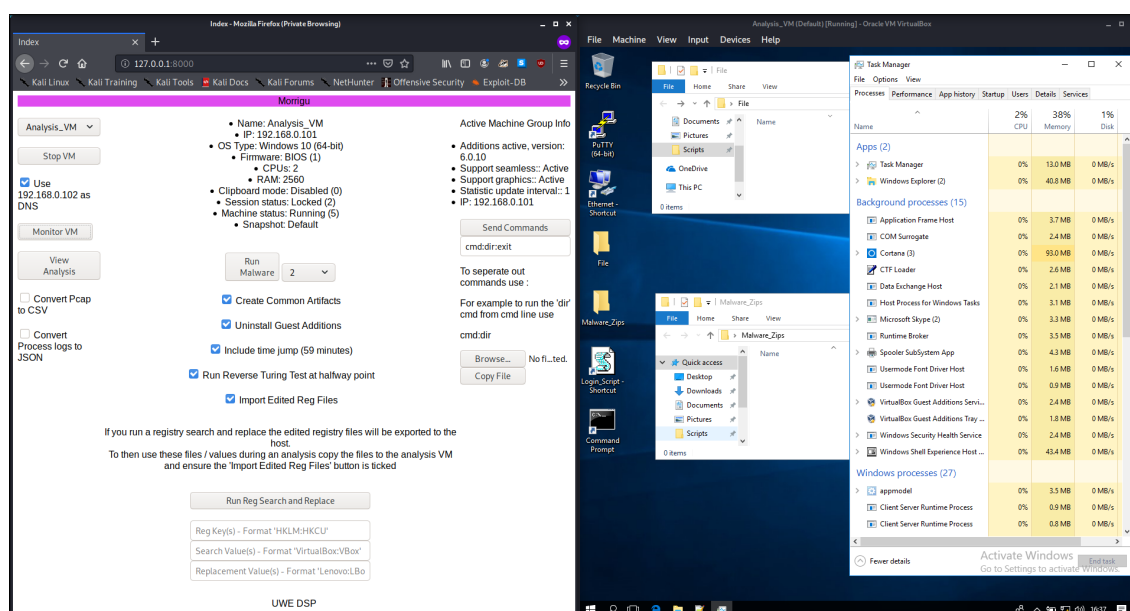


Figure 2. Example to show the MORRIGU analytics environment and configuration options (left), and the associated malware testing environment using a Windows 10 virtual machine (right).

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 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 set-up and torn down, helping to limit the risk of cross contamination caused by malware activity trying to escape from the sandboxed virtualised environment while 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 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). Our initial release of MORRIGU supports five different anti-evasion methods, which we choose due to them being recognised as common methods of malware evasion [2], yet could also yield different malware behaviours based on how the method parameters are configured and so benefit from dynamic reconfiguration. The five methods are as follows.

- **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 set-up, 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 min (as a default value); 59 min 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 four 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 renamed 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 s, 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 all possible combinations for text replacement would be unfeasible, and is considered out of scope for this paper, although our approach makes it easier to test a given set of text replacement terms for investigation.

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 ransomware encrypts files within the testing environment and creates ReadMe files to notify the user of the attack (the time difference between the testing and analytics environments is due to the independent time management of the testing environment required for techniques such as system time jump). Ransomware by its very nature will make itself known to the target (as a payment is often demanded to retrieve a decryption key). Other forms of malware such as Infostealers, Downloaders and Remote Access Trojans may intentional be more discrete, however their behaviour can still be observed using live monitoring tools.

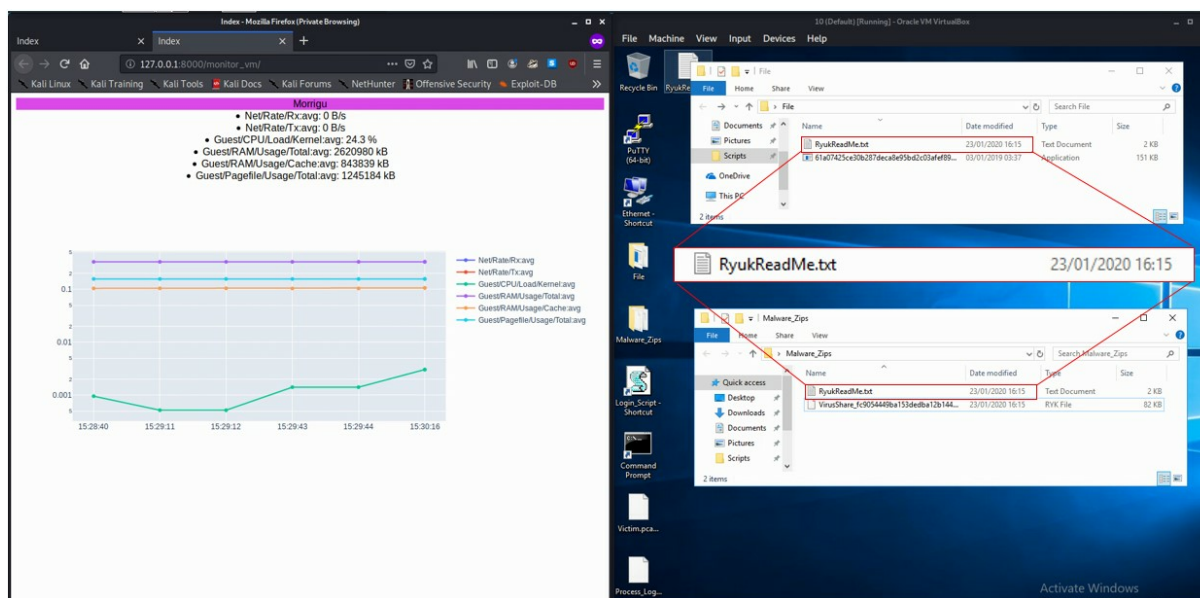


Figure 3. Example to show live monitoring in the MORRIGU GUI (left) when the Ryuk Ransomware has been deployed using MORRIGU within the Windows 10 testing environment (right).

MORRIGU 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 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. The use of simple and intuitive visual analytics for malware investigation can provide quick summarisation of complex and detailed activity logs. As 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 (a data attribute that records how many objects or files a process has interacted with) 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 machine learning pipeline [13] and would help to alleviate many of the issues related to feature engineering and data preprocessing required for classification and regression tasks.

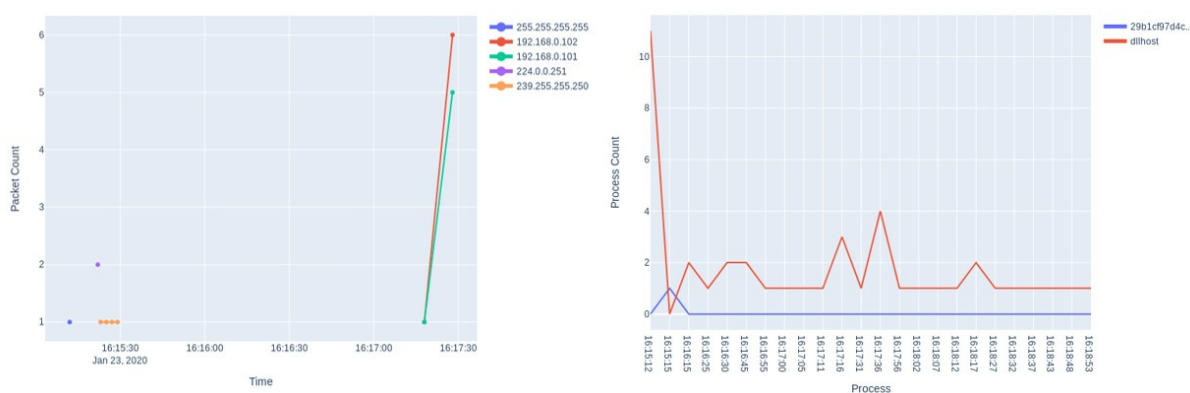


Figure 4. Network and process activity for Bayrob—No Change Environment.



Figure 5. Network and process activity for Bayrob—Guest Additions Uninstalled.

4.2. Testing Methodology

Malware samples were accessed by MORRIGU from our malware repository using batch processing to reconfigure the testing environment, deploy the malware sample and capture 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 three groups; (1) no activity that deviates from the observed baseline system behaviours, (2) activity that results in new communication being made to a single IP address and (3) 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 or generic Content Delivery Networks are excluded using rule-based filtering (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). 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.

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

5.1. 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, also known as the Tinba malware family), we append the last four 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, four triggered with TJ, 18 triggered with RTT, eight triggered with RegEdit, and two triggered with CAC. While samples may be triggered with further anti-evasive methods used, we show here the minimal required set of methods. Furthermore, there are four 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 (Neutrino_4929, Gozi_3130 and Strictor_5e9c) were found to trigger under different “single” checks: specifically, they were triggered either by GAU or by RegEdit. Given that GAU performs a RegEdit as part of the removal process, we consider the RegEdit aspect to be what provokes the behaviour. 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). It should be noted that two of these cases were the result of malicious traffic being redirected or blocked. These variations of Tinba illustrate the evolution of malware capabilities to naturally extend upon existing examples and require more sophisticated evasion checks.

5.2. Anti-Evasion Results Based on Malware Families

Table 2 shows the eight malware classifications that we found respond to anti-evasion checks using the MORRIGU environment. As 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.

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

Malware Sample	Guest Additions Uninstalled	Configuration Options			Common Artefacts Creation
		Time Jump	Reverse Turing Test	Registry Edit	
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)				✓	

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 utilise evasion.

Malware Classification	Guest Additions Uninstalled	Configuration Options			Common Artefacts Creation	Sample Count
		Time Jump	Reverse Turing Test	Registry Edit		
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

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 min. However, this was not the case for all samples which used hardcoded IPs, as one sample of Dridex (Banking Trojan) connected to only two malicious IPs, spread out over 4 min, with the largest amount of data (13k bytes) being sent in less than 6 s. 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 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.

6. 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 utilise 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. By comparison MORRIGU is designed as a point-and-click tool. Once the server is started, the system performs the network setup automatically 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. MORRIGU is designed to be user-friendly with a lower barrier to entry for researchers new to malware analysis.

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 predefined signature rule-based system. Cuckoo documentation suggests that these can be treated as three classifications: low (0–4), medium (4–7) or high (7–10). However, as reported by Walker et al. [37] this scoring classification can be misleading due to the underlying methodology of severity classification. They suggest that greater emphasis is required on analysing the behaviour 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 [37]. While 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. First, 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. Moreover, 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. While this is supposedly between 0 and 10, two 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. While 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 three in Windows 10) as shown in brackets. Surprisingly, some cases of

Pushdo, Dridex and Wannacry actually 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 checks as used in MORRIGU (e.g., System Time Jump, RegEdits or Common Artefacts Creation). The default RTT in Cuckoo is limited to generating randomised mouse movement that could easily be identified. This provides further justification for developing more sophisticated anti-evasion checks such as those in the MORRIGU toolkit, that could be used in conjunction with analysis platforms such as Cuckoo.

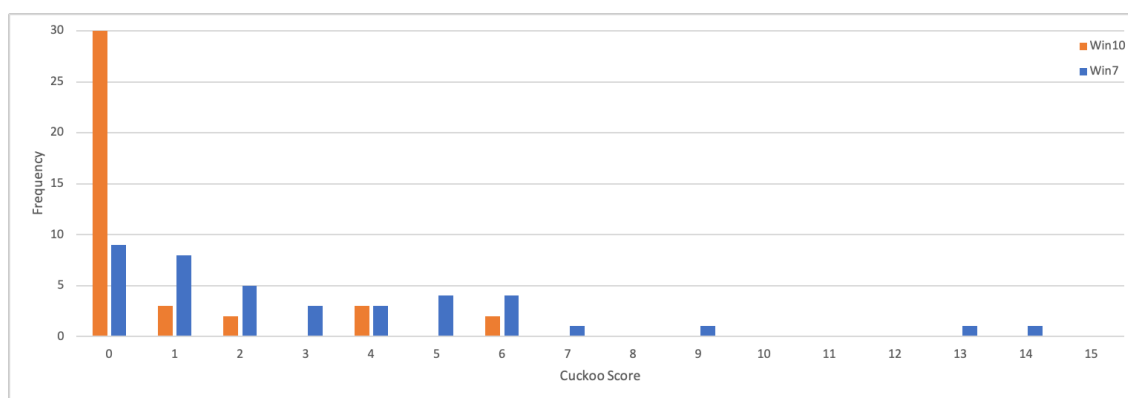


Figure 6. Distribution of Cuckoo scoring for the 40 evasive malware samples (Windows 10 (Orange): Mean = 1.2, Median = 0.6; Windows 7 (Blue): Mean = 3.66, Median = 2.2).

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

Table 3. Cont.

Malware Sample	Cuckoo Score (Win10)	Cuckoo Score (Win7)
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)

7. Discussion

We present a study of assessing anti-evasive malware triggers, using a systematic testing and reconfiguration environment that we refer to as MORRIGU. This enables a streamlined approach for preparing a reusable virtualised sandbox environment that allows for different evasive properties to be studied, deploying malicious software samples from a repository and capturing the resulting behaviours that can be examined using a visual analytics dashboard. From an initial set of 251 malware samples that were gathered based on the CIS top 10 malware threats between August 2019 and January 2020, we use MORRIGU to study the nature of evasive malware in the currently trending threats. We found that at least 40 (16% of our sample set) exhibit evasive methods, such that 53% of samples would execute and reveal the malicious behaviour in any given environment (i.e., a fresh install of the Windows 10 environment), whereas when also factoring in the anti-evasion checks in MORRIGU, we observe that this increases to 69% of samples. While this percentage increase may seem relatively low, it is important to appreciate the true nature of this finding—this represents 40 malware samples that if examined under “normal” testing conditions could be deemed as benign software, and yet, if allowed into a production environment, could cause serious damage. This work is designed to provoke further investigation into the nature of evasive malware and contributes software tools to aid the community in doing so. In our dataset, 78 samples did not exhibit any malicious activity. It is important to recognise here that these samples could well be evasive, whereby they may only perform the intended malicious function under a different set of environment conditions. While it is impossible to test every possible environment configuration, MORRIGU enables efficient testing of a larger number of possible configurations; therefore, improving the likelihood of identifying a trigger for deployment. However, another significant challenge in malware research is that samples could be obsolete and associated with older software platforms or applications. Our samples were gathered between August 2019 and January 2020, which could be designed to exploit vulnerabilities associated with software versions particular to that time period, which could potentially be redundant already. This is yet why identifying a subset of evasive malware, and more specifically,

offering tools to identify future evasive malware, is both important to our community and remains a continual challenge.

Our research shows how malware families may depend on a particular set of evasion strategies. Clearly, the presence of “Guest Additions” is the most noticeable indicator that the environment is virtualised and could be a analysis sandbox, and so removal of this software as well as references to “VirtualBox”, “VBox” or “VMware” in the registry would be a suitable first step for most investigations (11 samples triggered follow removal of guest additions). Table 1 shows that many of the Banking Trojans were triggered by the Reverse Turing Test, although not exclusively. IcedID required the RTT in conjunction with the Registry Edit, while the Tinba variants all required different evasion methods, including registry edits and common artefacts creation. The Time Jump method was required to provoke the Qakbot Infostealer to trigger, as well as the Nymaim Ransomware. Four of the samples (Adposhel, IcedID, Lokibot and Pushdo) required multiple evasion methods to be deployed, while all five evasion methods were observed within the dataset. Despite what is a seemingly low number of cases, this again highlights an important point: there are malware variants that will deploy multiple evasion methods. Even if this number is currently low (based on our current dataset), the reality is that more samples will adopt this approach in future. Much like a game of hide and seek, for the malware developer to hide this functionality in software is relatively trivial, yet for the malware analyst it is much more difficult to find, simply because there may be many possibilities of how the original developer specified the trigger. Adding multiple evasion methods would make a sample increasingly more successful at avoiding detection in a sandbox environment as the complexity of search by the analyst would increase exponentially. At the very minimum, our research would suggest that all five of the anti-evasion checks performed in MORRIGU should be tested in practice, as we have identified active threats in the wild that have triggered based on each of these. Importantly to reiterate, at the time of data collection the researchers had no knowledge of which anti-evasion methods would work for the different malware variants, and the malware was gathered based on the classification type of the malware rather than because of having evasive characteristics. In practice, an analyst may primarily want to study whether a sample will trigger or not, rather than the specific combination of anti-evasion methods that caused it to trigger. Therefore, one could perform a test using many anti-evasion checks, and then if a trigger is identified, work backwards to identify which checks caused the reaction to occur. We have illustrated how these checks could be performed, with relatively simple parameter selection. The five methods were adopted due to being common methods of malware evasion [2]. If we omit “Guest Additions”, as this could be considered as initial configuration for most investigations, the remaining anti-evasion methods are complimentary as they examine different attributes about the system and its user (system time, system applications and parameters, user files and folder, and user interactions). While in theory there are further anti-evasion methods that could be imagined, they would typically fall within the scope of these four areas. Further research would, therefore, aim to explore more extensive parameter selection and modification for the anti-evasion checks.

Evasive malware detection shares a number of commonalities with insider threat detection [38,39]. In the case of insider threat detection, an adversary will have privileged access and knowledge about the organisation, and will want to perform some malicious act whilst concealing their behaviour. Likewise, malware developers will understand how analysts may utilise a sandbox environment to study malware samples, and will again aim to conceal their behaviours. While we can detect “noticeable” activities of insider threat using statistics and comparative models of behaviour, the same is true of evasive malware assuming that appropriate tooling such as MORRIGU is available to facilitate this. What is important for both domains is that by improving the suite of automated tools available to the analyst means that the “noticeable” cases can be resolved quickly, allowing analyst time and effort to be devoted to more complex cases such as an APT that may require a longitudinal perspective.

Similar to our approach of Common Artefacts Creation, Miramirkhani et al. [23] describe the use of “wear-and-tear” artefacts. In particular, they look at modelling artefacts that cover a greater period of time (e.g., years), and also creating associated log files that show artefact usage during that time so that the system appears to have been operational and active for a significant period of time which would make it a valuable target for malware execution. In recent times, there has become a great interest in the notion of “digital twins” for replicating production systems (e.g., Industrial Control Systems) in simulation [40]. With a digital twin, one could model the system characteristics that they wish to protect, rather than speculating on the system characteristics that the malware will execute against. In this manner, if malware execution does not provoke an observable reaction, it may reveal that the system is not vulnerable to attack—or rather, may not be the intended target for attack. Stuxnet [41] is a well-known example of this, since it required a specific Siemens programmable logic controller to be present, with a matching serial number, in order for the malware to truly deploy the payload. How we study malware that aims to remain dormant in sandbox analysis environments, versus how we study malware that has specific execution requirements before revealing a malicious payload, is an important aspect to consider. This also highlights the limitations of relying on behavioural malware analysis alone. Static analysis is the process of examining the function calls performed by software, which could help to identify suspicious routines that would suggest evasive behaviours, such as stalling evasion to delay execution time [4]. Behavioural analysis could easily miss such activity if the delay imposed is longer than the analysis period. However, static analysis can be challenging, especially where code obfuscation is used to conceal the intended function [3]. While there exist proposed methods for static and dynamic malware analysis [42], further work should address how such combined methods can be extended to better identify evasive or targeted characteristics within malware.

8. Conclusions

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. Furthermore, we fully appreciate that malware analysis is an arms race between those attacking systems, and those trying to defend systems. Whilst such tools as MORRIGU could potentially be used by attackers, it is well recognised that malware development takes many forms, from script kiddies through to sophisticated organised crime and nation states. As defenders, we offer these tools in an attempt to close the gap, fully in the knowledge that defence is continually playing catch-up to those developing malware. 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 malware threats between August 2019 to January 2020, showing that they are current threats being observed “in the wild”. While 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 curated dataset to explore further evasion strategies, including how bespoke simulated networking access (e.g., INetSim) could be incorporate for anti-evasion detection beyond “catch-all” and “block-all” rules, and also studying patterns and trends in malware evasive characteristics over time in conjunction with static analysis techniques.

Author Contributions: Conceptualization, A.M. and P.L.; Data curation, A.M.; Investigation, A.M. and P.L.; Methodology, A.M. and P.L.; Project administration, P.L.; Software, A.M.; Supervision, P.L.; Writing—original draft, A.M. and P.L.; Writing—review and editing, A.M. and P.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

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*, 6. [CrossRef]
2. Or-Meir, O.; Nissim, N.; Elovici, Y.; Rokach, L. Dynamic malware analysis in the modern era—A state of the art survey. *ACM Comput. Surv.* **2019**, *52*, 1–48. [CrossRef]
3. Moser, A.; Kruegel, C.; Kirda, E. Limits of static analysis for malware detection. In Proceedings of the Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, USA, 10–14 December 2007; pp. 421–430.
4. Bulazel, A.; Yener, B. A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web. In Proceedings of the 1st Reversing and Offensive-Oriented Trends Symposium, Vienna, Austria, 16–17 November 2017; Association for Computing Machinery: New York, NY, USA, 2017. [CrossRef]
5. Cuckoo Sandbox. Available online: <https://cuckoosandbox.org> (accessed on 23 April 2020).
6. JOE Security. Available online: <https://www.joesecurity.org> (accessed on 23 April 2020).
7. HookMe. Available online: <https://code.google.com/archive/p/hookme/> (accessed on 23 April 2020).
8. Firdausi, I.; Lim, C.; Erwin, A.; Anto, S.N. Analysis of machine learning techniques used in behavior-based malware detection. In Proceedings of the 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, Jakarta, Indonesia, 2–3 December 2010; pp. 201–203.
9. Tian, R.; Islam, R.; Batten, L.; Versteeg, S. Differentiating malware from cleanware using behavioural analysis. In Proceedings of the 5th International Conference on Malicious and Unwanted Software, Nancy, France, 19–20 October 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. In Proceedings of the International Conference on Computing, Networking and Communications (ICNC), Workshop on Computing, Networking and Communications (CNC), Columbus, OH, USA, 8–10 December 2016; pp. 1–5.
11. Tobiyama, S.; Yamaguchi, Y.; Shimada, H.; Ikuse, T.; Yagiup, T. Malware Detection with Deep Neural Network using Process Behavior. In Proceedings of the 2016 IEEE 40th Annual Computer Software and Applications Conference, Atlanta, GA, USA, 10–14 June 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. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. [CrossRef]
13. Mills, A.; Spyridopoulos, T.; Legg, P. Efficient and Interpretable Real-Time Malware Detection Using Random-Forest. In Proceedings of the International Conference on Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA), Oxford, UK, 3–4 June 2019; pp. 1–8.
14. Chumachenko, K. Machine Learning Methods for Malware Detection and Classification. 2017. Available online: <https://www.theseus.fi/handle/10024/123412> (accessed on 23 April 2020).
15. Python-Scriptable Reverse Engineering Sandbox. Available online: <https://github.com/Cisco-Talos/pyrebox> (accessed on 23 April 2020).
16. Rhode, M.; Burnap, P.; Jones, K. Early-stage malware prediction using recurrent neural networks. *Comput. Secur.* **2018**, *77*, 578–594. [CrossRef]
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. In Proceedings of the EG conference on visualization (EuroVis)-STARs, Sardinia, Italy, 25–29 May 2015; pp. 105–125.

18. LastLine. Labs Report at RSA: Evasive Malware's Gone Mainstream. Available online: <https://www.lastline.com/labsblog/labs-report-at-rsa-evasive-malwares-gone-mainstream/> (accessed on 23 June 2019).
19. Afianian, A.; Niksefat, S.; Sadeghiyan, B.; Baptiste, D. Malware Dynamic Analysis Evasion Techniques: A Survey. *arXiv* **2018**, arXiv:1811.01190.
20. Keragala, D. *Detecting Malware and Sandbox Evasion Techniques*; SANS Institute InfoSec Reading Room: Bethesda, MD, USA, 2016; p. 16.
21. Lau, B.; Svajcer, V. Measuring virtual machine detection in malware using DSD tracer. *J. Comput. Virol.* **2010**, *6*, 181–195. [CrossRef]
22. Liston, T.; Skoudis, E. On the Cutting Edge: Thwarting Virtual Machine Detection (2006). Available online: <https://ci.nii.ac.jp/naid/10021375130/> (accessed on 23 April 2020).
23. Miramirkhani, N.; Appini, M.P.; Nikiforakis, N.; Polychronakis, M. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 1009–1024.
24. Pektaş, A.; Acarman, T. A dynamic malware analyzer against virtual machine aware malicious software. *Secur. Commun. Netw.* **2014**, *7*, 2245–2257. [CrossRef]
25. Singh, J.; Singh, J. Challenges of Malware Analysis: Obfuscation Techniques. *Int. J. Inf. Secur. Sci.* **2018**, *7*, 100.
26. Wueest, C. Threats to Virtual Environments. Symantec Security Response. Version. 2014. Available online: <https://vxug.fakedoma.in/archive/Symantec/threats-to-virtual-environments-14-en.pdf> (accessed on 23 April 2020).
27. Brumley, D.; Hartwig, C.; Liang, Z.; Newsome, J.; Song, D.; Yin, H. Automatically identifying trigger-based behavior in malware. In *Botnet Detection*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 65–88.
28. Mehra, M.; Pandey, D. Event triggered malware: A new challenge to sandboxing. In Proceedings of the Annual IEEE India Conference (INDICON), New Delhi, India, 17–20 December 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* **2019**, arXiv:1901.05674.
30. Noor, M.; Abbas, H.; Shahid, W.B. Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis. *J. Netw. Comput. Appl.* **2018**, *103*, 249–261. [CrossRef]
31. Veerappan, C.S.; Keong, P.L.K.; Tang, Z.; Tan, F. Taxonomy on malware evasion countermeasures techniques. In Proceedings of the IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 5–8 February 2018; pp. 558–563.
32. Royal, P. Entrapment: Tricking Malware with Transparent, Scalable Malware Analysis. Black Hat. 2012. Available online: <http://media.blackhat.com/bh-eu-12/Royal/bh-eu-12-Royal-Entrapment-WP.pdf> (accessed on 12 January 2020).
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. In Proceedings of the IFIP International Conference on ICT Systems Security and Privacy Protection, Ghent, Belgium, 30 May–1 June 2016; pp. 323–336.
34. Lindorfer, M.; Kolbitsch, C.; Comparetti, P.M. Detecting environment-sensitive malware. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Menlo Park, CA, USA, 20–21 September 2011; pp. 338–357.
35. VirusShare.com. Available online: <https://virusshare.com/> (accessed on 29 March 2020).
36. CIS Centre for Internet Security. Available online: <https://www.cisecurity.org/> (accessed on 8 May 2020).
37. Walker, A.; Amjad, M.F.; Sengupta, S. Cuckoo's Malware Threat Scoring and Classification: Friend or Foe? In Proceedings of the IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 7–9 January 2019; pp. 678–684.
38. Legg, P.A.; Buckley, O.; Goldsmith, M.; Creese, S. Automated Insider Threat Detection System Using User and Role-Based Profile Assessment. *IEEE Syst. J.* **2017**, *11*, 503–512. [CrossRef]
39. Legg, P.A. Visualizing the insider threat: Challenges and tools for identifying malicious user activity. In Proceedings of the IEEE Symposium on Visualization for Cyber Security (VizSec), Chicago, IL, USA, 26 October 2015; pp. 1–7. [CrossRef]

40. Bitton, R.; Gluck, T.; Stan, O.; Inokuchi, M.; Ohta, Y.; Yamada, Y.; Yagyu, T.; Elovici, Y.; Shabtai, A. Deriving a Cost-Effective Digital Twin of an ICS to Facilitate Security Evaluation. In *Computer Security*; Lopez, J., Zhou, J., Soriano, M., Eds.; Springer: Cham, Switzerland, 2018; pp. 533–554.
41. Langner, R. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Secur. Priv.* **2011**, *9*, 49–51. [[CrossRef](#)]
42. Shijo, P.; Salim, A. Integrated Static and Dynamic Analysis for Malware Detection. *Procedia Comput. Sci.* **2015**, *46*, 804–811. [[CrossRef](#)]

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).