International Journal of Engineering Applied Sciences and Technology, 2021
Vol. 6, Issue 6, ISSN No. 2455-2143, Pages 146-150
Published Online October 2021 in IJEAST (http://www.ijeast.com)

IMPERCEPTIBLE MALWARE: BYPASSING MODERN AV - ENGINES BY AI-ASSISTED CODE

Aviral Srivastava
Student
Amity University Jaipur

Dhyan Thakkar
Student
Nirma University

Pulkit Verma
Student
Amity University Jaipur

Preay Patel
Student
Amity University Jaipur

***Abstract*— In recent times new Antivirus software are using Machine learning to make their detection even more sophisticated. Machine learning, reinforcement learning, and deep learning along with data analysis have made it possible to implement a dynamic analysis procedure to detect any malware. So in this paper, we will be introducing an algorithm by which we will not only be able to bypass signature-based detection by 'rephrasing the code' using CLP, along with the behavioral-based analysis, which are the most prominent methods for the job, but also will be attempting to go around the real-time monitoring and try to be undetected during the forensic investigation by clearing code footprint.**

Keywords—**Malware, Antivirus, Dynamic and Static analysis, Behavioral-based detection, Code language processing, Machine Learning**

I. INTRODUCTION

Earlier when the Anti-malware services used the signature-based detection method the black hat hacker and people with malicious behavior could easily bypass it using some sort of wrapping or binding the malware to another file and using cryptography or steganography. The idea was to change the signature a little bit so that it wouldn't be recognized by the AV engine. But now in the era of AI, we are using neural networks and machine learning to do things easily which were considered complicated. One such thing is Dynamic analysis which is like a behavioral-based detection which we will discuss later in-depth further in this paper. The main challenge is how to make malware not behave like malware so that it could bypass the behavioral-based detection without changing the primary objective.

Since this technique is still under the research and development phase, therefore being a Security analyst and researcher there will not be any better opportunity to find flaws in this system that can be fixed right now before any person with a wrong intent gets this. The main questions we will be answering in this paper are (1) Is it possible to bypass behavioral-based detection (2) What is CLP (code-language-processing) (3) How do Machine learning algorithms still need to be worked on to get ahead of the malicious human-brain.

II. WHY IS THIS PAPER AND WHO IS THIS FOR?

This paper is for the antivirus agencies and the big tech companies who make anti-malware software and tools. The motive is to point out the flaws in the system in an ethical way without performing any real exploit and causing damage. All the tests were done in an observed and private virtual environment. Since almost everyone whether it is an individual or a tech giant or a startup is at risk of cyberattacks and malware attacks because of which they implement these anti-malware services to avoid those, therefore this paper will help the developers to patch the flaws pointed in this paper to make the system more efficient. Also, this is for malware researchers and cybersecurity and machine learning students who are willing to learn about the new technologies and their working and how you can help to find flaws and help the authorities to patch them up.

III. EXISTING WORKS

Currently, Anti-malware companies use signature-based detection which is not that great, and malware/exploit developers can find a way around it. There are even tools available that help in doing the same. This is called Static analysis. To make things better the companies and researchers are working on finding ways to detect malware based on its behavior. There are only two profound ways to classify any object: first is based on its signature and the second is its behavior or characteristics. Now, this technique needs the code to run to observe its behavior, therefore this is also called Dynamic analysis. In Dynamic analysis, we mostly do pattern recognition to achieve behavioral-based analysis.

A. Static Analysis

Signature-based approach: Signature detection [1] is the simplest method and is the most widely used for traditional malware detection. This method constructs a database that contains signatures of all known malware. When analyzing a new programming code, it compares the signature of the analyzed virus with its database, if the matching is found, the analyzed file is considered as a virus. This approach is fast and has a high positive rate, however, the database needs to be updated with new signatures. Although this technique is old, it

International Journal of Engineering Applied Sciences and Technology, 2021
Vol. 6, Issue 6, ISSN No. 2455-2143, Pages 146-150
Published Online October 2021 in IJEAST (http://www.ijeast.com)

was used in the early days of polymorphic detection when an investigator/researcher analyzed the virus manually, one by one, line by line to detect various sequences of programming codes [2]. As the number of viruses has been increasing so fast, this technique quickly becomes time-consuming, expensive, and impractical.

System call analysis: Sung et al. [3] proposed the Static analyzer for malicious executable (SAVE) to detect malware, mostly focusing on polymorphic and metamorphic viruses which run on Windows Operating System. This method works based on the assumption that all malware variants share a common core signature - a combination of several features of the programming code. In their method, two critical steps were involved: First, the Portable Executable (PE) decompressed and passed through a parser, this parser produced a list of Windows API calling sequences. Second, this API sequence will be compared against the signature database; a similarity measure was used to conclude the analyzed file. If the similarity is greater than a certain threshold, the detection is trigged.

Control-flow graph:: Graphs are also used in the static analysis [4] and [5] where a set of control flow graphs (CFG) were constructed and reduced (where possible) and used as a signature database. This method works based on the assumption that the control flow graph of the malware was not modified in most of the mutation engines. Detection is carried out by comparing the sub-GFGs of the malicious file against the signature database to find if any sub-CFG is matched with the database. However, this method does not work when analyzing the metamorphic virus because this virus can change the code itself for each execution or change the branching structures of that flow graph.

Model-checking: This method assumes that systems have a finite state or may be reduced to a finite state by abstraction. Serge Chaumette et al [6] used context-free grammars as viral signatures and a process was designed to extract the simple virus signature. This method was based on two assumptions: First, most mutating engines generate code belonging to a language that is low complexity, that is, belonging to either natural language or context-free language. Second, the mutation engine has to be embedded inside the self-replicating malware, hence it is feasible to extract the grammar of the mutation engine via static analysis. However, this method is very time-consuming. Another study was presented by Gerald R. Thompson and Lori A. Flynn [7], they compared the program's hierarchical structure and mapped this structure to a context-free grammar, normalized the grammar, and finally, they used a fast check for homomorphism between the normalized grammars. This technique is resilient despite polymorphism that reorders instructions, rewrites instructions, inserts instructions, or removes instructions. This approach did not address encrypted files but can be applied after the file is decrypted if the unencrypted virus is suspected to be polymorphic.

Data-flow analysis: This method gathers information about the possible set of values of objects and variables involved in the specimen. Agrawal, Hira, et al. [8] proposed a Malware Abstraction Analysis (MAA) method. They used two stages to derive the semantic signature of a binary instance: First, all functions were analyzed and abstracting away all unnecessary control flow artifacts from their flow graphs. Second, all local, function-level signatures were combined into a single, global signature while abstracting away all call and return specific artifacts. This method is resistant to such large-scale, global transformations.

Machine learning analysis: In recent years, machine learning has gained popularity in many fields including security. Robert Moskovitch et. al. [9] proposed a technique that monitors a small set of features that are sufficient for detecting malware without sacrificing accuracy. The result of the study showed that only using 20 features, the mean detection accuracy was greater than 90 percent, and for specific unknown worms, this accuracy was over 99 percent, while maintaining a low level of false-positive rate. The advantage of machine learning techniques is that they will not only detect a known malware but also act as a database for detecting new malware. Similar studies can also be found in other models such as Naive Bayes, Decision Tree, Neural Network [9]. Although this technique is practical, it may not replace the standard detection methods, rather than act as an add-on feature because machine learning techniques are computational and may not be suitable for end-users.

B. Dynamic Analysis

Dynamic analysis was mainly developed to counter the Polymorphic malwares. Trevor Yann and Oleg Petrovsky [10] proposed architecture to detect polymorphic viruses, this architecture includes three components: (1) an emulator that emulates a selected number of instructions of the computer program, (2) an operational code analyzer that analyzes a plurality of registers/flags accessed during emulated execution of the instructions and (3) a heuristic analyzer that determines a probability that the computer program contains viral code based on a heuristic analysis of register/flag state information supplied by the operational code analyzer.

Polychronakis et al. [11] presented a heuristic detection method that scans network traffic streams for the presence of polymorphic shellcode. This algorithm relied on a fully-blown IA-32 CPU emulator that makes the detector immune to runtime evasion techniques such as self-modifying code. Each incoming request was executed in a virtual environment. Their algorithm focused on identifying the decryption process that takes place during the initial execution steps of a polymorphic

International Journal of Engineering Applied Sciences and Technology, 2021
Vol. 6, Issue 6, ISSN No. 2455-2143, Pages 146-150
Published Online October 2021 in IJEAST (http://www.ijeast.com)

shellcode. The study result showed that the proposed approach is more robust to obfuscation techniques like self-modifications. One limitation of this approach was that it detected only polymorphic shellcodes that decrypt their body before executing their actual payload, it did not capture the shellcode that did not perform any self-modifications.

Antony Rogers [13] proposed an apparatus to detect malicious code that uses calls to an operating system to damage computer systems. This method will be creating an artificial memory region, this region may span one or more components of the operating system. The malicious file will be executed and the method will try to detect whether the executable code attempts to access the artificial memory region. The method may comprise determining an operating system call that the emulated code attempted to access, and monitoring the operating system call to determine whether the code is viral.

Another apparatus was presented by Igor et al. [12] where they patched additional program instructions into an emulator for detecting suspect code. During operation, a first emulator extension was loaded into the emulator then the suspect code was loaded into an emulator buffer within a data space of a com-

puter system. The suspect code was executed in the first emulator extension. During this emulation, the system identifies whether the suspect code is likely to exhibit malicious behavior.

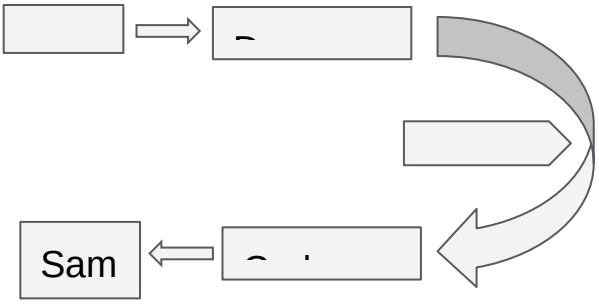
Stepan [14] proposed a method to detect malware by dis-assembling the malicious code dynamically then compiling this code to target the CPU host, the execution file will be executed safely on the host CPU. The code obtained can be used to compare with the original cost. This method increases the analysis speed significantly.

IV. METHODOLOGY

The process is divided into 2 parts: First is Code-language processing or CLP and the second is the execution of the attack itself and the bash script. As we go along we will also keep a note of the detection stages that we bypass. The 4 main stages will be (1) Real-time monitoring (2) SIgnature based detection (static analysis) (3) Behavioral-based detection (Dynamic analysis) and (4) Forensics. Keep in mind that we aren't proposing or writing a full-fledged exploit or malware to bypass any of this. We will just be giving a concept based on the tests which we have performed on a virtual machine. The CLP is where we will work on the code which will allow us to stay undetected during the forensics and bypass the Signature-based detection (Static analysis).

A. Code-Language Processing (CLP)

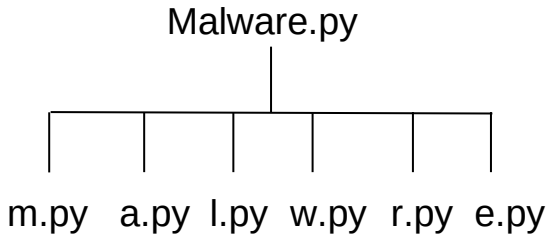
Code-Language Processing is nothing but "rephrasing code". Rephrasing code by hand is easier but can lead to detection with signature. Using Google Translate to translate text from one language to another language and back doesn't always mean the same. but commenting on the code and then using advanced models like GPT-3 to rewrite the functions from a large database helps us in bypassing signature detection as the code is generated with help of AI and thus has a unique signature. The auto commenting of code can be achieved using various models. \cite{surveycodecomment}.



As Deep Neural Network-based algorithms have a high recall and context understanding of comments, these comments can then be fed to a code GAN like GPT using OpenAI Codex or any other prevalent algorithms to get AI-generated code with the same behavior but distorted signature.

B. Uploading malware

Now since we have our morphed malware we will break it into pieces of code and save each part as a different file and name them. Now, why do we do this to avoid being detected by any real-time malware detection service? Since we have broken the malware, each file will be nothing but simply an insufficient piece of code. Now if you observe that these small files will not behave as malware on their own, This will help us to circumvent our main stage which is Behavioural-based detection. Since the files won't be showing any behavior then performing the behavior-based analysis will not show any "Red Flags".



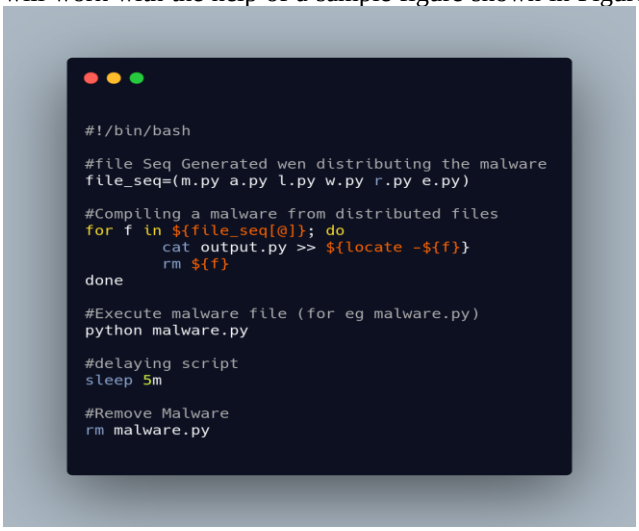
This is the key idea and the main stage where we will be bypassing the Dynamic analysis. Continuing with our process we will upload the small files on the target network or machine. During our tests, we used two virtual machines, windows and Linux and for our test, we simply used a physical flash drive to upload the malware on the target system or network. We can do this by any method like SSH, CURL, TELNET, FTP, or by any

International Journal of Engineering Applied Sciences and Technology, 2021
Vol. 6, Issue 6, ISSN No. 2455-2143, Pages 146-150
Published Online October 2021 in IJEAST (http://www.ijeast.com)

vulnerability found in the network through which we compromised the network, if nothing works we can always use social engineering to get our files on the target system or network. Since we aren't proposing an exploit so we can figure this out in-depth while making a full-fledged exploit for this. Since the files individually won't be anything harmful therefore any real-time time monitoring and detection will be bypassed even if it will scan the files it won't find anything but just an insufficient piece of code. To bypass the Signature-Based Detection we could do it just by simply breaking the complete malware into pieces but to add a layer of assurance we used the CLP. Now as the Behavioral-based analysis will be running it will be observing the behavior of all the files but they won't show any sort of abnormal behavior which will help us be out of the radar. Now since we have our malware on the system undetected by both Static and Dynamic analysis we can proceed further, the next step is to compile and run the malware inside the machine, and for that, we will use a small bash script to do so.

C. The Bash script

The bash script here is very important as it is the one that will be responsible for the actual exploit. As mentioned earlier this is not a full-fledged exploit so, We won't be sharing the script but we will have a look at what it would look like and how it will work with the help of a sample figure shown in Figure 3.



```
#!/bin/bash

#file Seq Generated wen distributing the malware
file_seq=(m.py a.py l.py w.py r.py e.py)

#Compiling a malware from distributed files
for f in ${file_seq[@]}; do
        cat output.py >> ${locate -${f}}
        rm ${f}
done

#Execute malware file (for eg malware.py)
python malware.py

#delaying script
sleep 5m

#Remove Malware
rm malware.py
```

Here we have a sample script which we will be using to run the malware. Taking the same example of Malware.py, the first step will be to find all the files I.E. the parts of malware for example in this case 'm.py, a.py, l.py, w.py, a.py, r.py and e..py, ' Then in the next step, we will Concatenate the content of all the files into our main file "malware.py" in this case and then remove all those files. just to cover the tracks because our final stage is to not get caught during the forensics as well. After that, we will run the recently made final output file which is malware.py here, and give it some time to execute completely, and then after 5 minutes remove it as well. Now even during the forensics if the logs will be analyzed and they will come to know about the malware file even then they won't be able to find it and even if they did anyhow then also that will not be the original malware by just a rephrased version of the original one. By this, we bypassed our final stage which is forensic investigation.

NOTE:- The bash file can be run over the network via remote execution or by any other tool out there; this is not a big deal for any cybersecurity analyst. Also, the bash file here is just an example to demonstrate how the process can take place.

V. IMPACT

The lethality of this is dependent upon the type of malware used. This is just an algorithm by which any malware bypasses the main 4 stages of detection. For example, if the first job of the malware after being executed is to escalate its privileges or turn off the security systems or stop and critical service, or even a simple buffer overflow or ransomware, all these can be harmful to the target.

VI. SUGGESTED IMPROVEMENTS

No perimeter can be completely secured and no AV engine can detect all kinds of malware. This algorithm is developed with a focus on how to bypass the Dynamic analysis and how malwares can get through the behavioral-based analysis while also evading signature-based analysis. Some changes which can be made to counter this kind of attack which have a distributed nature are:

1) Improving the real-time monitoring to analyze the compiled processes when we used the bash script to compile the malware
2) To make the behavioral analysis faster so that it can observe the malware's behavior when it is executed but before it can cause severe damage.
3) To set up a process analyzer which will be helpful in case if the malware is already executed then this can work along with the behavioral analysis to monitor the background process to look for anything harmful.
4) An advanced pattern recognition system implementation can work in the background to find any suspicious patterns in the background processes because even if the behavioral-based analysis is not able to classify the behavior on the file then this system can find patterns in the backgrounds for example if there is ransomware then it will start to encrypt the files so this can be detected in the process monitoring and analysis because every malware needs to create a service or at least need to bind with one to fully function.
5) At last, the first section of our CLP can be used to comment on the code with proper context and read those comments and send them to the codex API to identify the intent of the code.

VII. CONCLUSION

We still cannot depend completely upon the behavioral-based detection method for our defense against malware attacks. Dynamic analysis has made it possible to detect different kinds of malwares like Polymorphic malware or even as advanced as Metamorphic malware. We also saw how we can use CLP which is inspired by Natural Language Processing (NLP) to rephrase the malware which is a new way to bypass the signature-based detection and also make it difficult to reverse engineer the source code. The distributed nature of our attack is still a way to fool both the Static and Dynamic analysis. But these suggested improvements can surely help us in enhancing the abilities of the Dynamic analysis used by the modern AV engines.

VIII. REFERENCES

[1] Griffin, Kent, et al. "Automatic generation of string signatures for malware detection." International workshop on recent advances in intrusion detection. Springer, Berlin, Heidelberg, 2009.

[2] Bondarenko, Yevgeniy, and Pavel Shterlayev. "Polymorphic virus detection technology." Fond in http://www.it.lut.fi/kurssit/05-06/,Ti5318800/assign/Ti5318800_virusdetection.pdf On 5th April (2006).

[3] Sung, Andrew H., et al. "Static analyzer of vicious executables (save)." 20th Annual Computer Security Applications Conference. IEEE, 2004.

[4] Christodorescu, Mihai, and Somesh Jha. Static analysis of executables to detect malicious patterns. Wisconsin Univ-Madison Dept of Computer Sciences, 2006.

[5] Bonfante, Guillaume, Matthieu Kaczmarek, and Jean-Yves Marion. "Control flow graphs as malware signatures." International workshop on the Theory of Computer Viruses. 2007.

[6] Chaumette, Serge, Olivier Ly, and Renaud Tabary. "Automated extraction of polymorphic virus signatures using abstract interpretation." 2011 5th International Conference on Network and System Security. IEEE, 2011.

[7] Thompson, Gerald R., and Lori A. Flynn. "Polymorphic malware detection and identification via context-free grammar homomorphism." Bell Labs Technical Journal 12.3 (2007): 139-147.

[8] Agrawal, Hira, et al. "Detection of global, metamorphic malware variants using control and data flow analysis." MILCOM 2012-2012 IEEE Military Communications Conference. IEEE, 2012.

[9] Moskovitch, Robert, et al. "Unknown malcode detection using opcode representation." European conference on intelligence and security informatics. Springer, Berlin, Heidelberg, 2008..

[10] Yann, Trevor, and Oleg Petrovsky. "Detection of polymorphic virus code using dataflow analysis." U.S. Patent No. 7,069,583. 27 Jun. 2006.

[11] Polychronakis, Michalis, Kostas G. Anagnostakis, and Evangelos P. Markatos. "Network–level polymorphic shellcode detection using emulation." International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Berlin, Heidelberg, 2006.

[12] Muttik, Igor, and Duncan V. Long. "Detecting computer viruses or malicious software by patching instructions into an emulator." U.S. Patent No. 6,907,396. 14 Jun. 2005..

[13] Rogers, Antony John, Trevor Yann, and Myles Jordan. "Detection of viral code using emulation of operating system functions." U.S. Patent No. 8,341,743. 25 Dec. 2012.

[14] Stepan, Adrian E. "Defeating polymorphism: beyond emulation." Proceedings of the Virus Bulletin International Conference. 2005.