An overview of malware analysis - what techniques are available, and how can they be evaded by attackers?

Sam Boyer

Summer 2020

Contents

1 Introduction

In the past 10 years, the total amount of malware in the world has increased by 1650%, with over 350,000 malicious programs discovered on a daily basis (AV-TEST, 2020). Considering the demand for running third-party applications has arguably also increased, the act of determining the safeness of a program is more important now than ever.

Malware analysis has a wide variety of use cases, such as in long-term forensics investigations, or antivirus software that must determine the safety of a file in a short time. Because of this, a range of techniques have been developed, each with their own intended time-frame and level of automation. The techniques are often classified as static or dynamic, depending on whether the code is viewed as binary data or actually executed.

Another point to consider is the urgency of the analysis. If a 'zero-day' cyber-attack has just begun and millions of machines are infected with a new strain of malware, then it's worth the time of a human security researcher (or many) to properly decompose the malware; such a response is described in the first two sections. For the day-to-day classification of millions of potentially malicious files, it's infeasible for humans to analyse each one, so automated methods as described in the last two sections are applied.

2 Manual code reverse-engineering

One of the earliest techniques to understand a program (without access to its source code) is by reading the machine instructions it performs and deducing their overall effect. While it's the cheapest technique here in terms of hardware resources, it's the most demanding in time and experience of a skilled reverse-engineer. If an analyst finds suspicious instructions in a program (things like disguising itself as another program, making strange system calls, etc.), regardless of if the code ever appears to execute, they can say with confidence that the program has the potential to be malicious.

Originally, the only tools available to analysts may have been a hex-editor and instruction set reference book, but nowadays their workload is significantly reduced thanks to disassemblers and decompilers. A disassembler is a program that converts a binary machine code program into a human-readable assembly file, specific to the binary's target architecture. Since assembly code can still be difficult to read for large-scale programs, a decompiler goes a step further and converts an assembly file into a high-level programming lanaguage. An example of these tools combined into one open-source project is Ghidra, released by the US National Security Agency in 2019[1]. Ghidra supports disassembly

[1] though its existence was leaked two years prior.

and decompilation of many architectures, such as x86, ARM, MIPS and JVM bytecode (NSA, 2019).

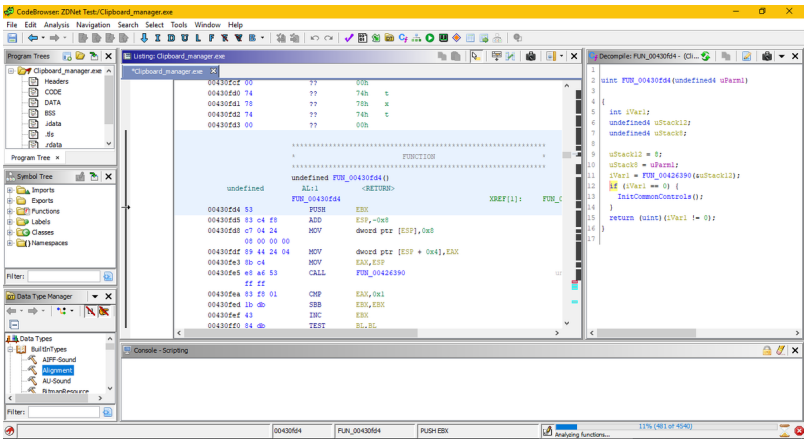

Figure 1: A screenshot of Ghidra's code browser screen, showing a disassembly of the program (center), and a decompilation into C (right). (Cimpanu, 2019)

2.1 Weaknesses: Obfuscated, encrypted & staged payloads

A common defence against reverse-engineering is to make the binary code difficult to understand, which is done by altering the instructions or control flow to less common variations, without changing the outcome of the program. Some innocent actions such as enabling compiler optimisations can disrupt the more rudimentary decompilers, but an attacker who deliberately wants to confuse a reverse-engineer could develop a method to make the code mostly unrecognizable - a good example of which is an obfuscator by Chris Domas that compiles C code into a (very long) series of x86 MOV instructions (Domas, 2015).

Furthermore, an attacker may be able to completely prevent a researcher from decoding a program by encrypting the main part of its code, leaving only a method to decrypt the code after some arbitrary event. Msfvenom is a popular malware payload encoder, which gives options to encrypt the binary (Security Tutorials, 2020), perhaps to evade detection across a network.

Lastly, it's possible that the malicious code isn't even available on the hard drive. Some attacks - especially those that infiltrate a computer via a computer network - use a staging system, where the only purpose of the original code is to download a second set of instructions, store them in memory, then run them. Such a program can only be analysed once the code is extracted from memory, which may be a difficult task if the computer's already infected. Using

a sandbox (as described in the next section) might make the job of finding and extracting malicious code from memory easier.

3 Interactive Behaviour Analysis

A simple way to determine the effects of opening or running a file is to just perform the action - but preferably not on one's personal computer! Ideally we should use a virtual (or isolated) machine so malware cannot cause damage; it would also be beneficial to have monitoring tools on hand such as process viewers, file watchers and packet analysers, to reduce the workload of the analyst. This technique is known as interactive behaviour analysis, where a process is monitored, automatically and manually, while under interaction with a human.

By running a program, this method can sometimes provide answers where static analysis is unable to, for example if a program is too obfuscated to understand, or is completely encrypted until a certain event is triggered. Such programs are much easier to analyse by experimentation and observation rather than reverse engineering. Interactive analysis can also be required where fully-automated automated analysis (See Section 4) falls short, such as when user input is required.

An example of an interactive analysis system is Any.Run, a cloud-based subscription service where researchers can submit files, interact with them in a VM of their choice, and observe many details about its behaviour:

An interesting consequence of having a popular, centralised analysis system is that it becomes a specific evasion objective for attackers (malware should avoid acting maliciously if it's being analysed on the platform), as has recently been achieved (Abrams, 2020a). Such a targeted method is less likely to happen with a custom-made sandbox, but still possible (see 4.1).

3.1 Weaknesses?

Aside from the specific attack against Any.Run, interactive analysis systems still have disadvantages. One drawback compared to manual reverse-engineering is the resources and infrastructure required to build a realistic and monitored sandbox - or the money to purchase a sandbox service. It's a lot larger in scope than a single disassembler program, but still within the reaches of an established research lab.

Additionally, one could argue these environments are only effective in detecting expected behaviours. Network activity, modified files or visual prompts are only observed because we expect modern malware to behave vaguely similar to previous ones. If a program were to do something completely unexpected (perhaps using a zero-day exploit), an interactive analysis may not notice it,

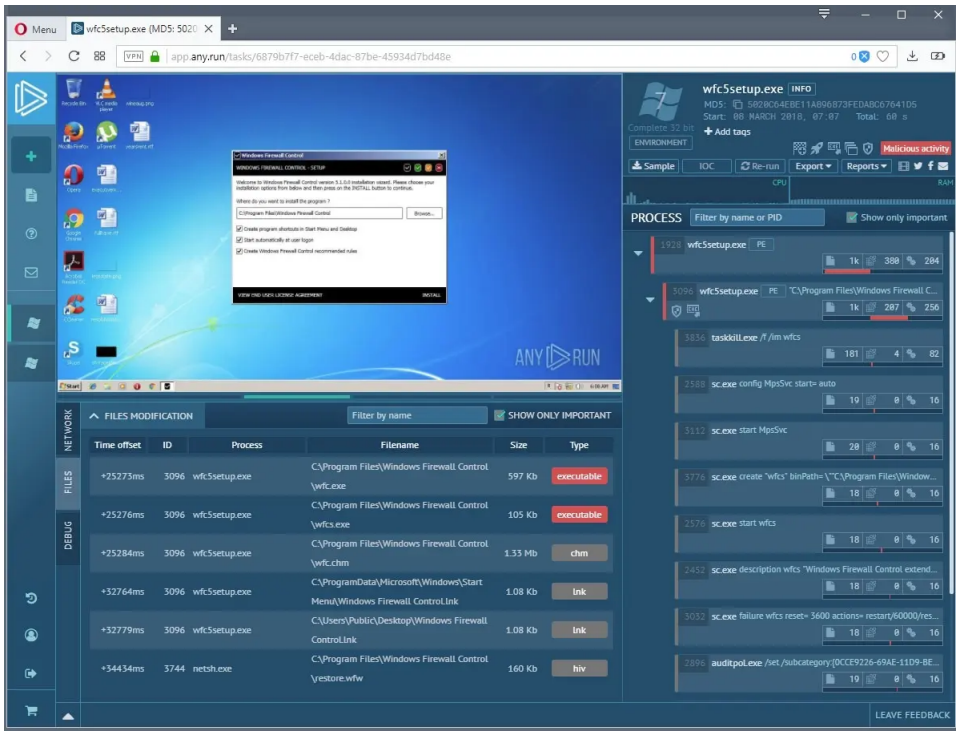

Figure 2: A screenshot of Any.Run's analysis tool, showing an interactive screen capture of the VM (top left), a list of modified files (bottom left), and a tree of processes spawned by the original program (bottom right). (Brinkmann, 2018)

whereas a static analysis of the code would reveal the technique. This argument doesn't hold for all behaviours: ransomware will always lock files in some way, distributed denial-of-service trojans will always send network packets, and so on.

4 Fully Automated Analysis

After learning how to perform a task, it seems the natural next step for computer scientists is to automate it. Automated analysis software grew popular in the early 2010s, largely due to the release of an open-source option in 2011 named Cuckoo Sandbox (Stichting Cuckoo Foundation, 2017). Such software attempts to determine the safety of a program (or office documents opened in their respective programs) by running them in an isolated virtual machine and observing any high-level actions taken - known as signatures - such as network requests made or system files modified.

Based on the actions observed, most sandbox systems return a score, indicat-

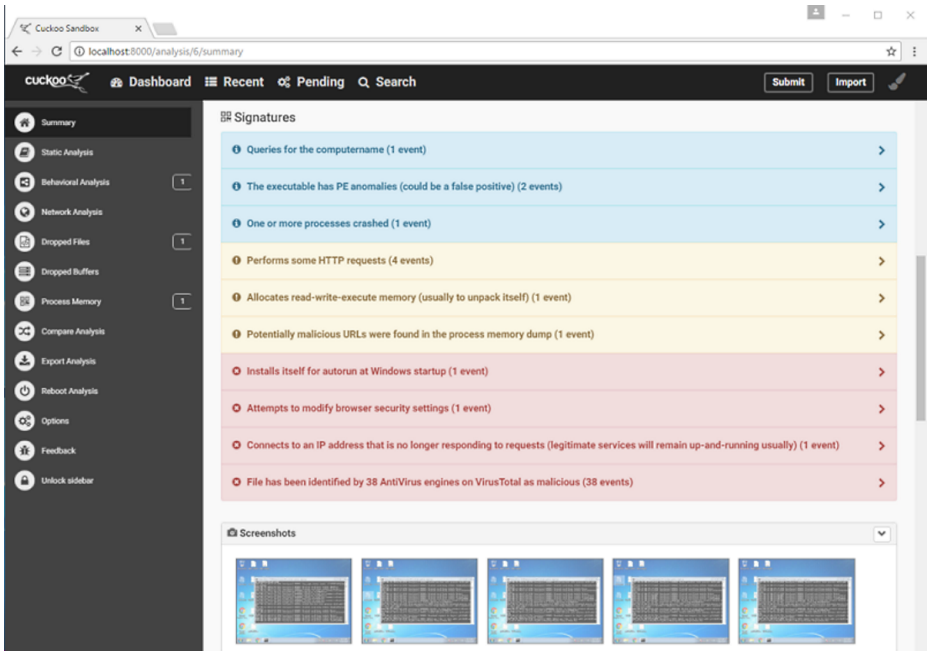

Figure 3: A screenshot of Cuckoo Sandbox's summary page, listing some low, medium, and high importance signatures detected. (Carsula, 2017)

ing how confident it is that the file is malicious. They also report many other data points acquired during the simulation, such as URLs requested, system commands run, and modified files. Such data points could be used by analysts or other systems to mitigate the global impacts of the program: file signatures can be added to anti-virus software databases, and IP addresses identified as command & control servers can be targeted for takedown (an example of which is reported at (TechRadar, 2019)).

4.1 Weakness: VM detection & Analysis Evasion

In early 2009, the Conficker worm spread to a peak of 15 million computers (UPI, 2009). The first analysts to reverse-engineer the malware had difficulty observing its effects in a virtual machine, since it was able to detect it was in a VM, and closed itself (Zdrnja, 2009). The worm's technique was discovered the year previous, so the analysts' work wasn't significantly hindered by the trick - but it was the first in a series of 'VM evasion' techniques to be spotted.

Now that automated sandbox analysis is used in many classification pipelines, it's very common for malware to try and detect a virtual or sandboxed environment, and perform no malicious action if so. When a new technique is invented, it's a very effective method of having a malicious binary be deemed

safe by sandboxes. But once the technique is discovered, it can be added to the actions that sandboxes look out for, and ironically become an effective method of classifying a file as malicious.

Such evasion measures are unlikely to be effective in Interactive Behaviour Analysis; for an analyst to be checking a specific file, they would already suspect it's malicious, and can take measures to make the malware behave properly.

While modern analysis software can defend against most known anti-VM techniques, new methods for identifying a sandbox are frequently being found. For example, it was discovered in late June that some modern strains of malware check the machine's screen resolution - since the VMs used in analysis sandboxes are often left at the default 800x600px resolution, which isn't commonly found in real life (Abrams, 2020b).

The cycle of creating a new anti-VM technique, analysts discovering the technique, and modifying the sandbox software will likely be a game of cat and mouse that continues as long as malware does.

5 Static Properties Analysis

While both effective methods, manual code review and automated sandbox analysis take a long time to complete and require significant resources, which can't be scaled to classify for millions of files required by an antivirus software vendor. We need a method to determine a file's safety in a short time and with little resources. For safety reasons, we may also want to avoid executing the file [2], so our classification will be made purely on the binary data.

Static Properties Analysis finds what we could call 'embedded' and 'generated' properties of a file, and uses to make educated guesses about its safety. Embedded properties include string constants (identified within the binary by their ASCII values and position in the file), file metadata (company names, version numbers etc.), and other details that can fingerprint compiler configuration. Generated properties mostly involve hashes and file sizes.

5.1 Weakness: randomised obfuscation

Unlike manual code-reversing, typical obfuscation or encryption doesn't stall a fingerprint-based approach, because copies of the file would still hash to the same value, and obfuscated/encrypted code and strings would still be suitable fingerprints. However, if a malware creator (or distributor) were to add random data to each copy of a file, they would hash to different values and it would no longer be a suitable detection measure.

[2] Some antivirus programs such as Avast have local sandboxing features, which can be used by a consumer to verify a program's safety (Avast, 2020).

To mitigate this, some malware analysis programs (e.g. Cuckoo and VirusTotal) provide an `ssdeep` hash, which is an implementation of context triggered piecewise hashes (CTPH)(ssdeep Project, 2017). Unlike typical hash functions, CTPH values for similar inputs (defined by `ssdeep` as having 'sequences of identical bytes in the same order') produce hashes that are also (bitwise) similar. This allows analysis software to identify file with small amounts of random data as the same family of programs (with reasonable confidence), and classify them accordingly[3].

However, CTPHs can also be defeated by a stronger randomization of the file; namely reordering or obfuscating instructions at random without damaging the overall effect of the code. This can remove almost all sequences of identical bytes between files, resulting in unrelated CTPHs.

6 Conclusion

Throughout this overview, we have seen how a combination of static and dynamic analysis is most effective to understand the behaviour of malware, in both human or automated contexts. It's important to bear in mind that current signatures and even toolsets may be inadequate to understand the capabilities of sophisticated malware in the future - perhaps even some in the wild right now - so analysts must learn from and adapt to the changing threat landscape.

[3]Interestingly, `ssdeep` was originally designed to detect spam email with random alterations, but was later found to also be effective for machine code.

List of Figures

References

Abrams, L. (2020a). *Malware adds online sandbox detection to evade analysis.* Bleeping Computer. URL: https://www.bleepingcomputer.com/news/security/malware-adds-online-sandbox-detection-to-evade-analysis/ (visited on 20/07/2020).

— (2020b). *TrickBot malware now checks screen resolution to evade analysis.* Bleeping Computer. URL: https://www.bleepingcomputer.com/news/security/trickbot-malware-now-checks-screen-resolution-to-evade-analysis/ (visited on 05/07/2020).

Avast (2020). *Avast Help — Avast Antivirus: Sandbox.* URL: https://help.avast.com/en/av_free/17/securitysandbox.html (visited on 22/07/2017).

Boyer, S. (2020). *Self-produced diagrams.*

Brinkmann, M. (2018). *Interactive Malware Analysis Tool Any.Run launches.* URL: https://www.ghacks.net/2018/03/08/interactive-malware-analysis-tool-any-run-launches/ (visited on 20/07/2017).

Carsula, G. (2017). *Cuckoo Linux Subsystem: Some Love for Windows 10.* URL: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/cuckoo-linux-subsystem-some-love-for-windows-10/ (visited on 12/07/2017).

Cimpanu, C. (2019). *NSA releases Ghidra, a free software reverse engineering toolkit.* ZDNet. URL: https://www.zdnet.com/article/nsa-release-ghidra-a-free-software-reverse-engineering-toolkit/ (visited on 21/07/2020).

Domas, C. (2015). *xoreaxeaxeax/movfuscator: The single instruction C compiler.* URL: https://github.com/xoreaxeaxeax/movfuscator (visited on 22/07/2017).

NSA (2019). *Frequently asked questions · NationalSecurityAgency/ghidra Wiki.* NSA. URL: https://github.com/NationalSecurityAgency/ghidra/wiki/Frequently-asked-questions (visited on 21/07/2020).

Oracle (2018). *Text Form of Oracle Critical Patch Update - January 2018 Risk Matrices*. URL: https://www.oracle.com/security-alerts/cpujan2018verbose.html#OVIR (visited on 11/07/2020).

Security Tutorials (2020). *Creating a Payload with Msfvenom*. URL: thttps://securitytutorials.co.uk/creating-a-payload-with-msfvenom/R (visited on 28/09/2020).

ssdeep Project (2017). *ssdeep - Fuzzy hashing program*. URL: https://ssdeep-project.github.io/ssdeep/index.html (visited on 11/07/2020).

Stichting Cuckoo Foundation (2017). *Cuckoo Sandbox - Automated Malware Analysis*. URL: https://cuckoosandbox.org/about (visited on 12/07/2020).

TechRadar (2019). *French police take down global malware botnet*. URL: https://www.techradar.com/news/french-police-take-down-global-malware-botnet (visited on 22/07/2017).

AV-TEST (2020). *Malware Statistics & Trends Report*. URL: https://www.av-test.org/en/statistics/malware/ (visited on 18/07/2020).

UPI (2009). *Virus strikes 15 million PCs*. URL: https://www.upi.com/Top_News/2009/01/26/Virus-strikes-15-million-PCs/19421232924206/?ur3=1 (visited on 12/07/2020).

Zdrnja, G. (2009). *More tricks from Conficker and VM detection*. URL: https://isc.sans.edu/diary/More+tricks+from+Conficker+and+VM+detection/5842 (visited on 12/07/2020).

Bibliography

Zeltser, L. (2015). *Mastering 4 Stages of Malware Analysis*. Bleeping Computer. URL: https://zeltser.com/mastering-4-stages-of-malware-analysis/ (visited on 08/07/2020).