International Journal of Scientific Engineering and Research (IJSER)
ISSN (Online): 2347-3878
Impact Factor (2020): 6.733

A Study of Malware Analysis and Malware Detection Methods in Cyber-Security

Karan Chawla

Abstract: ***Any programme or file that purposefully hurts a computer, network, or server is known as malware, or malicious software. These harmful programmes steal, encrypt, and destroy private information. To detect malware, antivirus software often relies on a signature-based approach. To transmit malware that infects devices and networks, malware developers employ a range of physical and virtual methods. On the other hand, the behavior-based approach makes use of suspicious files that are run in a controlled setting, observed, and classified as hazardous if their behaviors resemble known malware. Behavior-based analysis may be used to detect new malware and malware that use obfuscation techniques; however it is time-consuming and has a high false positive rate. The memory-based approach is an option that is now gaining favor in malware detection due to the volume of data disclosed in the memory dump that may be used to investigate dangerous activities. Future malware is predicted to be more sophisticated. Attackers may utilize cutting-edge encryption or obfuscation technologies to render malware detection and analysis nearly difficult. Anti-virus programmes often detect malware by looking for well-known signatures. Unfortunately, a simple obfuscation technique may be used to readily avoid this method. Both static and dynamic assessments have significant drawbacks. As an alternative, malware may be thoroughly analyzed via memory analysis. Malware has a strong ability to conceal its code within the computer system. To finally carry out its operations, malware must, however, run its code in memory. This review paper analyses three different methodologies for malware analysis, namely, static, dynamic and memory analyses.***

Keywords: Encryption, Malware, Signature, Computer Network, Behavior-Based Analysis, Memory Analysis, Dynamic Analyses

1. Introduction

Software that is installed on a computer without the user's consent is known as malware. By interfering with computer operations, stealing data, or getting around access rules, it can cause damage to the computer system. Malware, sometimes known as harmful software, is posing an increasingly serious danger to the computing industry. The sheer volume of malware makes it hard for human engineers to tackle it. Therefore, malware detection systems are used by security researchers to find malware. Systems for detection go through two stages: analysis and detection. A signature-based technique is frequently used by antivirus software to detect malware.

This method is quick and has a low risk of false positives for identifying known malware. However, malware that employs obfuscation techniques may readily circumvent signature-based detection, which fails to find undiscovered malware. The behavior-based technique, on the other hand, uses suspicious files to be performed in a controlled environment, watched, and labeled as harmful if their behaviors match those of recognised malware. Detecting unknown malware and malware that uses obfuscation techniques can be done with behavior-based analysis, but it takes a long time and has a high false positive rate.

The abundance of data revealed in the memory dump that may be utilized to look into harmful activity makes the memory-based technique an alternative that is lately growing in popularity in malware detection.

The majority of antiviral programmes on the market employ a signature-based strategy. This method uses a malware file that has been captured to extract a distinctive signature that may be used to identify other malware that is identical. A file hash or a series of bytes known as a signature can be used to recognise particular malware. Attackers may easily alter the malware signature to avoid being picked up by antivirus software, though. While signature-based malware detection is highly quick and efficient at catching known malware, it cannot catch recently released malware. The signature-based strategy for malware detection relies on applying static analysis to extract special byte sequences known as marks and demonstrates the general signature-based process.

Anomaly or behavior-based detection are other names for heuristic-based detection. The actions taken by malware while it is running are examined in a training (learning) phase of this detection. On the basis of a pattern gleaned from the training test, the file is then classified as malicious or genuine during a testing (monitoring) phase. Heuristic-based methods frequently rely on data mining methods to comprehend how running files behave; examples of these methods are Support Vector Machine, Naive Bayes, Decision Tree, and Random Forest.

2. Static Analysis

Software static analysis is carried out without actually running the programme. Opcode sequences (obtained by disassembling the binary file), control flow graphs, and other information are a few examples of what we may learn via static analysis. These feature sets can be combined or used singly to identify malware. The authors introduced a malware detection method based on control flow graphs and static analysis. Their method, which focuses on identifying malware's obfuscation patterns, has a high degree of accuracy. Malware detection via static detection has made use of machine learning approaches. Based on collected opcode sequences, metamorphic malware is efficiently categorized using hidden markov models. Malware detection uses a similar technique using Profile Hidden Markov Models and Support Vector

Machines. For malware identification, some researchers have employed function call graph analysis, while others use an opcode-based similarity measure that uses straightforward substitution cryptanalysis methods. Both API call sequences and opcode sequences are used to assess if a section of code resembles any specific malware. Based on collected opcode sequences, metamorphic malware is efficiently categorized using hidden markov models. Malware detection uses a similar technique using Profile Hidden Markov Models and Support Vector Machines. For malware identification, some researchers have employed function call graph analysis, while others use an opcode-based similarity measure that uses straightforward substitution cryptanalysis methods. Both API call sequences and opcode sequences are used to assess if a section of code resembles any specific malware. With this method, Portable Executable files (PE files) are examined without being run. To avoid being analyzed, malware frequently utilizes binary packers like UPX and ASP Pack Shell. Before being analyzed, a PE file has to be unpacked and decompressed. A disassembler programme, like IDA Pro or OlleyDbg, which shows assembly instructions, provides details about the virus, and extracts patterns to pinpoint the attacker, may be used to decompile a windows executable file.

The detection pattern may be determined using static analysis techniques including Windows API calls, text signatures, control flow graphs (CFG), opcode (operation codes) frequency, and byte sequence n-grams. Windows API (short for Application Programming Interface) calls are used by almost all programmes to interact with the operating system. For instance, the Windows API "OpenFileW" in "Kernel32. dll" creates new files or opens existing ones. As a result, API calls provide information about how programmes behave and may be used to identify malware.

For instance, the Windows API calls "WriteProcessMemory," "LoadLibrary" and "CreateRemoteThread" are suspected of being utilized by malware to inject DLLs into processes, while infrequently occurring as a valid set. In the section on memory analysis, DLL injection is covered. Strings are a reliable sign of malevolent activity. Since strings frequently contain essential semantic information, they indicate the attacker's intentions and objectives. As an instance, the string "This programme cannot be run in DOS mode" denotes a malicious file when it is discovered outside of the standard PE header, a characteristic of installers and droppers.

A control flow graph (CFG) is a directed graph that shows how a program's control flow is implemented. Code blocks are represented by nodes in the CFG, and control flow channels are represented by edges. CFG may be used to record a PE file's behavior and extract the programme structure during malware detection.

Opcodes are the initial component of a machine code instruction, commonly referred to as machine language that specify the operation the CPU should do. A complete machine language command, such as "move eax 7", "add eax ecx" or "sub ebx 1" is made up of an opcode and, optionally, one or more operands.

By analyzing opcode frequency or comparing opcode sequences, opcode may be used as a characteristic in malware identification.

All consecutive subsequences of a sequence of length N are referred to as N-grams [21]. For instance, the word "MALWARE, " which consists of a string of seven characters, may be broken up into three-gram units called "MAL, " "ALW, " "LWA, " "WAR, " and "ARE. " NGrams have been used in conjunction with a number of detection tools, including API calls and opcodes. Other characteristics, such as file size and function length, have also been employed in static analysis in addition to the ones mentioned above. Static analysis also includes networking aspects like TCP/UDP ports, destination IP addresses, and HTTP requests.

Kirat and Vigna have conducted some of the most important studies on malware signature avoidance methods. They were able to gather 78 comparable evasion signature approaches from 2810 different malware samples. A novel method was put up by Hashemi and Hamzeh that turns the executable file's unique opcodes into digital images. Then, using one of the most well-known texture extraction techniques in image processing, Local Binary Pattern (LBP), visual characteristics are recovered from the image. Finally, malware detection is done using machine learning techniques. The accuracy percentage for the suggested detection method was 91.9%. Additionally, Shaid and Maarof recommended showing viruses as pictures.

Their method records malware API requests and transforms them into visual clues or graphics. Malware variations are recognised using these photos.

The accuracy of the API arguments list, which is 98.4%, and the false positive rate, which is roughly 3%, demonstrated that it is superior to the other two sets. Similar to this, Han et al. used static analysis to extract APIs from the IAT table (import Address Table). They assessed the similarity between the extracted API sequence and another sequence in order to categorize the malware family. Han discovered that malware from the same family is around 40% identical and that the false positive rate is 16%. To find malware that uses shellcode, some people have used the WinDbg programme to analyze native API sequences and used Support Vector Machine. They had a 94.37% accuracy rate despite using a too-small training set. False negative rate, on the other hand, was as high as 44.44%.

3. Dynamic Analysis

Dynamic analysis can provide details about API calls, system calls, instruction traces, registry changes, memory writes, and other things. The authors create fine-grained models that are intended to capture malware's system call-based behavior. The resultant behavior models are

represented as graphs, where system calls are represented by the vertices, and dependencies between calls are represented by the edges.

The API calls' parameters and return values make up the geographical information, while their ordering makes up the temporal information. For the purpose of detecting malware, this data is utilized to create formal models that are fed into industry-standard machine learning techniques. Again, malware detection uses API sequences. Frequency analysis of API call sequences is used to analyze malware.

The conversion of dynamic instruction sequences into abstract assembly blocks is logged. A classification model is created using data mining methods and feature vectors that are taken from this data. Some people employ executables' instruction trace logs, where this data is dynamically gathered. These traces are then analyzed as graphs, with the instructions acting as the nodes, and transition probabilities are computed using statistics from the instruction traces. The actual categorization is made using Support Vector Machines.

Another name for it is behavior analysis. In this study, suspicious files are run and watched in a supervised setting like a virtual machine, emulator, or simulator. The basic reason why the infected files must be examined in a covert environment is that certain malware is backed by anti-virtual machine and anti-emulator capabilities. When malware recognises such an environment, they perform properly and do not engage in any nefarious behavior.

Dynamic analysis is more efficient than static analysis because it can analyze an infected file without having to deconstruct it. Additionally, malware that is both known and unknown can be found through dynamic analysis. Additionally, malware that is polymorphic and disguised cannot avoid dynamic detection. Dynamic analysis requires a lot of time and resources, though.

Several methods, including function call monitoring, function parameter analysis, instruction traces, and information flow tracing; can be employed with dynamic analysis. According to a review of the assessed articles, network characteristics, file systems, Windows registries, and system calls are all often used in malware dynamic analysis.

Zeus malware was categorized by Mohaisen et al. using a variety of machine learning methods. The classifier was trained using artifacts including registry, file system, and network information.1980 samples of the Zeus Banking Trojan made up the dataset, and accuracy was close to 95%.

Mohaisen et al. then suggested AMAL, an automated and behavior-based malware analysis and labeling system, in their subsequent study.

AutoMal and AutoLabel are the two halves of AMAL. To analyze malware samples, Automal makes use of file systems, network activity logging, and registry monitoring functions. Additionally, AutoLabel groups malware samples into families according to their behavior. Over 115, 000 malware samples were utilized by AMAL, and a 99% detection rate was attained.

The majority of dynamic approaches employed API calls to simulate malware behavior. Following that, linked API requests with similar semantic goals are organized into sequences. Using a decision tree, they were able to attain a maximum accuracy of 97.19%.

The method has a 99.8% accuracy rate and zero false positives. Only 80 APIs were chosen for the experiment, and utilising Decision Tree and Naive Bayesian methods, the identification rate approached 95%. Dynamic analysis uses malware that is executed in a safe setting to watch how dangerous files behave in real time without getting yourself into trouble. There are several kinds of control environments, including virtual machines, emulators, debuggers, and simulators. Next, we describe each sort of malware and the methods it employs to look for a controlled environment. An emulator is a controlled environment that is used to manage how a malicious programme is executed. The CPU, hard drive, and resources are all under the control of a comprehensive emulation system. Emulators can be distinguished based on the portion of the running environment that is under control. The BitBlaze project's TEMU, a comprehensive emulation system that provides dynamic binary analysis by keeping track of elements including network activity, memory locations, function calls, processes, modules, and API calls, was presented in 2008. Another form of emulator, TTAnaylze, uses the free source machine emulation QEMU and offers an automated malware analysis module that logs native and Windows APIs. The bulk of viruses can, however, recognise a simulated environment. Malware can carry out operations that function outside the mimicked environment when just partial emulation is used to determine if it is executing in a controlled environment. Additionally, malware may still identify the traits and consequences of a whole environment system, such as identifying flawed CPU features and contrasting system characteristics (i. e. the user who is now signed in).

Another kind of controlled environment is a debugger, which is a programme that monitors and investigates the operation of other binary programmes. Debuggers like WinDbg, OllyDbg, and GDB may be used to track the execution behavior of questionable binaries down to the individual instruction level. WinDbg further enables kernel debugging, unlike OllyDbg. Additionally, the built-in debugger in IDA Pro, a static analysis tool, is less powerful. However, the easiest way for malware to detect that it is being debugged is to leverage the Windows API. The API calls "IsDebuggerPresent", "CheckRemoteDebuggerPresent", and "OutputDebugString" can all be used to prevent debugging.

Malware will also examine registry keys, files, and directories for indications that a debugging tool has been installed on the system. Additionally, malware can

leverage there are a number of methods, such as exceptions and interrupts, to stop a programme from running while it is being debugged.

Simulator is a software programme that mimics an operation so that it may be watched by the user without actually conducting it. Virus may run in a regulated virtual environment with the help of simulator programmes like CWSandbox, Norman Sandbox, and Detours, which also record the behavior of the virus. DLL injection is accomplished with Detours to intercept function calls made by a process to any DLL, and API hooking is accomplished with CWSandbox to record calls to Windows APIs made by malicious software.

Conversely, Norman Sandbox replicates the host machine's Windows operating system, LAN, and Internet access. Malware searches for certain sandbox products by checking for their presence in the registry, files, or processes in order to perform anti-simulation. The execution time is another method for sandbox and virtual environment detection since controlled environment instruction execution takes longer than real-world instruction execution.

Virtual machines (VMs) are the most popular controlled environment. The computer programme known as VM runs both an operating system and applications. These programmes are kept separate from the host system. As a result, executing files or programmes within a virtual machine cannot affect the host computer. Applications for virtual machines include VirtualBox, Parallels, and VMware.

Software that creates operates, and controls virtual machines are known as a virtual machine monitor (VMM). It is also in charge of designating hardware for virtual machines. Malware, on the other hand, looks for artifacts that installed VM tools leave in the file system, registry, and process listing to determine whether a virtual machine (VM) is present on a system. In order to detect the presence of VM tools, malware may also search for specific instructions that may be called in user mode, such as "sidt," "sgdt" and "sldt". Additionally, hardware traits and features may contribute to the presence of virtual machines. Malware can test certain bits to see if they are operating within a virtual machine, such as the CPUID hypervisor bit, which is set to zero in the actual system. Additionally, because the majority of debuggers and virtual machines produce files and drivers specific to that tool, malware can search for these artifacts to determine whether virtual machines or debuggers are present.

4. Memory Analysis

In recent years, memory analysis has gained popularity as a method for malware analysis due to its effectiveness and accuracy. Malware researchers are drawn to memory analysis because it provides a thorough examination of malware by looking into malicious hooks and code outside of the function's typical scope. It employs memory images to analyze data about the operating system, open programmes, and overall health of the machine. Memory forensics investigations go through two stages: memory analysis and memory acquisition. Tools like Memoryze, FastDump, and DumpIt are used in the memory acquisition to dump the target machine's memory in order to generate a memory picture. Using software like Volatility and Rekall, the memory analysis stage involves examining the memory picture in search of harmful activity.

Numerous studies on memory forensics methods have been put out. Through the use of API trigger-based memory dump technology, which enables Cuckoo to dump memory at any desired API call. In order to identify malware, researchers have examined three elements taken from memory images: imported libraries, registry activity, and API function calls. Their maximum accuracy, utilizing SVM and registry activity data, was roughly 96%. A method that uses the process handles to determine if the suspected sample is malicious or benign has been suggested by some.

The investigation has shed information on the primary handle types that malicious processes frequently employ, including section handles, process handles, and mutants. Rootkits can leave behind memory artifacts at the kernel level, including: drivers, modules, SSDT hooks, IDT hooks, and callbacks. Callback functions, changed drivers, and connected devices are the kernel-level operations that the experiment has shown to be the most dubious. The outcomes of the publications surveyed that used memory analysis in their malware detection techniques.

5. Analysis

One of the main problems is that modern antivirus software relies on signature-based detection methods, which are ineffective given the daily creation of hundreds of new malwares and the ease with which malware authors may alter the malware signature. Obfuscation methods including dead-code insertion, register reassignment, and subroutine reordering are a problem with signature-based approaches. The tiny false positive rate with signature-based detection is another problem.

The problem with heuristic or behavior-based approaches is the high probability of false positives and excessive monitoring time. Additionally, hundreds of extracted characteristics are being reduced, their similarities are being assessed, and malware activity is being watched, all of which have a direct impact on the capacity to identify zero-day malware.

The capacity to identify zero-day malware assaults is also directly impacted by the reduction of thousands of extracted characteristics, comparison of their similarities, and monitoring of malware activity.

API calls are a crucial indicator for malware identification in static analysis since they show programme behavior. For example, the Windows API calls "WriteProcessMemory", "LoadLibrary", and "CreateRemoteThread" are suspected of being utilized by malware to inject DLLs into processes, while infrequently

occurring as a valid set. Strings are a reliable way to spot nefarious activity. Since strings frequently contain essential semantic information, they indicate the attacker's intentions and objectives. For instance, the phrase "This programme cannot be run in DOS mode" denotes a malicious file when it is discovered outside of the standard PE header, which is a characteristic of installers and droppers.

A better technique to use since static analysis required disassembling of information such as opcode and N-grams and dynamic analysis is resource consuming would be to use resource efficient data mining approaches during dynamic analysis or to send a pre-worked image of the possible marks in the malware using historical data that could be found to minimize the workload of the dynamic analysis process. Another method would be to use “insertion” techniques in dynamic analysis to insert pre-written code in the malware to check if it reacts to give a certain known behavior outcome which would also reduce the false positive rate since this would not be a shot in the dark but rather an interaction with the malware.

6. Conclusion

We face a serious danger from malware to our computer systems, internet, and data. Malware authors present many difficulties for anti-virus software and security researchers by creating complex malware that frequently modifies its signature to evade detection and by releasing more sophisticated versions of malware that use new obfuscation techniques. In this article, we provide a quick overview of malware detection techniques and malware kind. Static, dynamic, and hybrid malware analysis methodologies have all been studied. We also spoke about how memory forensics may be used to locate malware artifacts. We also talked about the potential of memory-based analysis for malware identification.
The use of obfuscation, attacking, and anti-analysis tactics by malware to avoid detection has also been examined. Finally, this article has examined the future course of malware development as well as the primary malware dataset sources.

7. Future Directions

Many security specialists think that malware's future is still up in the air. Future malware creation will face a variety of difficulties that security companies and researchers should take into account. The automation of malware variant creation is the first cause for concern.

Attackers can create automated systems that can generate hundreds of different malware samples each day by researching the most recent malware detection techniques and utilizing machine learning. Second, malware groups may rent or sell such malware automation technologies, opening the door for unskilled groups and amateur hackers to enter the realm of malware.

Third, the functionality and structure of malware are always changing. The majority of the surveyed approaches learned and tested the behaviors (the classifier) on a single malware dataset.

RAM that is volatile retains its data until the device is turned off. As a result, examining the RAM can provide information about system activity. Running processes, Dynamic Link Library (DLL), files, registry keys, services, sockets and ports, and active network connections are just a few examples of the valuable live information that may be found in memory. Thus, memory analysis is a potential method that is anticipated to gain popularity in malware detection alongside data mining and machine learning methods. Collecting malware samples is crucial for researchers to investigate malicious tactics and strategies. Using honeypots, a special machine set up to draw attackers so they may observe their attacking methods, is one approach to gather samples. Additionally, researchers may employ known harmful URLs.

References

[1] H. Sun, X. Wang, R. Buyya and J. Su, "CloudEyes: Cloud-based malware detection with reversible sketch for resource-constrained Internet of Things (IoT) devices", *Softw. Pract. Exper.*, vol.47, pp.421-441, Mar.2017.
[2] M. Noor, H. Abbas and W. B. Shahid, "Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis", *J. Netw. Comput. Appl.*, vol.103, pp.249-261, Feb.2018.
[3] S. Sharmeen, S. Huda, J. H. Abawajy, W. N. Ismail and M. M. Hassan, "Malware threats and detection for industrial mobile-IoT networks", *IEEE Access*, vol.6, pp.15941-15957, 2018.
[4] O. A. Waraga, M. Bettayeb, Q. Nasir and M. A. Talib, "Design and implementation of automated IoT security testbed", *Comput. Secur.*, vol.88, pp. 1-17, Jan.2020.
[5] R. Kumar, X. Zhang, R. U. Khan and A. Sharif, "Research on data mining of permission-induced risk for Android IoT devices", *Appl. Sci.*, vol.9, no.2, pp. 1-22, Jan.2019.
[6] P. K. Sharma, J. H. Park, Y. -S. Jeong and J. H. Park, "SHSec: SDN based secure smart home network architecture for Internet of Things", *Mobile Netw. Appl.*, vol.24, no.3, pp.913-924, Jun.2019.
[7] Y. -S. Jeong and J. H. Park, "IoT and smart city technology: Challenges opportunities and solutions", *J. Inf. Process. Syst.*, vol.15, no.2, pp.233-238, Apr.2019.
[8] T. Lei, Z. Qin, Z. Wang, Q. Li and D. Ye, "EveDroid: Event-aware Android malware detection against model degrading for IoT devices", *IEEE Internet Things J.*, vol.6, no.4, pp.6668-6680, Aug.2019.
[9] P. K. Sharma, J. H. Ryu, K. Y. Park, J. H. Park and J. H. Park, "Li-Fi based on security cloud framework for future IT environment", *Hum. -Centric Comput. Inf. Sci.*, vol.8, no.1, pp.1-13, Aug.2018.
[10] J. Kang, S. Jang, S. Li, Y. -S. Jeong and Y. Sung, "Long short-term memory-based malware classification method for information security", *Comput. Electr. Eng.*, vol.77, pp.366-375, Jul.2019.

[11] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques", *Hum. -Centric Comput. Inf. Sci.*, vol.8, no.1, pp.1-22, Jan.2018.
[12] R. Tahir, "A study on malware and malware detection techniques", *Int. J. Eng. Educ.*, vol.8, no.2, pp.20-30, Mar.2018.
[13] B. Yu, Y. Fang, Q. Yang, Y. Tang and L. Liu, "A survey of malware behavior description and analysis", *Frontiers Inf. Technol. Electron. Eng.*, vol.19, no.5, pp.583-603, May 2018.
[14] R. Sihwail, K. Omar and K. A. Z. Ariffin, "A survey on malware analysis techniques: Static dynamic hybrid and memory analysis", *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol.8, no.2, pp.1662-1671, 2018.
[15] C. -W. Tien, J. -W. Liao, S. -C. Chang and S. -Y. Kuo, "Memory forensics using virtual machine introspection for malware analysis", *IEEE Conf. Depend. Secure Comput*, Aug.2017.
[16] J. Landage and M. P. Wankhade, "Malware and malware detection techniques: A survey", *Int. J. Eng. Res. Technol.*, vol.2, no.12, pp.61-68, Dec.2013.
[17] D. Ucci, L. Aniello and R. Baldoni, "Survey of machine learning techniques for malware analysis", *Comput. Secur.*, vol.81, pp.123-147, Mar.2019
[18] M. Rhode, P. Burnap and K. Jones, "Early-stage malware prediction using recurrent neural networks", *Comput. Secur.*, vol.77, pp.578-594, Aug.2018.
[19] K. Han, B. Kang and E. G. Im, "Malware analysis using visualized image matrices", *Sci. World J.*, vol.2014, pp.1-15, Jul.2014.
[20] E. Cozzi, M. Graziano, Y. Fratantonio and D. Balzarotti, "Understanding Linux malware", *39th IEEE Symp. Secur. Privacy*, May 2018.
[21] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", *13th Eur. Conf. Comput. Vis*, Sep.2014.
[22] H. Zhou, "Malware detection with neural network using combined features", *15th Int. Annu. Conf. Cyber Secur*, Aug.2018.
[23] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar and A. Sharif, "A multimodal malware detection technique for Android IoT devices using various features", *IEEE Access*, vol.7, pp.64411-64430, 2019.
[24] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "OPEM: a static-dynamic approach for machine-learning-based malware detection, " in International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions. Berlin, Germany: Springer, 2013, pp.271-280.
[25] M. Tan and Q. V. Le, "EfficientNet: rethinking model scaling for convolutional neural networks, " in Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, 2019, pp.6105-6114.