2017 Volume 6 Evasive Malware Tricks How Malware Evades Detection by Sandboxes

Author: Clemens Kolbitsch
Date Published: 1 November 2017
español

Sandboxes are widely used to detect malware. They provide a temporary, isolated and secure environment to observe if a suspicious file attempts anything malicious. Of course, criminals are well aware of sandboxes and have created a wide range of techniques to detect if there is a malicious file in a sandbox. If the malware detects a sandbox, it will not execute its true malicious behavior and, therefore, appears to be another benign file. If all goes well from the criminal's point of view, the sandbox then will release the file, deliver it to its intended user and the malware can launch the attack against the real user's environment.

It is a cat-and-mouse game where sandbox vendors add new techniques to detect malware, and criminals develop creative ways to evade detection and respond to the new detection techniques added to the sandbox.

This article describes a representative sample of the techniques criminals use to evade detection, including the most recently developed methods.[1] This is not meant to be comprehensive, especially considering that new evasion techniques are continually being created. With every sandbox revision, criminals respond with a new evasion technique.

This article provides specific examples of three types of evasion techniques:

- **User behavior-based evasion**—Used to detect user actions that indicate the presence of a real user or inaction that indicates a sandbox. Examples of user behavior-based evasion include using Application.RecentFiles.Count and triggering macro code on close.

- **Virtual machine (VM)-based evasion**—Used to detect artifacts that are indicative of a VM-based sandbox. Examples of VM-based evasion include looking for Zone:Identifier and Windows Management Instrumentation (WMI) based evasions.
- **Timing-based evasion**—Used to evade sandboxes by delaying execution of malicious behavior or detecting sandbox timing artifacts. Examples of timing-based evasion include using delay application programming interfaces (APIs), sleep patching and time bombs.

User Behavior—Based Evasion Examples

Criminals deploy a range of techniques to detect user activity that, they assume, would not be present in a sandbox. Two of the most recent examples of this are using Application.RecentFiles. Count and triggering malicious code when a document is closed.

Use Application.RecentFiles.Count

A recent Dridex malware dropper (malware that is designed to subsequently install additional malware) was distributed as a document file containing macros. As background, Dridex is known as Bugat and Cridex (a form of malware specializing in stealing bank credentials via a system that utilizes macros from Microsoft Word). The macros use Application. RecentFiles.Count to check how many files have been accessed recently. A low count suggests that there is not a person using the machine and, therefore, the machine is more likely to be a sandbox.

Trigger Macro Code on Close

Early sandboxes did little to emulate user activity beyond opening a file inside Microsoft Office. As a result, only code registered for the Document_Open event would be triggered within the sandbox. However, real users typically interact with a document much more. They scroll as they read, and once they are done, they close the document. This discrepancy between a real user's behavior and a sandbox can be observed, and malware now often triggers its code via the Document_Close event, meaning it will only execute the code once the document is closed.

VM-Based Evasion Examples

In addition to looking for user activity, criminals program their malware to detect when it is running in a virtual machine and, therefore, likely is a sandbox. As with user activity, there is a long list of techniques criminals use, the most recently detected examples of which are described here.

Look for Zone:Identifier

When a file is downloaded from the Internet onto a computer running Microsoft Windows, the operating system adds an alternate data stream (ADS) to the file to store Zone:Identifier metadata. This metadata includes information about the file, such as information about the URL from which the file was downloaded, and Windows uses it to show appropriate warning messages to the user before opening potentially untrusted content.

On the other hand, when a file is copied into a sandbox for analysis, this Zone:Identifier metadata is usually not present, as the sandbox cannot know where the file originated. Malware will check for this discrepancy. The presence of the Zone:Identifier ADS hints at a real user machine. If it is not found, the malware concludes that it is in a sandbox.

WMI-Based Evasions

The WMI interface allows Microsoft Windows machines and any service running on them to query information about running processes, available services, hardware (e.g., disk) information and more. Typically, system administrators use WMI to automate tasks.

At the very least, sandboxes have to monitor the primary subject, i.e., the program that is to be executed, and the processes with which it interacts. Interactions can be as simple as one program starting another or injecting new code into a target process. WMI is simply another type of inter-process communication (IPC), but it uses a more complicated client-server model. More precisely, it uses advanced local procedure calls (ALPC) to send queries to be executed in the context of system server processes.

If a sandbox is not able to intercept this type of communication, it will miss the activities performed by malware using WMI. Examples of malware using WMI to evade sandboxes include:

- **Checking cores count**—Due to resource constraints, sandboxes attribute the minimum required central processing unit (CPU) cores to a VM, typically just one, so they can run in parallel on as many VMs on a server as possible. However, most modern computers have multiple CPU cores. Malware will execute a WMI query to fetch the cores count, and if the value is one, it concludes that it is running inside a sandbox.
- **Checking disk space and physical memory**—Just like the case for CPU cores, VMs are typically allocated a limited amount of disk space and physical memory. To detect if it is running on a VM, malware checks if the total disk space of the drive is low, such as below 80 GB. Similarly, it checks to see if there is a small amount of physical memory, such as less than 1 GB of RAM. These configurations are not typically found on end user machines.

Without the ability to see this type of IPC, a sandbox is unable to intercept (and manipulate) the data returned by the server process. Thus, malware finds the limited hardware resources and detects the sandbox.

BIOS Info

Basic Input/Output System (BIOS) information for VMs and emulators is different from BIOS information for a real system, and it often contains strings indicative of VMs. Malware can create a list of strings found in BIOS information for VMs and can check if the current system BIOS information contains those strings. If so, malware can be fairly certain that it is running in a VM.

Geolocation Blacklisting

The Internet offers various services that allow a user to request geolocation data based on the client's IP address. Maxmind is one such service, and malware can query this service to get information about the system on which it is running. One piece of available information is the company to which the IP is assigned. Malware compares this data to a list of known vendors, e.g., security companies. A match will indicate that it is executing inside a sandbox.

Check Username

Malware also fingerprints the sandbox using the name of the logged-in user. This trick works because some vendors do not randomize the Windows user under which the analysis is run. The malware simply checks the username against a list of well-known usernames attributed to sandboxes. For example, older versions of two well-known public sandboxes, Hybrid Analysis and Malwr. com, used to have fixed usernames, KR3T and

PSPUBWS, respectively. This makes it easy for malware to detect these sandboxes based on the name of the current user.

Using Specific Instructions

Modern virtualization technologies support instructions that will unconditionally provoke a "VM Exit" into the hypervisor (a system that creates and runs VMs). This allows a VM to modify how the instruction triggering the VM Exit behaves, similar to an interrupt handler. However, this interrupt introduces a discrepancy in the execution time: When executed on a real machine, such instructions are faster than when they are executed inside the hypervisor managing VMs. Malware can use this discrepancy to detect the hypervisor, thereby tipping it off that it is running inside a VM. For example, it can measure the execution time of the CPUID instruction and compare it to the expected execution time of this instruction on a real machine.

Timing-Based Evasion Examples

A final category of evasion includes techniques that use a variety of timing mechanisms. Recently detected examples include using delay APIs, sleep patching and time bombs.

Using Delay APIs

Some sandboxes are programmed to simply wait and watch for a period of time, and if a file does not do anything malicious, it will release the file. To avoid this, malware uses the Sleep and NtDelayExecution APIs available in Windows. Malware calls these functions to sleep for a period of time to outwait the sandbox.

Sleep Patching

Sandboxes will patch the sleep function to try to outmaneuver malware that uses time delays. In response, malware will check to see if time was accelerated. Malware will get the timestamp, go to sleep and then again get the timestamp when it wakes up. The time difference between the timestamps should be the same duration as the amount of time the malware was programmed to sleep. If not, then the malware knows it is running in an environment that is patching the sleep function, which would only happen in a sandbox.

Time Bombs

Another way that malware tries to outwit sandboxes is to include code that will only run on a specific date sometime in the future—criminals can be very patient—especially for targeted attacks. The goal is simply to outwait any timing delays introduced by a sandbox.

Recommended Techniques for Detecting Malware

To be effective, security technologies must be able to detect malware that uses these and many other techniques to avoid detection, including new evasion strategies that criminals continue to develop in response to ever-improving security systems.

The good news is that while evasive malware poses a challenge to traditional sandboxes, modern analysis sandboxes are built on a technique called full system emulation. The key difference between sandboxes based on virtual machines and those based on code emulators is that VMs typically do not fully virtualize the CPU used to run malware code. Instead, it passes most instructions through to the underlying hardware-supported hypervisor for execution.

A code emulator, on the other hand, directly handles each instruction executed within the analysis system and is thus able to tamper with the execution in any way the system wants to. This is done in a way that is completely transparent and invisible to the malware program under analysis.

For example, a full system emulator can tamper with the outcome of string comparison instructions (e.g., when used to compare the username of the system) and force execution down a path that reveals a program's true intent. Similarly, it can detect when a program is executing instructions that allow fingerprinting the hardware configuration and manipulate the effect of this code in a way to trigger additional behavior, helping it to correctly classify malware.

Even more, using full system emulation gives the sandbox complete visibility into the inner workings of programs running inside the analysis sandbox. That is, instead of only observing how a program interacts with the operating system (e.g., via system calls), a code emulator can also track data that are processed by the instructions making up the malware program. As a result, the sandbox can not only track what type of data are read from the operating system, but also how they are used, to what values they are compared (e.g., in code fingerprinting the system), to where the data are sent (when leaking confidential data) and much more.

Last, but not least, by having instruction-level visibility into the programs under analysis, a code emulator can also reason about code paths that the malware program did not execute in a particular analysis run. For example, the system can see what other potential behavior may be lurking in the malware that was not triggered during the dynamic analysis, giving the sandbox even more information for classifying a piece of malware.

Conclusion

Criminals are motivated, creative and persistent. For every sandbox enhancement used to detect evasive malware, criminals will develop a technique to avoid being detected and often use multiple techniques in combination to improve their success with detecting a sandbox. Security companies must offer, and their customers must implement, advanced malware detection technologies that are effective regardless of who has made the most recent move—the cat or the mouse.

Endnotes

[1] Lastline, *An Introduction to Advanced Malware and How It Avoids Detection*, 2017

Clemens Kolbitsch
Is leading the antimalware group at Lastline and works on various projects related to analysis and detection. As security researcher and lead developer of Anubis, he has gained profound expertise in analyzing current, malicious code found in the wild. He has observed various trends in the malware community and successfully published peer-reviewed research papers. In the past, he also investigated offensive technologies, presenting results at conferences such as BlackHat.