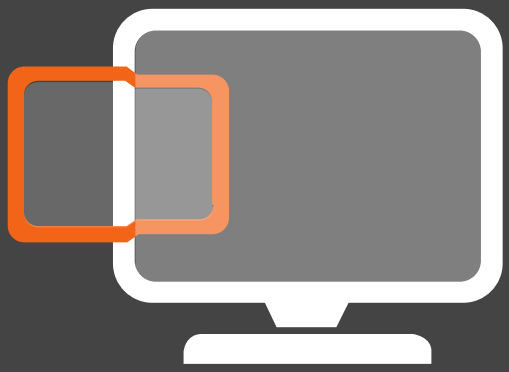
2017



The Rise and fall of the Sandbox

The Rise and Fall of the Sandbox

Antivirus software initially relied heavily on signatures to identify malware and other object based threats. Indeed, even today's current AV products still primarily use a signature engine for detection. Signatures were and are determined when a malware (or malicious file object) arrives in the hands of an antivirus firm, and is analyzed by malware researchers or by dynamic analysis systems. Once a file is determined to be a malicious, a signature (typically an MD5 or SHA 256 hash) of the file is computed and added to the antivirus software's database of known bad files.

This method of detection works well when the malicious file is known ahead of time and appears in the same [known] form on the infected machine, but as antivirus software became commonplace, malware authors began to write "polymorphic" or "metamorphic" viruses, which encrypt parts of themselves or otherwise modify themselves as a method of disguise, so as to not match the virus signatures in the antivirus software's database.

Because metamorphic files cannot be reliably detected with a simple signature based approach, it became necessary to devise a new method of detection: the sandbox.

Instead of relying on pre-defined signatures, sandbox based detection executes a program in a virtual environment, logging what actions the program performs. Depending on the actions logged, the sandbox can determine if the program is malicious or not. By 2006, this technique proved to be more effective than signature based detection and spawned multiple sandbox based products from a variety of companies.

While sandbox based products did provide value for a short period of time, today's threats easily evade the technology. What follows is a description of the difficulties sandbox makers face when trying to design a detection system and clues about what to watch out for if this is your preferred method of detection.

Sandbox Limitations

Profile Mismatch

This is perhaps the most difficult problem for sandbox makers to overcome. To be effective the sandbox environment must precisely represent every end point variant on the network. This means that the sandbox provider must update the sandbox product to support each operating system and application variant – no easy task. In fact, it's almost impossible to maintain a single software profile inside a virtual machine (sandbox) that matches the configuration of all deployed devices on a corporate network. This profile mismatch severely limits the sandboxes' ability to detect an attack. For example, an attack might succeed when run in a sandbox that has vulnerable software or OS installed but fail on the actual host with a patched version of the software or OS – causing False Positives. Or, an attack might fail when run in a sandbox that has the most recent patches installed, but succeed on the actual machine that has not yet had patches applied – in this case missing a threat, since the sandbox will fail to alert.
Similarly, if a particular software or library is not installed or missing inside a sandbox but present on a real machine then an exploit targeting that software or library will succeed on the real machines but will fail inside the sandbox.

Encryption

If the object is delivered via an encrypted network stream (HTTPS/TLS/SSL) the sandbox cannot extract the object and so cannot test it. Modern shell code and malware may download subsequent binary payloads using customized encryption making it impossible to acquire the object even if the network is designed to "man-in-the-middle" decrypt and re-encrypt HTTPS/TLS/SSL network flows.

Limited Detection Coverage

Since the sandbox "observes" what an object does, it is only useful for malware delivered as a payload object such as an EXE, DOC or PDF file. Callbacks, Data Exfiltration, Credential Phishing attacks (Fake Login Page etc.), Java-script exploits, or attacks embedded inside a container such as a SWF or applet cannot be detected using a sandbox because the sandbox technology cannot acquire the object.

Polymorphic Rootkits

Most malware developed prior to 2008 had constant rootkits that would produce a fixed set of OS changes (registry, file system, memory, etc.) making it relatively easy for the sandbox based system to do its job so long as it's configuration properly matched the target machine.

With static rootkit variants, it is easy to manually develop sandbox rules to catch a wide range of attacks, however as the malware authors realized that hard coded rootkit logic (inside the actual binary) is easy to detect, they began designing polymorphic variants that generated rootkits "on-the-fly". These polymorphic variants easily bypass static sandbox rules. To make matters worse, the practice of using dynamically generated rootkits gave birth to the so called FUD (Fully Undetectable) tools, essentially allowing even unskilled programmers to pair their malware with a rootkit that cannot be detected by either signature or sandbox based technology.

FUD tools can attach a dynamic rootkit to any binary without changing the underlying source code – which forced sandbox vendors to write thousands of sandbox rules just to catch even a small subset of malware. Essentially sandbox rules became like just signatures, requiring continual updates from an army of human researchers just to keep up with the constantly changing flow of polymorphic malware.

Limited Operating System Support

A sandbox is not a universal detection technology applicable to any platform or OS. As described above, it must be expressly designed to emulate a target platform. For example, a sandbox designed to emulate Windows XP is different from a sandbox designed to emulate Windows 7, which is different from a sandbox designed to emulate iOS, etc.

Out of practical business need most sandbox vendors focus on popular operating systems (e.g. Windows, OSX and Linux), but today's heterogeneous networks are composed of much more than these popular systems. Consider Smart TVs, printers, medical devices, mobile devices, IoT devices, SCADA systems, and the variations introduced by the accelerating BYOD trend.

There is no practical way for sandbox providers to create the entire range of possible variants, and so, a large portion of the attack surface goes unprotected.

Latency

Detecting a malicious object using a sandbox is much more time consuming than a simple signature check and cannot be accomplished in real-time. Because of the amount of time required for a sandbox to execute its detection rules, a malicious binary or exploit requested by a user endpoint is very likely to be received by the requesting endpoint before the sandbox completes its detection sequence. This is true even if a network based sandbox is running in 'inline blocking' mode. Because it is unacceptable to delay network traffic for several minutes, initial inspection of objects is limited to signature based detection.

Sandbox Evasion

In addition to the systemic problems listed above, modern malware and exploits try to detect the presence of a sandbox environment at the very beginning of their code execution. If they detect that they are running inside a sandbox, the execution is aborted and the malicious payload is never executed, making the sandbox believe that it's a benign object.

Sandbox evasion can be divided into several categories:

Emulator Detection

Emulator or virtual machine detection code looks for Easter eggs left by particular virtual machine implementations such as virtual hardware profiles, virtual machine tools, file system references and registry keys. The presence of any such finger print immediately terminates the program.

Hypervisor Detection

Sandbox virtual machines use virtualized hardware capabilities and some form of kernel hypervisor. There are three popular hypervisors currently used by sandbox vendors: KVM, XEN, and MS Hypervisors. Malware often use x86 instruction anomalies to detect the presence of a hypervisor. Many x86 instructions run perfectly on a bare metal system but are mapped to 'NOP' inside a hypervisor. Similarly a lot x86 instructions behave differently when executed through hypervisors. If such variations are detected, the malware aborts any malicious activity.

Timing Attacks

Timing attacks take advantage of the fact that a sandbox can only run an object for a short period of time. If malware can pause itself for just a few minutes, the sandbox will consider the object to be benign and recycle its VM instance to move on to the next object in the queue.

Execution suspension is achieved using sleep APIs or by executing benign tasks such as repeatedly executing large 'while' or 'for' loops to delay execution.

Lack of Real Environment

One fundamental problem with sandboxes is that they are an artificial environment. Unlike the real machines that are operated by a human and are connected live with the internet, sandboxes have no true user interaction and no real network activity. Modern malware detects the presences of keyboard and mouse movement and often provide some kind of interaction challenge like a dialog box before executing malicious code. Because no human is sitting and operating at the screen, these challenges go unanswered, allowing the malware to detect that it is running inside a fake environment.

A New Approach

To resolve the problems described above, the SlashNext Active Cyber Defense System uses neither signatures nor sandboxes. Instead, artificial intelligence algorithms extract a complex set of “features” from the entire infection sequence to determine if a flow is malicious or not. Because these features are not solely dependent on the actual object (or payload), we inherently do not suffer from the same limitations as signature or sandbox based systems – making our detection capabilities substantially more difficult to evade. Additionally, because the majority of our detection logic resides in the cloud and we have strict control of every network that routes traffic to our cloud, there is no way for a cyber-criminal to devise a test suite that checks whether our cloud will discover his malicious code or not.

To learn more or to request your free trial visit www.slashnext.com