

WINTER 2020



VMRAY

DEFEATING EVASIVE MALWARE

EXECUTIVE SUMMARY

Sandboxes are automated behavior-based malware analysis systems that are at the core of most network security solutions today. The deployment of sandboxes to detect advanced threats began over a decade ago. Back then, malware authors had already found ways to evade traditional antivirus solutions, which rely on static analysis, by using techniques such as polymorphism, metamorphism, encryption, obfuscation and antireversing protection. Malware analysis sandboxes are now considered the last line of defense against advanced threats.

The operating principle of a sandbox is simple. It determines if a file is malicious based on the observed behavior of the file in a controlled environment over a defined analysis period. It does this by recording all the actions performed by the file and determining if any of these represent malicious behavior patterns. Since detection is not based on static signatures, sandboxes can even detect zero-day and targeted malware, previously unknown to security researchers or analysts.

The success of behavior-based malware detection hinges on the behavior exhibited by the file during analysis. Thus, the objective of any sandbox evasion technique is to conceal the real behavior of the malicious file, thereby evading detection. Malware authors are always looking for new, innovative ways to elude sandboxes.

In this whitepaper, we look at three categories of approaches taken by malware to evade sandboxes and explore techniques associated with each approach.

- **Evasion by active detection of analysis**

environment: In the first approach, malware uses several techniques to actively detect the existence of a sandbox. It then conceals malicious intent by

altering its behavior as soon as it determines that it is being executed in a sandbox, thereby evading detection. Since sandboxes often use virtual environments, a widely-used technique in this category is detecting the presence of a virtual machine or hypervisor.

- **Evasion by using time, event or environment based**

triggers: In the second approach, malware delays the execution of a malicious payload until a certain trigger or event occurs. By choosing a trigger that is unlikely to be activated inside a sandbox, malware remains undetected in an analysis environment. As an example, since sandboxes usually spend only a few minutes analyzing each file, malware can evade detection by delaying the execution of a malicious payload by a certain amount of time. Besides timetriggers, malware can also use other events such as a system reboot or user interaction that do not normally occur in a sandbox.

- **Evasion by exploiting sandbox weaknesses:** In the third approach, malware performs malicious operations but exploits weaknesses in the underlying sandbox technology or in the surrounding ecosystem to avoid detection. Evasion techniques that belong to this approach include hook circumvention, using obscure file formats which cannot be handled by the sandbox or exploiting the sandbox's inability to process files that exceed a certain size.

Approach 1: EVASION BY ACTIVE DETECTION OF ANALYSIS ENVIRONMENT

The first approach to sandbox evasion relies on actively detecting the presence of a sandbox by looking for small differences between the analysis environment and the system on which the malicious file is meant to be executed i.e. the intended victim's system. If the malicious file determines that it is being executed in a sandbox, it usually reacts in one of the

following ways to avoid detection.

- It terminates immediately without performing any malicious operations. However, this is likely to raise suspicion.
- It terminates immediately without performing any malicious operations and displays an error message related to a missing system module or a corrupted executable file to avoid raising any suspicion.
- It performs only benign operations, thus prompting the sandbox to classify it as non-malicious.

There are several techniques associated with this approach and in this section we look at four of the most widely used ones.

TECHNIQUE 1: DETECTING THE HYPERVISOR OR THE VIRTUAL ENVIRONMENT

Since sandboxes often use virtual environments, a widely-used technique in this category is detecting the presence of a virtual machine or hypervisor. This evasion technique is not very relevant today since many production environments (workstations and servers) are virtualized and virtualization is no longer primarily the realm of only researchers and malware analysts. To determine if it is being executed in a virtual environment, malware looks for certain specific artifacts. Before the emergence of full hardware support for virtualization, there were several technical artifacts that existed in virtual machines that malware could look for. These include:

- Trying to access port 0x5658 to detect VMWare or detecting Virtualbox via a backdoor
- Looking for generic VM artifacts (for example the popular "Red Pill" method)

However, this is not a very effective detection mechanism today. With hardware virtualization support, there are very few visible artifacts, if any, inside the VM since most hardware aspects are now virtualized and handled by the CPU itself. Therefore, they do not have to be simulated by the hypervisor. A detection mechanism that is still relevant today involves detecting the implementation artifacts of the hypervisor. An example of this is determining the vendor from the MAC address, device ID or the CPU ID or from the existence of certain processes, files, drivers, registry keys or strings in memory.

TECHNIQUE 2: DETECTING SANDBOX ARTIFACTS

In this evasion technique, it is not the hypervisor that the malware is trying to detect, but the sandbox itself. This can be done in any of the following ways:



Severity	Category	Operation	Classification
High	Anti Analysis	Tries to detect application sandbox	
		• Possibly trying to detect "Sandbox" by checking for existence of module "lsidll.dll".	
		• Possibly trying to detect "wine" by calling GetProcAddress on "wine_get_unix_file_name".	
High	Anti Analysis	Tries to detect virtual machine	
		• Reads out system information, commonly used to detect VMs via registry. (Value "SystemDiskVersion" in key "HKEY_LOCAL_MACHINE\HARDWARE\Description\System").	
		• Possibly trying to detect VirtualBox via registry "HKEY_LOCAL_MACHINE\SOFTWARE\Oracle\VirtualBox Guest Additions".	
		• Reads out system information, commonly used to detect VMs via registry. (Value "VideoDiskVersion" in key "HKEY_LOCAL_MACHINE\HARDWARE\Description\System").	

Figure 1 – VMRay Analyzer reporting a malicious file attempting to detect the presence of a sandbox and virtual machine

- Malware can use vendor specific information associated with popular sandbox products. For example, the existence of certain files, processes, drivers, file system structure, Windows ID, username etc. can reveal the presence of a sandbox
- Alternatively, malware can use vendor-specific mechanisms related to the ecosystem. For example, mechanisms to revert the analysis environment back to a clean state after infection (examples include Deepfreeze and Reborn Cards) or mechanisms to perform communication with the sandbox controller

(examples include additional listening ports and a specific network environment such as master server).

- Knowledge of the underlying sandbox technology can also be used by malware to detect the analysis environment. Emulation and Hooking are examples of underlying technologies that can reveal the presence of a sandbox to malware (Refer to the VMRay technology whitepaper for more details). In the case of hooking, sandboxes inject or modify code and data within the analysis system. The 'hook' is essentially a shim layer capturing communication between processes, drivers and the OS. A hook can be implemented in many ways such as: inline hooks, IAT/EAT modification, proxy DLL, filter drivers etc. This makes them detectable by inspecting certain instructions or pointers or by verifying the integrity of the system, for example by verifying hash signatures of relevant system files.

A theoretically perfect emulation-based sandbox is identical to a native system. However, in practice, there are always differences. For example, different instruction semantics and different time lapses during an operation can be exploited by malware to detect an emulation based sandbox and evade it.

TECHNIQUE 3: DETECTING AN ARTIFICIAL ENVIRONMENT

Sandboxes are usually not production systems, but specially set up for malware analysis. Hence, they are not identical to real computer systems and these differences can be detected by malware. Differences may include:

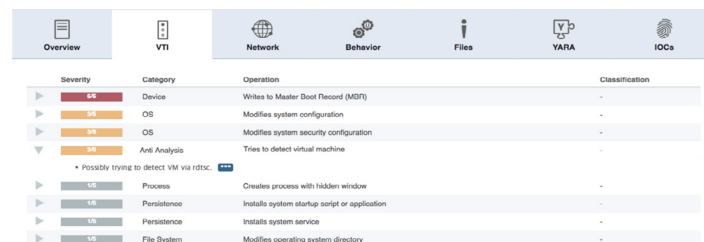
- Hardware properties such as: unusually low screen resolution, no USB 3.0 drivers, lack of 3D rendering capabilities, only one (V)CPU and small hard disk and memory sizes.

- Software properties such as an atypical software stack. For example, no Instant Messenger, no mail client.
- System properties such as uptime ("system restarted 10 seconds ago"), network traffic ("system uptime is days, but only a few megabytes of data have been transmitted over the network"), no printers or only default printers installed.
- User properties such as a clean desktop, clean filesystem, no cookies, no recent files, no user files

TECHNIQUE 4: DETECTING TIMING DIFFERENCES

Monitoring the behavior of an application comes with a time penalty, which can be measured by malware to detect the presence of a sandbox. Sandboxes try to prevent this by faking the time. However, malware can bypass this by incorporating external time sources such as NTP or timestamps included in HTTP requests. An example of timing-based detection is shown in figure 2, where the time-stamp counter is checked by the malware.

Figure 3 shows the Pafish application detecting artifacts that often exist in analysis environments. Malware will also run checks like this. For example, if certain operations take longer than expected, the malware terminates or performs only benign operations on the assumption that it is running inside an analysis environment.



Severity	Category	Operation	Classification
65	Device	Writes to Master Boot Record (MBR)	-
55	OS	Modifies system configuration	-
55	OS	Modifies system security configuration	-
55	Anti Analysis	Tries to detect virtual machine	-
* Possibly trying to detect VM via rdtscl			
15	Process	Creates process with hidden window	-
15	Persistence	Installs system startup script or application	-
15	Persistence	Installs system service	-
15	File System	Modifies operating system directory	-

Figure 2 – VMRay Analyzer reporting a malicious file attempting to detect a VM using the time-stamp counter

APPROACH 2: EVASION BY USING TIME, EVENT OR ENVIRONMENT BASED TRIGGERS

In this approach, malware does not actively try to detect a sandbox or analysis environment. Instead, it delays or postpones its malicious payload until a certain trigger or event occurs. The trigger that is chosen is very unlikely to be activated inside a sandbox. Several techniques can be used to implement a trigger or event based payload delivery.

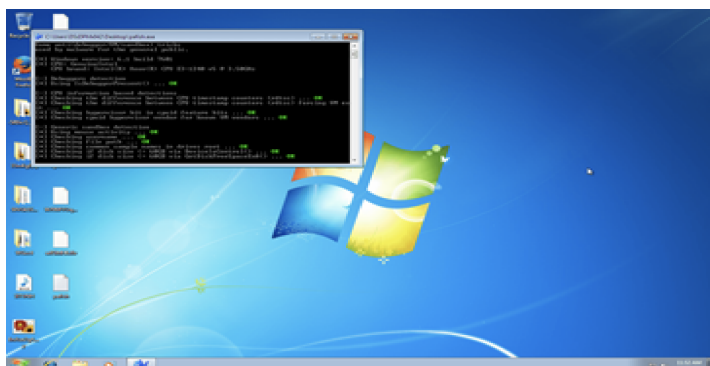


Figure 3 – Pafish detecting artifacts that exist in analysis environments

TECHNIQUE 1: USING TIME BOMBS

One of the most common techniques is to delay execution for a certain amount of time since sandboxes usually run samples only for a few minutes. As with many other evasion techniques, the utilization of time bombs is an ongoing cat and mouse game: the malware goes to sleep, the sandbox tries to detect the sleep and reduce the sleep time, the malware detects the reduced sleep time, the sandbox tries to hide this by also updating system timers and so on. Some of the methods used by malware include:

- Simple to very complex sleep mechanisms.
- Executing only at a certain time or on a specific date: For example, 12AM or the 12th of March.
- Slowing down execution significantly: For example, injecting millions of arbitrary system calls that have

no effect except to slow down execution, especially when being executed in a monitored or emulated environment.

TECHNIQUE 2: WAITING FOR SYSTEM EVENTS

In this technique, malware becomes active only on shutdown, after reboot, or when someone logs on or off. It does this because these triggers are unlikely to be activated inside a sandbox.

TECHNIQUE 3: WAITING FOR USER INTERACTION

Another technique frequently used by malware to evade sandboxes is waiting for specific user actions before becoming active. Again, the trigger that is chosen is very unlikely to be activated inside a sandbox. Examples include:

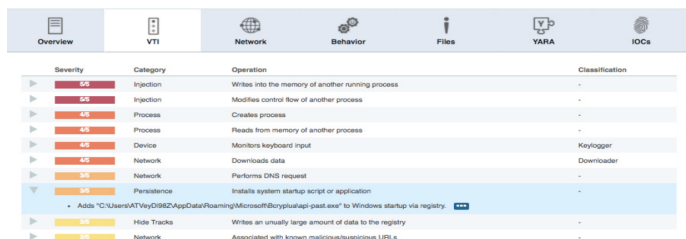
- Waiting for mouse movement or keyboard input.
- Interacting with certain applications, for example browser, email, Skype, an on-line banking application.
- Becoming active after a user has clicked multiple buttons and checked various checkboxes (Fake installers).
- Becoming active only when the user scrolls or clicks (Office documents with malicious embedded content).

TECHNIQUE 4: OPERATING ONLY IN A SPECIFIC TARGET SYSTEM

Sophisticated targeted malware only works on the intended target system. The identification is usually based on the current username, time zone, keyboard layout, IP address or some other system artifacts.

The check itself can be done in various ways, ranging from simple to very complex methods.

- Simple checks include string checks.
- Complex checks that are nearly unbreakable if the expected target environment is not known. For example, decryption with the hash taken from the environment settings.



Severity	Category	Operation	Classification
50	Injection	Writes into the memory of another running process	-
50	Injection	Modifies control flow of another process	-
40	Process	Creates process	-
40	Process	Reads from memory of another process	-
40	Device	Monitors keyboard input	Keylogger
40	Network	Downloads data	Downloader
50	Network	Performs DNS request	-
50	Persistence	Installs system startup script or application	-
50	Hide Tracks	Writes an unusually large amount of data to the registry	-
50	Network	Associated with known malicious/suspicious URLs	-

Figure 4 – VMRay Analyzer reporting a malicious file that installs a system startup script for persistence

The malware will only proceed to the second stage, i.e. downloading the main payload, if it determines it is in the expected target environment. This is related to the scenario where the malware detects that the environment is most likely an artificial analysis environment. Examples include:

- Checking the network usage statistics of the system
- Checking the 'Recently used documents' folder since real systems usually have many files stored here
- Checking the number of running processes and only continue if the number exceeds a threshold.

APPROACH 3: EVASION BY EXPLOITING SANDBOX WEAKNESSES

Explicitly searching for the existence of a sandbox could raise suspicion. Advanced malware, therefore, exploits weaknesses in the underlying sandbox technology to perform operations in the sandbox's blind spot. By exploiting these blind spots, malware does not have to worry about being detected even if it is being executed in a sandboxed system.

There are many evasion techniques under this

approach and in this section we look at two of the most widely used ones.

TECHNIQUE 1: BLINDING THE SANDBOX

Most sandboxes do in-guest-monitoring, i.e. they place code, processes, hooks etc. inside the analysis environments. If these modifications are undone or circumvented, the sandbox is blinded – in other words, visibility into the analyze environment is lost. This blinding can take the following forms:

- **Hook Removal:** Hooks can be removed by restoring the original instruction or data.
- **Hook circumvention:** Hooks can be circumvented by using direct system calls instead of APIs, calling private functions (which are not hooked), or performing unaligned function calls (skipping the "hook code").
- **System file replacement:** Hooks usually reside in the system files that are mapped into memory. Malware can unmap these files and reload them. The new loaded file version is then "unhooked".
- **Kernel code:** Many sandboxes are not capable of monitoring kernel code or the boot process of a system.
- **Obscure file formats:** Powershell, .hta, .dzip are just some examples of file formats that may slip by and simply fail to execute in a sandboxed environment.
- **Unsupported technology:** Examples of technologies that are not supported by some sandboxes include COM, Ruby, ActiveX and JAVA. Their usage can help evade these sandboxes.
- **Operating system reboots:** The idea here is to exploit the fact that some sandboxes cannot survive reboots. Some sandboxes try to emulate a reboot by re-logging in the user, however this can be detected and not all triggers of are boot are executed.

TECHNIQUE 2: BLINDING THE ECOSYSTEM

By simply overwhelming the target analysis environment, malware can also avoid analysis with this crude but sometimes effective approach. For example, some sandboxes only support files up to a certain size (10 MB). Others don't support multiple compression layers. By exploiting these shortcomings, malware can easily evade sandboxes.

CONCLUSION

Of the three approaches, evasion by using time, event or environment based triggers is the least sensitive to the underlying sandbox analysis technology. As analysis technology improves, environmental triggers will become increasingly important to malware authors.

Hence, it is critical for incident responders and analysts to ensure they are using target analysis environments that accurately replicate in every detail the actual desktop and server environments they are protecting. Furthermore, it is important to have pseudo-random attributes as part of the target analysis environment.

Generic sandboxes running identical standard target environments are no longer sufficient. Further, the analysis environment needs to be able to detect environment queries and identify hidden code branches.

To ensure that malware cannot evade analysis, a sandbox analysis environment should:

- Not modify the target environment.

In common sandbox analysis methods like hooking, the presence of a hook (the injected user-mode or kernel-level driver that monitors and intercepts API calls and other malware activity) gives malware the

opportunity to evade detection or disable the analysis.

- Run gold images as target analysis environments. For efficiency and convenience, many sandboxes have a 'one size fits all' approach. A single type of target environment is used for all analysis. A better approach is to use the actual gold images (that is, the standard server OS and application configurations that your enterprise uses) as the target environment. That way, you can be assured that any malware that is targeting your enterprise and could run on your desktops or servers will also run in the analysis environment.

- Monitor all malware-related activity, regardless of application or format. Some sandboxes, particularly those using a hooking-based approach, take shortcuts for the sake of efficiency in determining what activity is monitored. This can leave blind spots.

REFERENCES

- **VMWare Knowledge Base:** How to determine if software is running in a VMware virtual machine
- **Black Hat documentation:** Comprehensive Virtual Appliance Detection: Kang Li and Xiaoning Li
- **Forcepoint Security Lab blogs:** Locky returned with a new Anti-VM trick
- **Exploit Database Papers:** Breaking the Sandbox: Sudeep Singh
- **Broken Browser blog:** Detecting analysts before installing the malware
- **Symantec Advanced Threat Research:** Attacks on Virtual Machine Emulators: Peter Ferrie
- **The invisible things Lab's blog:** Introducing Blue Pill: Joanna Rutkowska
- **Pafish:** <https://github.com/a0rtega/pafish>
- **VMRay Blog:** <https://www.vmrays.com/blog>