The 13th IEEE International Conference on Dependable Systems, Services and Technologies, DESSERT'2023
13-15 October, 2023, Athens, Greece

2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT) | 979-8-3503-9611-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/DESSERT61349.2023.10416467

State-based Sandbox Tool for Distributed Malware Detection with Avoid Techniques

Pavlo Rehida
Department of Computer Engineering and Information Systems
Khmelnytskyi National University
Khmelnytskyi, Ukraine
pavlo.rehida@gmail.com

George Markowsky
Department of Software Engineering and Game Development
Kennesaw State University
Marietta, USA
gmarkows@kennesaw.edu

Anatoliy Sachenko
Department of Informatics
Kazimierz Pulaski University of Technology and Humanities
Radom, Poland
Research Institute for Intelligent Computer Systems
West Ukrainian National Unversity
Ternopil, Ukraine
sachenkoa@yahoo.com

Oleg Savenko
Department of Computer Engineering and Information Systems,
Khmelnytskyi National University
Khmelnytskyi, Ukraine
savenko_oleg_st@ukr.net

***Abstract* — This paper deals with the problem of detecting the malware by using emulation approach. Modern malware include various avoid techniques, to hide its anomaly actions. Advantages of using sandbox and emulation technologies are described. Various anti-emulation techniques that are used in modern malware considered. Obfuscation as one primary approach to hide malware malicious actions described and discussed. State of emulator is presented, and the advantages of its usage are covered. Distributed model for malware detection is considered. Basic emulator and its current capabilities presented. Prepared files that represent malware are described. Experimental results for developed files that differs with included avoid techniques are presented. Disadvantages of proposed approach is described. Future research and sandbox improvement are described.**

Keywords — malware detection, sandbox, anti-emulation techniques, obfuscation techniques

I. INTRODUCTION

Information technologies (IT) are increasingly being used both for personal and business purposes every year. IT is constantly evolving and with each year it provides more convenience in its use. Especially, it can be noticed by growing number of internet services, desktop, and mobile applications, that are widely used for solving personal and working tasks.

One of the major drawbacks of the widespread use of IT services is that they often involve the utilization of personal or business information. This data can be used by malicious actors for personal gain. Every year, various scientific laboratories and research centres record an increasing number of both the viruses themselves and the method of their influence [1-3].

Given these conditions, the search for new way of detecting malicious software is a relevant and important task. The primary goal of this paper is to discover a new approach for detecting malicious software based on sandbox and emulation techniques. Their effectiveness is confirmed by numerous scientific studies [4-6] and practical usage in various specialized software products, such as Cuckoo sandbox [6,7], FireEye [8], Joe Sandbox [9], VMRay Analyser [10], McAfee Advanced Thread Defence, Bitdefender etc. Also, it is necessary to take into account the advantages and disadvantages of these technologies and involve studies from other fields of computer science to achieve enhanced outcomes.

The remaining sections of the paper are organized as follows:

In the next section considers usage and combination of the sandbox and emulation techniques as most efficient for detecting malwares. The primary drawback is also discussed here, along with the suggested approach to overcome it. Widely used anti-emulation and obfuscation techniques that are used for masking of malware software presented and discussed. Section III considers the utilization of emulator's states to trigger the usage masking techniques by malwares. Presents the developed emulator and its core functionality. Described basic method of detecting the malware with using the developed emulator This approach enables the detection of malware when it temporary suspends malicious behaviour during some phases of its execution. Presented the distributed system model for detection malwares that can overcome the main disadvantage of emulation technology. Section IV. shows the main results of this paper and considers next scientific research. Approach to overcome the primary drawback of the emulator suggested.

The objective of this paper is to present a method for distributed detection of malwares with obfuscation and anti-emulation techniques that is based on sandbox and emulation technologies.

II. STATE OF THE ART

A. Malware detection based on emulation

Developing and implementing masking techniques is one of the primary tasks faced by developers of malicious software. Their primary objective is to conceal the malicious actions executed by malwares. This allows to hide viruses from analysis and the potential inclusion of their

"fingerprints" in malwares signature databases [11]. Masking also serves the important function of safeguarding the intrusion method for malware developers. Although there are now many viruses, the methods of their intrusion into host are not unique for all of them. Therefore, malware developers use various methods to hide them from detection.

Papers [4,12,13] examine approaches in detail for detection the software analysis within the host, that is runs on it. These studies are important, because it allows to understand how exactly malwares scan host for presentence of emulation. These papers deal with detailed classification of exiting masking (avoid) techniques, and provide next classes: anti-emulation, anti-debugging, anti-disassembly and etc. As it has mentioned earlier, the usage of combination of sandbox and emulation technologies gives the best results for detection malwares. This combination allows to analyse in detail the behaviour of potential malware. This study will help to create a "fingerprint" of malware, that will help to detect other viruses that are based on already analysed ones, or even that who uses the same strategies for invasion. These fingerprints may be used in other software applications or even systems [14] that provides high level of safety by combining multiple security approaches.

Usage of sandbox and emulator, allows to achieve the full emulation of the system. In this case, the sandbox used as wrapper and provides huge options for studying of malware executing process. Developing such application requires deep understating basic concepts of hardware components operation. Since sandbox is a software, the representation of physical components of emulated hardware will be as a basic block of any program. While emulating the Central Processing Unit (CPU), all registry values will be shown as the variables, internal memory – data structure, and the instructions – as methods that operates with the variables. In this case, wrapper (sandbox) should be developed as part of the software that will have access to read the registers. In order to provide this approach, it is necessary to use software interrupts. Generally, this method allows to read registers after some number of executed commands CPU, and form state of CPU, to understand is the tested software trying to perform the malicious actions.

B. Anti-emulation techniques

Big number of malwares do not have a unique core (method of invasion into host) and often use methods already known system vulnerabilities. This allows to significantly reduce the time of developing new malwares. On the other hand, this causes one drawback: if invasion method was already analysed and malicious actions were detected by security engineers, the signature of malware is most likely known and written into signature database [15]. To avoid this situation, malware developers use different techniques.

Anti-emulation techniques are one of the key approaches that allows to hide the malwares from being detected. Malware developers knows that sandboxes are widely used to detect their software, so they trying to protect it. So, they trying to find new ways to detect the emulation and stop malicious action until malware get to host system. Emulated environment signals to malware that it will be analysed soon. Detecting the emulation achieves by performing special checks of environment, for example: searching specific files on host, processes and even existence of some instructions. Some of the malwares may even change its behaviour after detecting emulation. Some most widely used checks that are described in papers [16-18] considered:

Searching for signature of virtual tools. In this case, malware will try to find processes that are most likely means usage the emulation. For example, by using the Virtual Box or VMware causes appearing a lot of processes that are named as: *WMwareService.exe*

Timing analysis. Malware may perform the analysis of time that needed to execute instructions or evaluate delays of instructions. This is caused by that fact, that some instructions execute faster on hardware comparing to emulated environment.

CPU semantic attacks. After developing the emulator for any CPU, not all aspects of each instruction of it may be emulated correctly. Malware developers may attack these instructions in order to find emulation.

Hardware check. Malware may check the information and various parameters about hardware, to analyse the serial number of motherboard, SCSI controller, or even check the MAC address of network adapters.

SIDT and SLDT tables analysis. Store Interrupt Descriptor Table (SIDT) and Store Local Descriptor Table (SLDT) are located in different regions of memory for emulated environment and hardware. So, this information may be used to detect the emulated environment where malware is running.

System call analysis. By monitoring and the analysing the behaviour of the system calls, is also allows to detect the emulation.

Checking the filesystem. Malwares may also check the presence of specific system drivers that is used only in emulated environment. There are few known types of malwares that uses this method.

It is important to note that these techniques are constantly changing and evolving as malware authors adapt to advances in security technology. Emulation-based security solutions and security analysts must continuously update their methods and countermeasures to effectively analyse and detect and new malware, despite the evolution and improvement of anti-emulation techniques.

C. Obfuscation techniques

In general, term obfuscation stands for something unclear or even not understandable. These techniques are used by malware developers to make malwares difficult to understand, evade the analysis, and hide the malicious actions performed by it [19-21]. Obfuscation techniques are used to overcome static[22], dynamic[23,24] and hybrid[25] analysis. Most widely used can be divided on next categories:

Code insertion. This is way to easily change code and keep original idea of it the same. Often, additional code can be used with NOP (No Operation) instructions. Generally does not affect the semantics of code.

Register reassignment. Code that modified with this technique uses additional instructions to reassign values that stored inside registers. It is also classified as easy technique, but in combination with other it may be difficult to detect.

Subroutine reordering. This technique uses permutation in some part of code. In this case the result of execution looks the same as for original code.

Code transposition. In this case, code changed so it's semantic doesn't. To achieve this result, two ways are used: first uses randomly reordered code used with jumps and unconditional statements, the second one is based only on reordering of independent code parts.

Code integration. With this approach, malicious code integrates into program. Target program, firstly have to be decomposed, after that malicious part will be integrated. This technique is very effective.

Instruction substitution. The main idea of using this technique is to replace number of instructions with another, so the resulting will have the same effect. Using equivalent instructions make detection of malicious programs very hard.

Table 1 shows how these techniques can actually affect the code. Table is divided on seven columns, first column represents original code for calculation basic math operation, next six columns show how will look like obfuscated code.

After execution most of these examples, the resulting state of emulator will have the same value. It means that all components such as registers and memory will keep the same values. Generally, the idea of any malware is to get the control over the execution of the instructions. And shown examples simply demonstrate its. Obfuscation techniques in other hand can also be used to call malicious instructions between instructions of original program. This will allow to perform malicious actions and pretend that infected program acts as normal one. The best example for this situation, when infected program starts background malware and then works the same as normal.

This paper focused on malwares that has two types of techniques that used to protect it from detection: anti-emulation and obfuscation. Of course, there is another way to hide the malicious actions, but different studies shows that malwares quite often build as small integration to some program. The main reason for it, is that infected program should act and look the same way as original from user's point of view.

III. STATE-BASED EMULATION

Emulator`s state includes state of each component in CPU. In order to overcome the masking techniques, we suggest using the sandbox to collect information about the state with using software interrupts.

For this paper the basic emulator and sandbox for it was developed. This emulator supports some widely used

TABLE I. REPRESENTATION OF HOW OBFUSCATION TECHNIQUES IMPACT THE ORIGINAL CODE

Original Code	Code insertion	Register reassigment	Subroutine reordering	Code Transposition	Code integration	Instruction Substitution
LDA #5 CLC ADC #10 ADC #5 SEC SBC #3 STA result TAX BRK	LDA #5 CLC ADC #10 Inserted code 1 NOP LDX #0 NOP INC $2000 ADC #5 SEC SBC #3 Inserted code 2 NOP NOP NOP STA result TAX BRK	LDA #5 CLC ADC #10 Register reassigment: LDY #5 SBC #2 LDX #0 ADC #Y LDY #10 ADC #X SEC SBC #3 STA result TAX BRK	Addition: CLC ADC #10 ADC #5 RTS Substraction: SEC SBC #3 RTS Main: LDA #5 JSR Substraction JSR Addition STA result TAX BRK	Subroutine1: CLC ADC #10 SEC SBC #3 RTS Subroutine2: CLC ADC #5 RTS Main: LDA #5 JSR Subroutine1 JSR Subroutine2 STA result TAX BRK	Initialize variables: LDA #0 STA result LDA #5 STA operand1 LDA #10 STA operand2 LDA #2 STA operand3 Additional subrout: PerformMath: LDA operand1 CLC ADC operand2 ADC operand3 SEC SBC #3 RTS Main: LDA #5 JSR PerformMath TAX BRK	Initialize variables: LDA #0 STA result LDA #5 STA operand1 LDA #10 STA operand2 LDA #2 STA operand3 Additional subrout: PerformMath: LDA operand1 CLC ADC operand2 ADC operand3 SEC SBC #3 STA result RTS Main: LDA #5 JSR PerformMath TAX InstructionSubstitut. LDA operand1 ADC operand2 STA result2 BRK NOP NOP NOP

instructions that are built in most of new CPUs. Emulator works with next components: registers (Accumulator, PC,SP, X,Y) and internal memory. Each of these registers uses 1 byte of memory, and memory has 65,535 bytes. Register A is often used for math operations, while X and Y designed as supporting for providing the math operations (storing some temporary data) and loops. PC stands for program counter and simply counts the number of executed instructions, while SP used in conjunction with some instructions, such as PLA and PHA. Here is some of instructions that is supported on this basic version: LDA, STA, ADC, SBC, INC, DEC, JSR, RTS, TAX, TAY, TXA, TYA, and NOP.

To read the state of the emulator the basic sandbox is also developed. The main task of it is to: read list of instructions, pass it to the emulator and get the state while emulator executes instructions. Getting state means read all values from the registers and memory and process it, by simulating the software interrupts. Sandbox at any moment can send this interrupt and form current state. Each current state is a hash value of all data that was collected from emulator.

To overcome the situation when code was obfuscated with code integration it is suggested to perform interrupts every 3 or 5 performed instructions. And add state of previous state in as a current state before calculating new hash value (state).

With this approach it will be possible to detect if obfuscation was present. In this case the hash value of each state will change the resulting (comparing to state after performing all selected instructions).

So, emulator state potentially can detect the difference between original and obfuscated code. As it was mentioned earlier, some malwares can dynamically change its behaviour when emulation detected and, in this paper, we consider malwares that combine anti-emulation and obfuscation techniques. Suggested approach may be used in situation when malware have a few anti-emulation techniques and uses obfuscation dynamically.

The main idea of proposed method for detecting the malware is to form as many states of emulator as it was researched. Due to developed emulation of the system, it will be possible to change the state of emulator for all known anti-emulation techniques. When all states are provided the potential malware will be executed. If all states will return the same hash result after execution, it means that this test software is safe. The opposite situation – if one hash [26] not equal for all other states, it means that tested software has built-in anti-emulation technique, and it was used. Any software that does not have malicious code would never change it behaviour.

The main disadvantage of emulation technology is that it needs more computational resources comparing with the original hardware. Another disadvantage is that suggested method require additional checks. Number of checks may grow in a future and depends on number of new studied anti-emulation techniques. It is suggested to use the distributed system to overcome both of disadvantages. On a figure 1 described basic model for distributed malware detection.

Suggested to use heterogenous computing system that will use volunteer resources. This type of distributed computing may accumulate a big number of computation resources. For example, at its pick usage Folding@Home[27] combined so many computation resources, so it was more efficient comparing to IBM Summit supercomputer [28]. So, this system will work with the volunteers that want to share their computation resources. There are a lot of other future research should be done to organize such computing. Each computing element will have the software client that will connect to intermediate servers. Intermediate servers will be connected to central server. The main part of the client application is build-in sandbox, that will perform checks on malicious actions tested software. The role of server and intermediate servers is to distribute the needed number of states and potential malwares to check it. There are a lot of important tasks should be considered, to organize such calculations: provide fault-tolerant topology on top level of the distributed system [29], implement the modified voting approach to ensure that result of states execution was not changed, analyse load-balancing algorithms to make calculations efficient [30,31]. Considering that such system may be characterized as heterogeneous and system can not trust any of connected host, special attention should be paid to finding the appropriate algorithm that will divide tasks with duplication. Duplication will help to perform checks that will help understand if any of host returns right result. It is caused by that fact, that any of host does not belong to the system, so there is no guarantee that node is not infected with malwares, that may change the result of calculations. For this task it is necessary to use voting approach. When all nodes send the results of calculation, and server will understand if any of them acts suspicious. On other hand, this algorithm should consider the active nodes and their computing resources. This will help to build synchronous processing and reorganise the nodes if any of host stop working or send suspicious result.

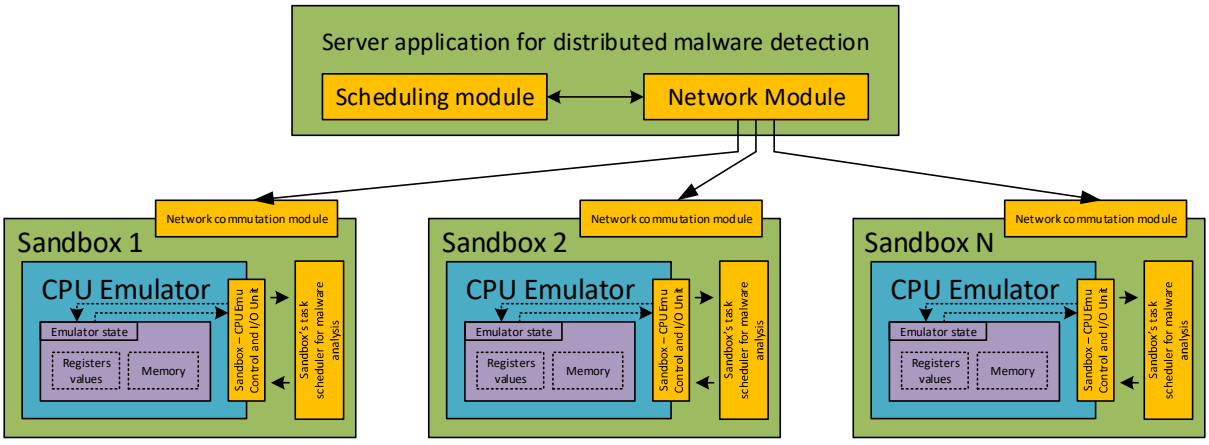

Fig. 1 – Distributed system for detection malwares based on emulator states

IV. EXPERIMENTAL RESULTS

For this paper it was developed basic emulator and sandbox that can be used for detecting the used obfuscation technique. The main goal of any developed malware is to take control over the process of executing the instructions. Any software must be compiled, before execution on any operation system. Compiled software consist of huge sequence of low-level instructions. Infected software can launch in background malicious action and then act with a normal behaviour. So, in other words, the process of detecting the malwares is to get know what instructions were used. The list of instructions may be changed if the emulation detected.

It is suggested that malwares with such combination of avoid techniques may have next acting strategies considering the detected emulation:

1. *Hiding mode*: malware detects the emulation, uses the obfuscation technique, launches program execution.
2. *Aggressive mode*: malware detects the emulation, do not use the obfuscation technique, launches program execution.
3. *Safe mode*: malware do not detect the emulation but uses obfuscation and launches program execution.
4. *Normal mode*: malware do not detect the emulation, do not use obfuscation, and launches program execution.
5. *Pause mode*: malware detects the emulation and pauses the execution of program.

To imitate the malware for this paper, it was used file that contains two lists of instructions and set of flags. First list of instructions stands original code, that was on infected. Another list has additional instructions that were added with the obfuscation techniques. Set of flags means some type of anti-emulation techniques that are included into current file (imitated malware). Generally, these flags indicate what checks will current file perform while executing in host.

On other hand, sandbox is also initiate a chosen state, that will change chosen characteristics of the emulator while executing the file. Executing the same potential malware on different states possibly will trigger the evasion technique. In this case, it may signalise that obfuscation technique was used. With obfuscation the code will be changed (executed second list of instructions) and behaviour will be changed too.

While executing the any developed file for these experiments, any of them will be processed as list of instructions and will be performed line by line. For this paper the state of emulator calculated (making a hash for each value that stored inside register) every five performed instructions. When hash value obtained firstly (after performed first instructions) it uses for next hash. In this case, hash value adds to the next state. After all instructions were executed on different states, this value should be compared. Furter, system decides about tested file. If all states return the same hash value it means that emulation technique was not used, so the tested software is safe to use. Otherwise, if any state will have different result, that means that obfuscation technique used, and malware detected.

In following example, the execution of instructions and the analysis of the obtained hash values (Table 2) will be considered.

TABLE II. HASH RESULT FOR TWO DIFFERENT EXECUTING MODES

Malware executed mode	***Hash state result***
Normal mode	e3e45ee99ec8e5c509b546b7e0280e60 b5ef2b87535b9b26e14bea6c7d195a76
Hiding mode	ab68ea2b6df4b86dab2e7b04f2077881 714cb110071b1eb42a34863e8729b1ef

Shown results represents the difference of final hash value state after executing the same code with obfuscation or without it.

For this paper it was developed client application and simple server application, and the following tests were performed. It was created a 10 base files with different original list of instructions. Each of these files also contains prepared obfuscated list of instructions and randomly assigned flag that indicates what emulation checked with this file. Based on these ten different files, it was created thousand files, where there are files with same instructions but with randomly distributed flags.

To evaluate the impact of redundant calculations that are necessary to ensure that host does not compromise the whole calculations, so some part of tasks was distributed over another hosts. For these experiments we define six different anti-emulation techniques that may be used to hide an activity. The following experiments were performed:

1. All files were checked on single host, so all states will be processed on it too.
2. Files were distributed between two hosts, states are distributed equally, one more random state added.
3. Three hosts get three states each, two of them are unique and additional one duplicates.

After all experiments finished next results (Table 3) obtained:

TABLE III. EXPERIMENTAL RESULTS

Number of used hosts	***Overall time consumed***	***General number of checks***
1	243 milliseconds	6 * 1000 * 1 = 6000
2	189 milliseconds	4 * 1000 * 2 = 8000
3	140 milliseconds	3 * 1000 * 3 = 9000

Shown results signalise about one disadvantage that should be discussed in future studies. Generally, is that adding two same hosts into system didn't help to achieve the double increasing of productivity. These caused with two key aspects: absence of the rating model for connected into system hosts and that fact that software applications require additional time for communication between each other. Well optimised rating approach will reduce the number of checks. It will help to choose some hosts that do not need to be checked, or the portion of such checks will be significantly smaller. Rating of each host should be based on several factors, such as: number of successfully completed tasks, predictable behaviour. Also, there are several assumptions how to optimise the network communication: enable using the asynchronous downloading data while current calculations performed and find the optimal portion of package size before sending it via Internet.

V. CONCLUSIONS

This paper deals with the problem of detecting the malwares that use anti-emulation and obfuscation techniques. The sandbox and emulation technologies as the most efficient approach considered. Both of avoid techniques are described, and their widely used examples are presented. The state of emulator as an approach that can be used to detect this type of malwares presented. Developed basic sandbox and emulator with technical details considered. For imitating the process of detecting it was suggested to use the file that contains two lists of instructions for developed CPU emulator. Also, this file includes special flags that stand for what list of instruction will be executed. This approach will allow to imitate the usage of code obfuscation if malware have detected emulation on host. This can be achieved by usage of software interrupts while executing the given instructions, that are part of any compiled application. For experiments it was usage strategy that calculates hash every five instructions. The results of experiments are presented. For future research it is planned to add more instructions into developed emulator, find more efficient way to calculate state of emulator while executing the instructions, research and propose a modified voting system for distributed malware detection system. As well, the process of duplication of states between members of distributed system is also very important. It is necessary to find the most efficient algorithm to reduce the duplication of tasks, respecting to appropriate level of calculation security.

REFERENCES

[1] D. Ghelani. "A perspective study on Malware detection and protection, A review". *Authorea.* September 13, 2022. https://doi.org/10.22541/au.166308976.63086986/v1

[2] F. Alswania, K. Elleithy. "Android Malware Family Classification and Analysis: Current Status and Future Directions". *Electronics*, 9(6), 942. https://doi.org/10.3390/electronics9060942

[3] R. Vinayakumar, A. Mamoun, K. P. Soman, P. Poornachandran, and S. Venkatraman. "Robust intelligent malware detection using deep learning". *IEEE access* 7 (2019): 46717-46738. https://doi.org/10.1109/ACCESS.2019.2906934

[4] S. Liu, P. Feng, S. Wang, K. Sun, J. Cao. "Enhancing malware analysis sandboxes with emulated user behavior" *Computers & Security*, 2022, 115, p.102613. https://doi.org/10.1016/j.cose.2022.102613

[5] S. Sechel. "A comparative assessment of obfuscated ransomware detection methods." *Informatica Economica 23(2/2019)*, 45-62. https://doi.org/10.12948/issn14531305/23.2.2019.05

[6] S. Jamalpur, Y. S. Navya, P. Raja, G. Tagore, and G. R. Koteswara Rao. "Dynamic malware analysis using cuckoo sandbox." *Second international conference on inventive communication and computational technologies (ICICCT)*, 2018, pp. 1056-1060, https://doi.org/10.1109/ICICCT.2018.8473346

[7] A. Walker, M. Faisal Amjad, and S. Sengupta. "Cuckoo's malware threat scoring and classification: Friend or foe?." *IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, 2019, pp. 0678-0684. https://doi.org/10.1109/CCWC.2019.8666454

[8] D. T. Le, D.T. Dinh, Q.L.T. Nguyen, and L.T. Tran. "A Basic Malware Analysis Process Based on FireEye Ecosystem." *Webology (ISSN: 1735-188X)* 19, no. 2 (2022), pp. 1011-1034.

[9] N. Kaur, A.K. Bindal, "A complete dynamic malware analysis." *International Journal of Computer Applications 135, no. 4*, pp. 20-25, 2016.

[10] M. Ali, S. Shiaeles, M. Papadaki, and B. V. Ghita. "Agent-based vs agent-less sandbox for dynamic behavioral analysis." *Global Information Infrastructure and Networking Symposium (GIIS)*, 2018, pp. 1-5. https://doi.org/10.1109/GIIS.2018.8635598

[11] K. Bobrovnikova, S. Lysenko, B. Savenko, P. Gaj, O. Savenko, " Technique for IOT malware detection based on control flow graph analysis" *Radioelectronic and Computer Systems*, 2022(1), pp. 141–153. https://doi.org/10.32620/reks.2022.1.11

[12] J. Singh, and J. Singh. "Challenge of malware analysis: malware obfuscation techniques." *International Journal of Information Security Science* 7, no. 3, 2018, pp. 100-110.

[13] C.V. Liță, D. Cosovan, and D. Gavriluț. "Anti-emulation trends in modern packers: a survey on the evolution of anti-emulation techniques in UPA packers." *Journal of Computer Virology and Hacking Techniques* 14, 2018, pp. 107-126. https://link.springer.com/article/10.1007/s11416-017-0291-9

[14] S. Bezobrazov, A. Sachenko, M. Komar, and V. Rubanau, "The methods of artificial intelligence for malicious applications detection in android os", *International Journal of Computing*, 15(3), 2016, pp. 184-190. https://doi.org/10.47839/ijc.15.3.851

[15] S. K. Sahay, A. Sharma, and H. Rathore. "Evolution of malware and its detection techniques." *Information and Communication Technology for Sustainable Development*, Springer Singapore, 2020, pp. 139-150, http://dx.doi.org/10.1007/978-981-13-7166-0_14

[16] S. Dash, "An overview of anti-virtualization technology and detecting virtualization in user space.", 2015.

[17] M. N. Olaimat, M. A. Maarof, and B.A.S. Al-rimy. "Ransomware anti-analysis and evasion techniques: A survey and research directions." *3rd international cyber resilience conference (CRC)*, 2021, pp. 1-6. https://doi.org/10.1109/CRC50527.2021.9392529

[18] D. Quist, V. Smith. "Detecting the presence of virtual machines using the local data table." *Offensive Computing* 25, no. 04, 2006.

[19] G. Markowsky, O. Savenko, S. Lysenko, A. Nicheporuk "The technique for metamorphic viruses' detection based on its obfuscation features analysis" *CEUR Workshop Proceedings 2104*, 2018, pp. 680–687.

[20] R. Sihwail, K. Omar, and K. A. Z. Arifin. "An Effective Memory Analysis for Malware Detection and Classification." *Computers, Materials & Continua*, 2021, 67, no. 2, pp 2302-2320, https://doi.org/10.32604/cmc.2021.014510

[21] R. Tahir "A study on malware and malware detection techniques." *International Journal of Education and Management Engineering*, 2018, no. 2, pp: 20-30. http://dx.doi.org/10.5815/ijeme.2018.02.03

[22] F. Biondi, T. Given-Wilson, A. Legay, C. Puodzius, and J. Quilbeuf, "Tutorial: An overview of malware detection and evasion techniques." *Leveraging Applications of Formal Methods, Verification and Validation. Modeling: 8th International Symposium*, November 5-9, 2018, pp. 565-586. https://doi.org/10.1007/978-3-030-03418-4_34

[23] H. Darabian, S. Homayounoot, A. Dehghantanha, S. Hashemi, H. Karimipour, R. M. Parizi, and K. R. Choo. "Detecting cryptomining malware: a deep learning approach for static and dynamic analysis." *Journal of Grid Computing*, 2020, pp. 293-303. https://doi.org/10.1007/s10723-020-09510-6

[24] J. Jueun, J.H. Park, and Y.S. Jeong. "Dynamic analysis for IoT malware detection with convolution neural network model." *IEEE Access*, 2020, 96899-96911. https://doi.org/10.1109/ACCESS.2020.2995887

[25] Y. S. I. Hamed, S. N. A. AbdulKader, and M. M. Mostafa. "Mobile malware detection: A survey." *International Journal of Computer Science and Information Security*, no. 1, 2019, pp. 56-65.

[26] I. Obeidat, M. AlZubi. Developing a faster pattern matching algorithms for intrusion detection system. *International Journal of Computing*, 18(3), 2019, pp. 278-284. https://doi.org/10.47839/ijc.18.3.1520

[27] Mengistu, Tessema M., and Dunren Che. "Survey and taxonomy of volunteer computing." ACM Computing Surveys (CSUR) 52, no. 3 (2019): 1-35.

[28] V.A. Voelz, V.S. Pande, and G.R. Bowman. "Folding@ home: achievements from over twenty years of citizen science herald the exascale era." *Biophysical Journal*, 2023 pp. 2852-2863. https://doi.org/10.1016/j.bpj.2023.03.028

[29] O. Goncharenko, P. Rehida, A. Volokyta, H. Loutskii, and V.D. Thinh. "Routing method based on the excess code for fault tolerant clusters with InfiniBand." *Advances in Computer Science for Engineering and Education II*, pp. 335-345, Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-16621-2_31

[30] A. Ullah, N. M. Nawi, J. Uddin, S. Baseer, and A. H. Rashed "Artificial bee colony algorithm used for load balancing in cloud computing: review", *IAES International Journal of Artificial Intelligence 8*, no. 2, 2019 pp. 156-167. http://doi.org/10.11591/ijai.v8.i2.pp156-167

[31] B. Kruekaew, and W. Kimpan. "Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning." *IEEE Access 10* , ISSN: 2169-3536, 09 February 2022, pp. 17803-17818. https://doi.org/10.1109/ACCESS.2022.3149955