Malware Analysis Sandbox Testing Methodology

Zoltan Balazs
MRG Effitas
Budapest, Hungary
zoltan.balazs@mrg-effitas.com

Abstract— **Manual processing of hundreds of thousands of malware samples became impossible years ago. Sandboxes are used to automate the analysis of malware samples to gather information about the dynamic behaviour of the malware. Some malware samples use known techniques to detect when it runs in a sandbox, but most of these sandbox-aware techniques can be easily detected and thus flagged as malicious.**

New approaches were invented to detect these sandboxes. A tool was developed, which can collect interesting information from these sandboxes to create statistics about how the current technologies work. After analysing these results a practical approach will be presented in order to detect sandboxes. The considered use cases cannot be easily flagged as malicious.

Some sandboxes do not support network connectivity under more restricted configurations in order to block data extraction. But with some DNS kung-fu the information can be extracted from these appliances as well.

Keywords — Malware Analysis Sandbox, Anti sandboxing

I. INTRODUCTION

The cat and mouse game between network attackers and network defenders has been changed a lot since the deployment of endpoint protection systems and traditional network intrusion detection systems, firewalls, mail and web proxies. The attackers have access to easy-to-use tools and services to bypass the conventional protection of enterprises. Firewalls are bypassed by HTTP based, connect-back C&C servers, proxy authentication is bypassed by malware calling Windows API calls which authenticate with the proxy. IDS is bypassed by obfuscation/encryption. Even sometimes web white-lists are bypassed by compromising legitimate websites, and exploiting and controlling the victim through this benign site. Enterprises all over the world are targets of industrial espionage, nation state attackers, or high-profile organized criminals.

History shows that the traditional defence tools are not adequate against targeted attacks. Therefore, the demand for new technologies addressing such problems has increased. One of these new technologies is the so called Breach Detection Systems (BDS).

The most important goal of a Breach Detection System is to identify infected systems in an enterprise where such cause of infection can be a known or unknown malware used during the attack. These systems typically detect the breach itself, allowing administrators to contain the threat and isolate the vulnerable systems as soon as possible. Previous examples showed that these targeted attacks usually last for months or even years. Therefore, an early detection is crucial for an enterprise.

However, it is worth mentioning that while most of these systems are marketed as the solution against targeted attacks, for the attackers these systems are “just another system to bypass”. For that reason, it is expected that attackers will invent new methods to bypass these breach detection systems, and it is in the best interest of the vendors to be aware of potential bypass-strategies and tactics.

There are three main types of analysis regarding new malware samples:

1. static analysis based on the executable layout, signatures of known malware, etc. (used in sandboxes)
2. automated dynamic analysis – runs the sample in a sandbox and detects suspicious behaviour (mostly used in sandboxes)
3. manual analysis

In the contrast to manual analysis, the first and second approach can be (and are) automated, making it both relatively cheap. Moreover, manual analysis is hard, resource-intensive, time-consuming, and thus expensive. When attackers bypass the detection at the first layer and second layer (explained in the presentation [15]) they can stay under the radar for a longer time.

For penetration testers, it is a very common task to generate malware, which can persist during the testing engagement. This malware has to be stealthy on both host and network level. As more and more companies use malware analysis sandboxes, penetration testers have to implement new techniques to avoid detection by these sandboxes. One of the most common techniques is querying the target system to detect whether it is running on e.g. CEO notebook, or in a malware analysis sandbox. If the sandbox is detected, the malware either finishes execution, or changes its behaviour (e.g. financial malware acts like adware to avoid detection [9]). During penetration test engagements, it makes sense to infect the targets with a simple malware. This malware can check the environment, and only infect intended victims with the real malware. By using this technique, the real malware can evade detection for a longer time.

For sandbox developers it is important to know the ways the sandbox can be detected, and either alert on these attempts, or fool the malware and emulate a real environment.

And last but not least, for potential buyers of malware analysis appliances it is important to test how well the sandbox hides its presence.

II. BACKGROUND

Although not many malware use anti-sandbox techniques to evade detection, some of them do. The traditional anti-sandbox techniques include detection of virtualization, running processes, detection of debuggers, detection of hooked functions, injected DLLs, etc. Most of these checks can be easily flagged as malicious. Some advanced techniques are also known; detecting whether sleep functions are emulated, detection of network connectivity, mouse movement, etc.

But the traditional virtualization detection techniques can be detected, and the malware can be blocked.

Also, some traditional sandbox detection techniques (e.g. know sandbox Windows product ID’s) can both be fooled and detected. For example, during the research, the following faked Windows product ID was found: 03DyM 03D 03DyM5G 03DyM

As product IDs contain numeric letters only, this is clearly a faked one. And while the sandbox detection was bypassed, the sample was flagged as malicious because it accessed the Windows Product ID.

Virtualization detection

It is not a trivial task to hide the presence of virtualization from a malicious process [1]. Nevertheless, despite existing multiple available tools focused on hiding virtualization ([2], [3], [4], [5], [14]) how the virtualization can be hidden, there are always new ways to detect it by exploiting common mistakes.

Other anti-sandbox methods

Some anti-sandboxing techniques involve the protection of C&C servers by using IP blacklists and IP range blacklists. One such publicly available project is AVTracker ([6]). This is a common technique used by malware writers and exploit kit operators ([7]). Although this technique is usually very effective, there are some drawbacks, like when the C&C server IP is revealed and it cannot be used on previously unknown sandboxes. Additionally, if the malware analysis sandbox uses the same Internet connection as the regular users, it is not possible to make a distinction between real users and sandboxes based on the IP address only.

Other anti-sandboxing technique is to create resource-intensive tasks, like

- brute-force AES keys ([8])
- multiple memory read-write operations ([10]), which are impossible to log or keep track.

III. THE SOLUTION

New techniques were invented that can detect the presence of a real user. By using DNS tunnelling techniques, or the report the tool can exfiltrate information from otherwise closed malware analysis appliances. These sandboxes usually try to obtain the IP of the domains related to the malware network activity; therefore, it is possible to leak out information from these closed sandboxes. For instance, if the DNS server for myhostname.com is controlled by a given administrator, and the malware performs a query to request the IP address for the “microsoftofficeisinstalled.myhostname.com” domain, it would, as consequence, leak information, i.e., Microsoft Office is installed on the environment where it was running.

Another possible way to extract information from the malware analysis sandbox is to read the report created by the sandbox and check the domain names the malware tried to contact to. For example information about the sandbox can be hidden in the filenames the tool creates, and these filenames are usually included in the report. The report can either be emailed to the attacker (typical for public sandboxes), or downloaded by someone who has access to the private malware analysis sandbox (not typical for low-budget attackers). If the domains contacted information is not available, the tool can create new files, and hide the information about the sandbox in the filenames.

The tool - sandbox_tester - will collect all of the important available information from the sandboxes. Based on this data, statistics were created which are the best parameters to check for sandboxes. Based on this data interpretation, most effective techniques can be implemented in a malware and thus it is highly possible that the malware can evade even a previously unknown sandbox.

Implementation of sandbox awareness

APT attackers use so-called validator style malware. This validator checks the environment, and only drops further (advanced) malware on the machine, when it is a validated target (and not a sandbox) (see [11]).

There are three layers where the sandbox-awareness can be implemented. Each layer has its own advantages and drawbacks. The decision can be implemented in the malware, on the C&C server (automatically) or on the C&C server (manual decision).

The following list summarizes the advantages and disadvantages based on where the decision is made:

Automated, in the malware:

Advantage: No information leak about C&C address

Disadvantage: Not everything can be implemented in this layer (e.g. screenshot analysis), or it is not effective to update the logic on the client (malware) side

Automated, on the C&C server:

Advantage: Almost every check can be implemented (e.g. IP/network based analysis)

Disadvantage: C&C server information leaked

Manually analyse results at the C&C server:
Advantage: Powerful (e.g. analyses desktop screenshot)
Disadvantage: Expensive

These validator-style malware samples follow the same logic of sandbox detection as mine – as the best approach is to implement all three layers, and terminate malware execution at the first detection.

The unsolvable problems

There are at least two problems which makes hiding the presence of the sandbox detection hard.

- The first problem is whether sleep calls are simulated or not. If sleep calls are simulated, it can be detected via two threads. The first thread makes some calculations while the second thread sleeps. In a normal environment the sleeping thread should finish execution later. If sleep is emulated, the sleeping thread will finish the execution sooner – and the simulation will be detected. If sleep-calls are not emulated, the malware can sleep for - e.g. - 30 minutes (or more hours), before starting any activity ([13]). In theory this method can be defeated with continuous sandboxing, but it complicates the malware analysis - e.g. - same malware using the same mutexes.

- The second problem is the network connection. Usually, when malware is dropped on normal targets, the target has Internet connection (either direct, or at least via web proxy). If the malware analysis sandbox allows HTTP traffic (directly or via proxy), information can be leaked from the sandbox. This information can be used to do a manual decision whether it is a sandbox or not (e.g. multiple screenshots during a whole day). If there is no HTTP connection at all, usually the attackers can decide not to infect the box, because it is either not important, or it is a sandbox. A third option is that sandboxes can emulate the network, but it can be detected as well. For example by downloading a known resource from an innocent website - e.g. favicon.ico, and compare the hash of the file with a known value ([12]).

The ultimate sandbox evasion

The following process can ensure that the real malware is not dropped into a malware analysis sandbox.

1. Drop a small, simple dropper (validator)
2. Dropper phones back to validator C&C (e.g. once daily, when user activity is detected)
3. C&C always answers with a new random string (only one per day per session)
4. New C&C calls should include the latest random string to receive new ones
5. Only drops the real malware when it receives the correct new string for days (or weeks)

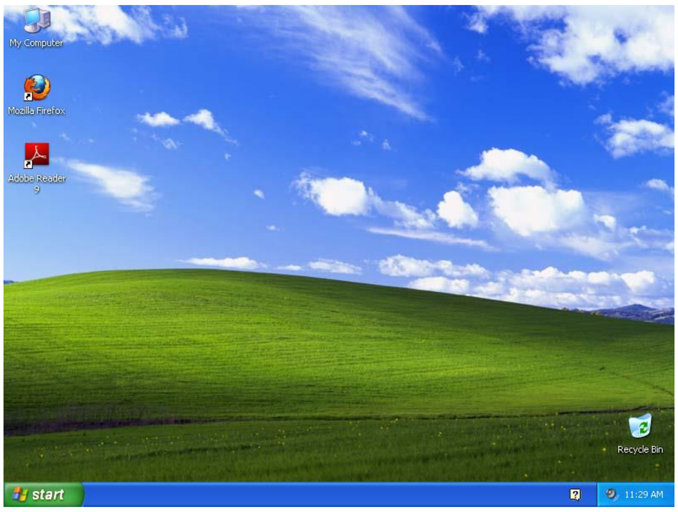
With this technique the automated analysis can be usually evaded (even continuous sandboxing), because most sandboxes don't have the resources to run a sample for weeks, or to save the last state and restore it daily. This can make manual analysis tedious as well - except when the real malware is dropped and the sample is found via forensics analysis.

IV. TEST CASES

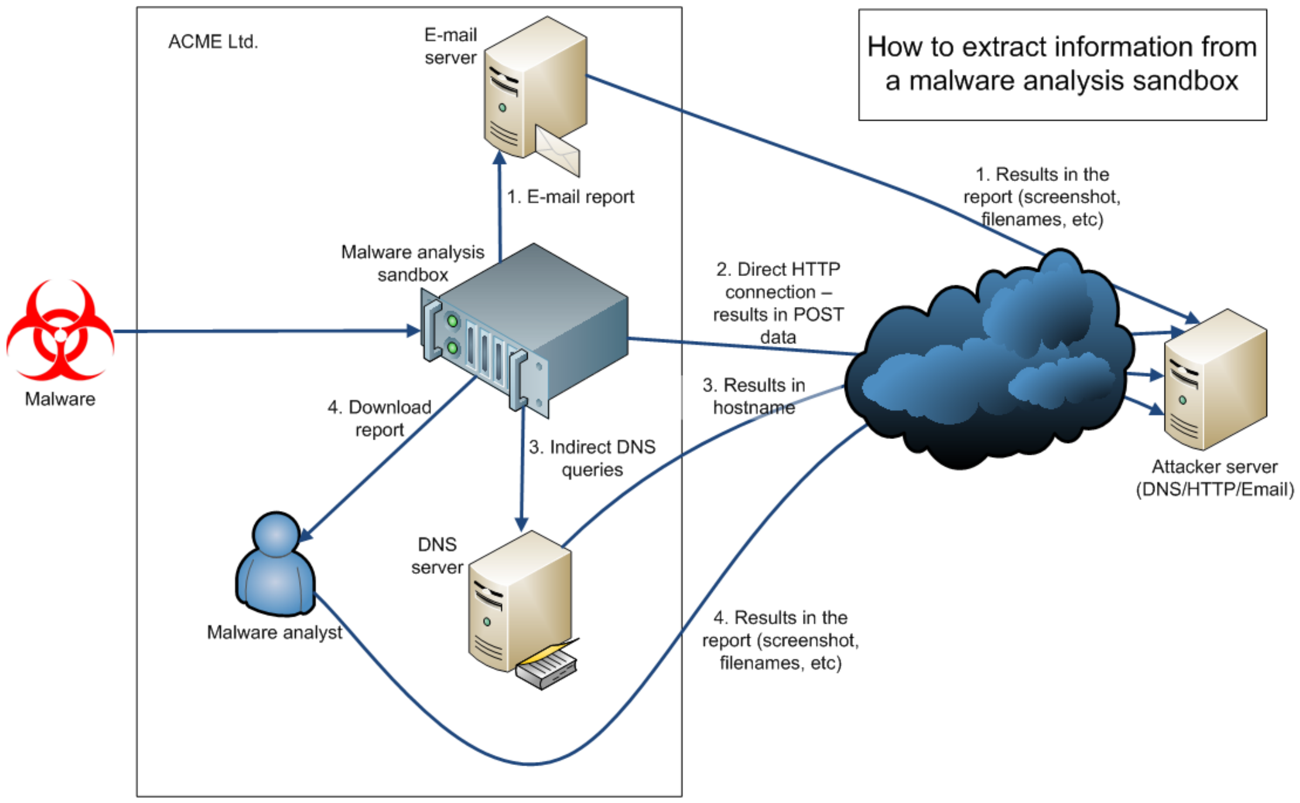
Following is a non-complete list of tests made by the tool:

- Windows product ID – is it a known sandbox product ID? Or a faked one including alphabetic letters?
- Hard Disk Type, layout – is HDD less than 20 GBytes?
- Hardware layout (processor, memory, motherboard, BIOS, network cards) – is it running with 256 Mbyte of memory? Is this a Qemu? Is the MAC address known for Virtualbox?
- CPU architecture – is CPU type Intel XEON while attacking a workstation?
- CPU architecture – 1 processor, 1 core only dedicated to the OS?
- Windows settings (installation date, version, current time) – e.g. is the current time on the OS years behind the real current time?
- System uptime – 2 years, 145 Days, 5 Hours, 3 Minutes, 11 Seconds for a desktop? Or only 1 minute?
- Installed programs – is Debugging Tools for Windows x86 installed?
- Running processes – eg. is cwsandbox.exe running?
- Malware executable name – has the executable renamed to sample01, virus02 or malware03?
- Screen resolution – is it 640x480? Or 800x600?
- Username, computer name, domain – in a targeted attack, attackers might know the Windows domain name, and only allow running if the domain is detected.
- Number of files modified in the previous week in the user profile – less than one? Not a very typical Windows user.
- Types, number of user files (desktop, documents, pictures folder) – missing selfies in the picture folder?
- Registry artifacts of user activity – recently opened files
- Available network shares – no network shares in a corporate environment?
- Ability to click on message window
- Local ports opened – is port 445 closed?
- Hooks installed – is deletefilew hooked?
- Sleep emulated
- HTTP connection is available – if the malware can't access the Internet, it is either a sandbox or an airgapped target
- Mouse movement – how many miles has the mouse moved in the last week?
- Printers installed – everybody has a printer! Except sandboxes …
- USB flash drives used – no USB flash drives used, ever?
- Machine name – is the machine named winxp_maltest?

Screenshot – this is how a normal user desktop looks like



Architectural overview of the proposed solution


ACME Ltd.
E-mail server
How to extract information from a malware analysis sandbox
1. Results in the report (screenshot, filenames, etc)
1. E-mail report
Malware analysis sandbox
2. Direct HTTP connection – results in POST data
Malware
3. Results in hostname
4. Download report
3. Indirect DNS queries
Attacker server (DNS/HTTP/Email)
DNS server
Malware analyst
4. Results in the report (screenshot, filenames, etc)

V. Results

During the tests the following interesting results were found:

68% of the sandboxes used an environment which is identified as virtualized, 18% identified as desktop computer, and 14% as a server (based on “wmic computersystem get model”).

80% of the sandboxes used a screen resolution of 1024*768 or less.

Most sandboxes use usermode tools to detect malicious behaviour, for examplr Python winappdbg 1.4, Debugging Tools for Windows x86, RaptorClient, Dumper.exe, StaticStreamMgr.exe, etc.

Some sandboxes try to evade the detection of the virtualized system by renaming VMWare tools to VEWare tools.

14% of the sandboxes used multi-core architecture to analyse the sample.

26% of the sandboxes used desktop type CPU.

20% of the sandboxes emulated mouse movement.

10% of the sandboxes had 2GByte or more memory.

10% of the sandboxes used flash drive.

None of the sandboxes had printer installed.

8% of the sandboxes had recently modified files in the Documents folder.

VI. Conclusions

A new methodology has been proposed to test the anti anti sandboxing capabilities of malware analysis sandboxes. The methodology uses HTTP, DNS and report based leak (information in the created file names) of the information from the sandbox. Detecting the virtualized environment as a form of anti sandboxing is becoming less and less effective for attackers, but other environment dependant anti sandboxing methods can be used to detect malware analysis sandboxes.

The tool can be downloaded from the following URL:
https://github.com/MRGEffitas/Sandbox_tester

References

[1] N. Rin, EP_XOFF, Virtual Machines Detection Enhanced, 2013
https://github.com/hfiref0x/VMDE

[2] Michael Boman - Making Virtualbox nearly undetectable, 2014 http://blog.michaelboman.org/2014/01/making-virtualbox-nearly-undetectable.html

[3] William Metcalf – Cuckoo building scripts, 2015
https://github.com/wmetcalf/buildcuckoo-trusty

[4] Jurriaan Bremer, VMCloak, a tool for automatically creating and configuring Virtual Machines for Cuckoo Sandbox, 2015
http://jbremer.org/vmcloak2/

[5] VirtualBox Anti-AntiVM, 2014
http://www.kernelmode.info/forum/viewtopic.php?f=11&t=1911

[6] Peter Kleissner, AVTracker
http://avtracker.info/

[7] SpiderLabs Research, Magnitude Exploit Kit Backend Infrastructure Insight - Part II, 2014 https://www.trustwave.com/Resources/SpiderLabs-Blog/Magnitude-Exploit-Kit-Backend-Infrastructure-Insight---Part-II/

[8] Christian Amman, Hyperion: Implementation of a PE-Crypter, Nullsecurity, 2012,
https://github.com/nullsecuritynet/papers/raw/master/nullsec-pe-crypter/nullsec-pe-crypter.pdf

[9] James Wyke, Duping the machine - malware strategies, post sandbox detection, 2015
https://www.virusbtn.com/virusbulletin/archive/2015/01/vb201501-duping

[10] Ben Baker, Alex Chiu, Threat Spotlight: Rombertik – Gazing Past the Smoke, Mirrors, and Trapdoors, 2015
http://blogs.cisco.com/security/talos/rombertik

[11] Kaspersky Labs' Global Research & Analysis Team, Animals in the APT Farm, 2015
https://securelist.com/blog/research/69114/animals-in-the-apt-farm/

[12] Joe Giron, Bypassing FireEye, ToorCon 15
https://www.youtube.com/watch?v=wynvicPjRDk

[13] Th4nat0s, No_Sandboxes
https://github.com/Th4nat0s/No_Sandboxes

[14] hfiref0x, VBoxHardenedLoader
https://github.com/hfiref0x/VBoxHardenedLoader

[15] Zoltan Balazs, Sandbox detection: Leak, abuse, test
https://www.botconf.eu/wp-content/uploads/2015/12/OK-S02-Zoltan-Balazs-Sandbox_mapping_botconf.pdf