A Distributed Malware Detection Model Based on Sandbox Technology

Pavlo Rehida[a], Tomas Sochor[b], Valeriy Martynyuk[a] , Olha Tarasova[a] and Viktoriia Orlenko[a]

[a] *Khmelnytskyi National University, Institutska str., 11, Khmelnytskyi, 29016, Ukraine*
[b] *Prigo University, Havirov, Czech Republic*

Abstract
The article proposes a model for distributed malware detection using sandbox technology. The analysis of modern malware detection tools and an overview of existing attacks were carried out. The justification of the selected detection method to be used by the model is carried out. Its main disadvantages are identified and the use of the distributed system as its solution is proposed. The key features of the use of heterogeneous computer systems for calculations and their adaptation to perform the task were considered. Detection of malware is proposed to be solved by analyzing the states of sandboxes, and evenly distributing these states among the computational elements of the system. Analysis how these states are changing will signal about potentially malicious software that uses anti-emulation techniques, thereby allowing the detection of malware. The basic set of levels of the proposed model is presented. The main tasks for the protection of calculations are defined, taking into account that the model will work in system with dynamical topology. The basic concept of load distribution between computing elements is proposed in order to ensure the synchronous operation of the system, taking into account the heterogeneity of the system. Two main strategies for protecting computing both at the level of computational elements and at the level of intermediate servers are defined. A basic algorithm for adding new elements to the system is proposed, and the use of a rating model is presented, which will ensure an appropriate level of protection of calculations.

Keywords 1
Malware detection, distributed computing, heterogenous computer systems, anti-malware techniques, voting system.

1. Introduction

In the modern world, the use of IT is widespread in almost all spheres of life, which greatly facilitates the completion of everyday tasks. Services based on the use of IT actively use personal or corporate information, which makes them very convenient to use. We trust our personal data to software and mobile applications, store it in cloud environments, companies use IT to automate internal processes, and conveniently operate confidential documents, etc. Therefore, the issue of the security of such information is important and considering the trends in their development [1,2], and it is necessary to investigate new methods and approaches of detecting malware.

The problem of malicious software in particular lies in several aspects, namely: the total number of already existing ones; speed of appearance of new ones; speed of appearance of new types; also with new software and hardware comes new vulnerabilities comes too. Considering all these factors, it is

IntelITSIS'2023: 4th International Workshop on Intelligent Information Technologies and Systems of Information Security, March 22–24, 2023, Khmelnytskyi, Ukraine
EMAIL: pavlo.rehida@gmail.com (P. Rehida); tomas.sochor@osu.cz (T. Sochor); martynyuk.valeriy@gmail.com (V. Martynyuk); tarasovao@khmnu.edu.ua (O. Tarasova); orlenkovs@khmnu.edu.ua (V. Orlenko);
ORCID: 0000-0002-6591-7069 (P. Rehida); 0000-0002-1704-1883 (T. Sochor); 0000-0001-5758-4244 (V. Martynyuk); 0000-0001-8574-6466 (O. Tarasova); 0000-0001-9601-1916 (V. Orlenko);

necessary to look for new ways of detecting malicious software, to use successes in other areas of IT to form a combined method that will effectively fulfil the task.

In paper [3], a thorough study was conducted on the dependence between the complexity of computer viruses and the probability of their detection. Thus, a comparison of the effectiveness of modern existing threat detection tools is carried out here, having previously divided them into the following categories: static threat detection [4], dynamic threat detection and modern web service solutions. Static tools include software analysers that can determine the application compilation time, check whether the application was packed by the UPX packager (which is often used for virus packaging [5]), check which functions and DLLs are used when the program is launched. Dynamic detection [6] is generally based on the use of a certain environment that allows you to observe its behaviour, namely: compare the status of registers [7], analyse the activity of processes and their impact on the operating system, and create an isolated environment for testing new software. Web service solutions usually use both detection approaches and are mostly free to use, but have one drawback, which is the maximum size of the file (software) that can be analysed.

One of the best methods for detecting malware is using sandboxes, as they provide full control over an isolated environment that will allow you to analyse the executable file. The main problem with the use of such tools is the significant need for computing resources. The purpose of this work is to present a method of testing software for anomaly behaviour by using a distributed sandbox system.

2. Emulation as an approach of detecting malicious software.

The article [3] presents a wide analysis of the use of modern tools for detecting malware, but if we analyse the test results for the selected set of malware, we can see that the largest percentage of detected threats is associated with the use of a sandbox, if we consider each technology separately. Symantec uses emulation technology to create an isolated environment, to test potentially malicious software. Emulators can use various techniques to search for viruses [8], in particular Cuckoo sandbox [9] uses the behavioural features of binary files. Independent studies performed by NSS Labs Breach Detection System Test in 2017. They showed the advantages of complete emulation of the system in the tasks of detecting malware. Lastline's sandbox, which uses emulation, has reached a 100% threat detection rate.

Therefore, emulation occupies an important place among the approaches that can protect computer systems. The following advantages of using this technology are determined:

1. Controlled environment for software testing.
2. Protection of hardware, operating systems, and the registry.
3. If malicious is detected, the sandbox is removed, which ensures the protection of the host.
4. Emulation technology eliminates incompatibility problems with software or hardware.

The disadvantages include the usual need for all software to constantly update, because older versions may contain security gaps and the fact that the sandbox technology requires large computing resources. It is important to take these features into account when creating a new tool for detecting malware. Also, before passing the malware check, the software will be considered as potential malware.

In addition to a large number of types of malwares, a large number of their pre-detection techniques are determined, which include: self-encryption and self-decryption, junk code, usage of equivalent instructions, block reordering [10], polymorphism, metamorphism, disguise and so on. Considering these tools from the point of view of using an emulator, such techniques are defined as anti-emulation, and are classified as obfuscation techniques. To detect emulation, the following are used: timing attacks, CPU semantics attacks, hardware characteristic attacks, fake API calls [11,12], structured exception handling. [13,14]

Timing CPU attacks include attacks that generally involve measuring the time required for a particular operation by the processor. These attacks are used to obtain cryptographic keys. Modern processors use the transition prediction module to build an effective order for executing instructions. Such attacks are aimed at confusing these queues, after which the processor will have to rebuild the execution queue [15].

CPU semantics attacks are aimed at analyzing the execution of instructions and their semantics. The developers of such attacks are guided by a well-formed set of instructions that will shift the execution of certain instructions so that important sensitive data remains in the cache.

Hardware characteristic attacks work with an attempt to use additional instructions to track changes in physical characteristics (such as electricity consumption), or to use error injections that will manipulate its behavior.

Fake API calls use scripts or programs that mimic the behavior of an authorized user or program and are used to search for system vulnerabilities.

Structured exception handling attacks uses an exploit to search for vulnerabilities in its mechanism, which involves overwriting the SEH chain, which will allow you to gain control of the program and execute needed code. A similar result can be obtained using a buffer overflow.

The attacks described above at first glance will not carry a threat, given that they are launched in a controlled environment. However, malware developers can use the results of their execution to analyze the external environment [16], and if emulation has been detected, then all malware actions are masked or stopped until the environment changes. Malware will wait for changes in the environment in which it operates in order to prevent it from being exposed and to continue performing its tasks. Taking into account these behaviors, it is proposed to introduce the concept of the sandbox state. The sandbox state accepts a certain set of characteristics that define it (command processing time, number of processes running, etc.) at a certain point in time. Define single state as, SB_1 then the set of states will depend on the number of characteristics $(n * n)$, so all state will be defined as:

$$SB_{state} = \{SB_1, SB_2 \dots SB_{n^2}\}$$

One of the biggest advantages of using sandboxes is full control over it and its state, so in [17] the proposed sandbox analyzes potential malware at 3 basic levels, namely: static analysis, real-time analysis and network analysis of malware. Modern technologies like machine learning also used in order to increase the malware detection rate by sandboxes, and based on RF, NN, DT and SWM algorithms [18]. Heuristic scanning also used as additional method of malware detection that based on examining its behaviour and characteristics. This approach based on three procedures: pattern matching, automatic learning, and environment emulation. In paper [19] this approach was analysed and one of the key conclusions is that it has high rate of false positive report, but combining both technologies: heuristic scanning and sandbox may help to achieve more reliable security results. Using the sandbox, we can record its state, throughout the entire potential malware set of commands, follow the behavior of the executable file during its operation [20] or, for example, generate a hash value, each step taking into account the result of the previous state, as this happens when using smart contracts. Thus, we will form a kind of imprint of the potential malware execution in the sandbox.

Considering anti-emulation techniques and changing sandbox states, we can assume that if we provide each potential malware with the required number of sandboxes with all sets of states that cover the processing of all anti-emulation techniques, we will be able to detect malware.

This approach will increase the success of the detection of malware, but it has one drawback: it is necessary to provide a large amount of computing resources to maintain all sandbox states for potential malware. Therefore, it is necessary to suggest a model that will meet all the described requirements.

3. Model of the distributed detection system using sandboxes.

Distributed computing has begun to be used in many areas of life. They are used to model natural processes, test hypotheses, help determine whether scientific research is taking place in the right direction. They have various forms and uses different approaches in relation to their organization. With the widespread use of IoT, the amount of data has also increased, and the importance of its processing also leads to the wide development of distributed computing. Since the widespread use of distributed computing, it is necessary to find a way of involving it in the issues of data security.

To solve the problem, it is proposed to use a distributed computing system. But first we will characterize it. Since, that it is a huge number of various components that is used to create computers and workstations, we can determine that designing a heterogeneous distributed computing system will be the most optimal approach. In addition, such systems form the most effective, in terms of computing resources, solutions [21], and a distributed computing system based on the use of volunteer resources, at peak times, had computing power that was 10 times greater than IBM Summit [22].

Let’s define the main components of the model taking into account its features:

- Server.
- Set of intermediate servers.
- Computing elements with an emulator for detecting malware.
- Distributed signature database.

This system provides a single-entry point for all users, that is, a server. All users (who have access) will be able to transfer through a web client or with the help of the application to send executable files for checking. It is assumed that the system will have a large number of computing elements, so the server will process incoming files using intermediate servers $(System = \{CS_1, CS_2, \dots CS_i\}$, where CS – intermediate computing system, and i – their number), which will control a certain number of computing devices $(CS = \{CN_1, CN_2, \dots CN_l\}$, where CN – computing element, and l – their number). The choice of an intermediate server is based on the current level of workload at the moment of each of them. After receiving the executable file, the intermediate server randomly distributes among all members of the network a set of states for which each of them is responsible, to cover all the necessary states (Fig. 1.). If a malware is detected, the intermediate server will update the signature database.

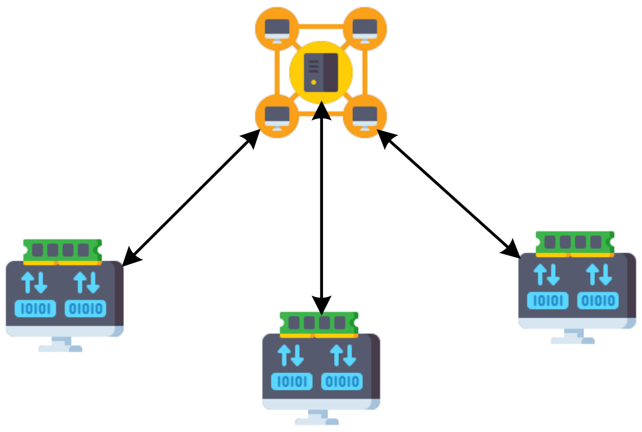

Figure 1: Intermediate computing subsystem

Each computing element during the operation of the system, in a certain period, will return one or more results of checks – imprints. All imprints are collected on the intermediate server, and the results are compared. Since each computing element uses the same emulator, it is expected that the imprint to be the same for all states of the same potential malware.

In general, each subsystem and its computing elements should work synchronously only in the middle of the subsystem itself since each subsystem works independently of each other. Also, for each subsystem, intermediate servers should be duplicated, since in case of failure of any of them, the computing elements should be able to send their results to the additional one.

Let's determine what answers the intermediate server can receive from its entire group:

- The task is completed, the format of the answer is correct.
- The task is completed, the format of the answer is not correct.
- The computing element did not finish task according to the time provided, instead of answering, a special message arrives, the computing element perform test check again, and joins another group with other characteristics, the level of trust in it decreases.
- Computation error notification.
- The answer did not come (a decrease of trust level in database that stores information about existing computing elements is recorded).

Considering the received answers, the system should be able to constantly balance groups of computing elements, this will allow to perform the tasks efficient and secure. In addition, intermediate servers should check each other's work. Since, intermediate servers are part of controlled system, it is possible to use additional security approaches, so it is possible to use some kind of simplified checks. The simplified checking approach basically stands for partial checks, in which each intermediate server will choose a certain part of its completed tasks, and send them to the main server with a special mark. Such tasks will undergo a full cycle of checks by all computing elements of another subsystem. If one intermediate server has completed for example ten tasks, the other intermediate server can choose randomly few of them at random and complete them too, and if the results match both of intermediate servers can trust each other. These checks can also be initiated by the main server.

We can present the whole system as follows:

$$System = \{CN_1 \dots CN_l, PM_1^1 \dots PM_m^1, PM_1^{SB} \dots PM_M^{SB}, SC_1 \dots SC_i, MS, D\}$$

where, $System$ – abstract distributed malware detection system, $CN_1 \dots CN_l$ – computing elements that perform potential malware analysis in different states, $PM_1^1 \dots PM_m^1, PM_1^{SB} \dots PM_M^{SB}$- potential malware ($1 \dots m$ – number of potential malwares, $1 \dots SB$- number of states to check), $SC_1 \dots SC_i$ – intermediate servers that works with CN, MS – main server, D – signature database.

The proposed abstract system will be as follows:

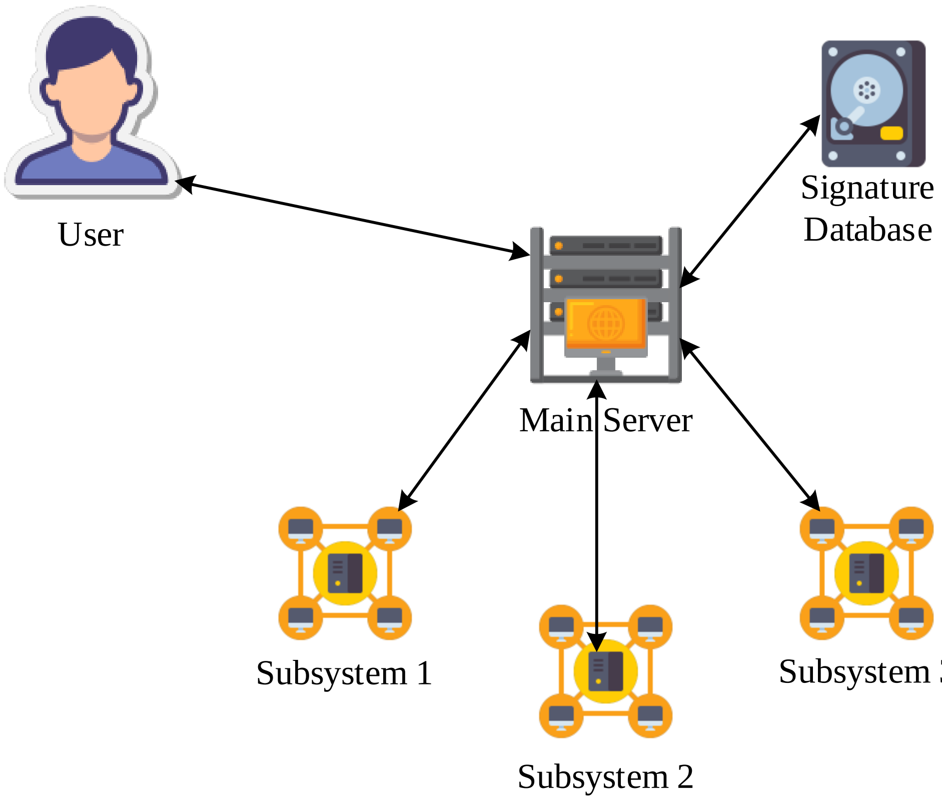

Figure 2: Abstract model of distributed malware detection based on sandbox.

In addition to the overall operation of the system, it is also necessary to describe the process of protecting the execution of calculations. The type of calculations described above involves the use of voluntary participants who propose to attract their computing resources to solve the problem. Although, this approach solves the problem of the necessary computing resources, it imposes one limitation, namely: complete trust in any member of the network is impossible. And this situation arises due to the inability to control all computing elements, so it is important to provide options for how to protect the results of calculations themselves from distortions of those interested in compromising such systems. Another important task is to protect intermediate servers from various types of attacks, in particular, it is necessary to pay attention to protection against DDoS attacks. The stability of each such element of the system affects the performance of the whole system. In paper [23], the difficulty of detecting the type of such attacks is considered, and their types are presented. Paper [24] presents a model that uses ML in real time to detect DDoS traffic. Such tools are useful to consider and implement as an additional module for the protection of a distributed computing system. Also, since the system is defined as heterogeneous, it will be necessary to identify the mechanisms for the correct distribution of tasks between the participants, taking into account their computational resources.

4. Evaluation performance efficiency of computational tasks

Considering the heterogeneity of the proposed model [25,26], attention should also be paid to the correct distribution of the complexity of the tasks. The first task is to try to identify its current computing elements. The system can store certain information about them, such as IP address, processor model, number of physical and logical cores, amount of RAM, etc. You can supplement this data with behavioral features, system time, session duration, number of completed tasks and so on. By combining both sets of knowledge, you can try to identify each user the next time when it connects to the system. This is important for several reasons, in terms of the efficiency of using computing resources. The problem of stabilizing the system [27] in such conditions is very important, and it is these methods that will help to solve it in some aspects.

When adding a new computing element to an intermediate server, the following actions are performed:

1. The system records all possible characteristics about the new computing element.
2. Sends a set of already completed tasks with varying complexity.
3. Records the results of performed tasks and sets assessment of efficiency.
4. Checks the correctness of completed tasks, and if all tasks are completed correctly (assigns a trust level of computing element as 50%.
5. Adds a new element in specific group (algorithmically chooses the best group: chooses a group where there are not enough computing elements, or with an excess of such in other group – forms a new group, taking into account the rate of delay in packet transmission).
6. At a certain time interval, updates basic user information to have the most accurate data about computing element.

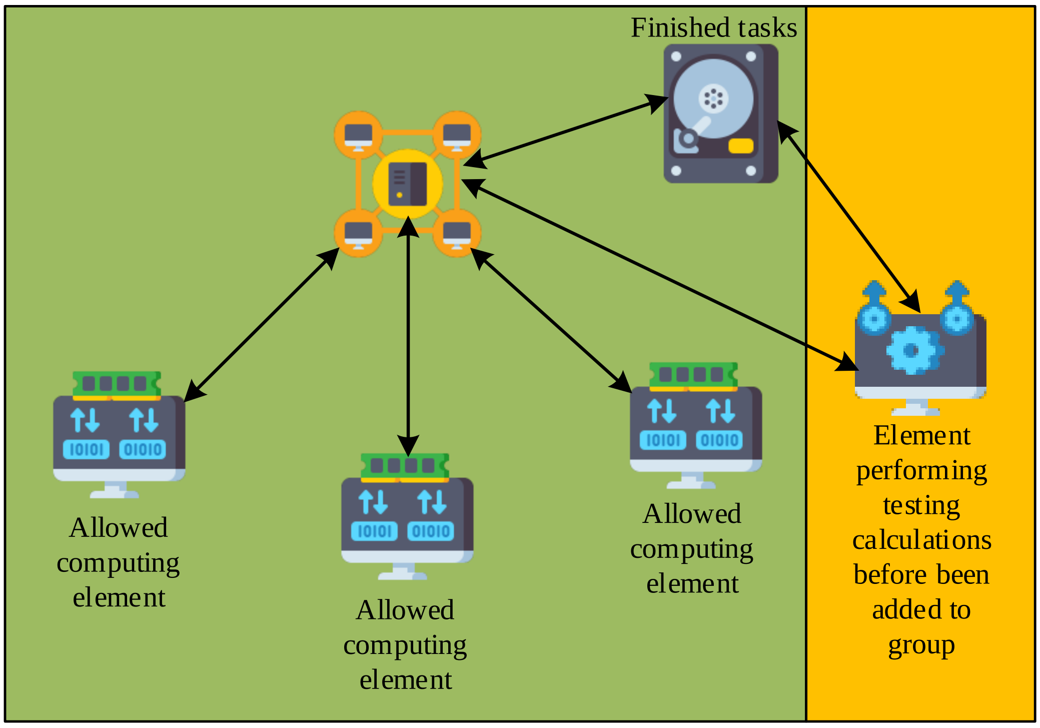
Finished tasks
Allowed computing element
Allowed computing element
Allowed computing element
Element performing testing calculations before been added to group

Figure 3: Checking a new computing element that connects to the system.

Considering that the system cannot hope for regular cooperation with each computing element, it is necessary to use all possible ways to reduce the time to check and find the appropriate group for the newly connected element to perform a real task. By recording information about participants, the system can reduce this time. For example, while started working with an already known computing element, reduce the portion of test tasks, or even assign a higher level of trust for it, which can help to balance quickly the group in the number of correctly completed tasks. Test tasks will also help determine the computing resources of new computing element, that connects to the system. These details can be used in order to find a better subsystem to take part into computing. In some cases, amount of computing resources that provided by element may change even in one session. On this case with proposed algorithm will help to detect this and will find the better group for computing, if it is needed.

5. Features of the protection of distributed computing in a heterogeneous environment.

To organize the protection of the chosen computing model, various approaches are proposed, and one of them is based on the trust model [28]. Its use is appropriate for several reasons: the system cannot manage its participants; the system does not have access to the list of the installed software and used hardware on the side of the computing element, and as a result cannot control the process of performing the calculation. So, the system cannot hope for a guaranteed computed result. That is, the system does not trust any connected computing element to the system. The trust model is based on the rating of each client, which is based on many factors, depending on the specific implementation. Examples of such factors can be the number of correctly completed tasks, the ratio of the number of correctly completed tasks to incorrectly completed ones, the total time of successful work in the system, and so on. This approach allows you to solve certain problems and, with the appropriate configuration, provide the necessary level of protection for both the entire system and the results of the calculation.

The rating model for organizing the protection of the presented computing system is a good practice [29] and is used in many modern tasks [30], since when using tools for controlling and changing the rating of users in the system can solve the issue of security of distributed computing and to a certain extent help with the effective distribution of tasks between clients. A key solution in the issue of computational protection is based on usage of replication approach [31, 32], which is aimed at selecting number of computing elements for one task. As a result of execution, the server or intermediate server will track the results of the calculation and, based on the result of computing elements voting, make decisions on the correctness of the task as a whole group, relative to each user. The obtained data must be converted into a rating, which may change throughout the computing element work in the system. On the other hand, computing element's rating will allow him to be evaluated as a whole, and plan future calculations accordingly. It will also reduce the cost of recalculation.

For example, at the beginning of work, the system cooperates with computing elements, after a some period of time, all of them received a positive assessment of their work (most of the tasks were completed correctly), so the system make a decision to divide this group into two smaller groups, each of which contains $n/2$ computing elements, and now the system can simultaneously perform twice as many calculations per unit of time. Another example of using the rating is the formation of groups of computing elements with low and high ratings, in which case the trusted client will play the role of the controller of calculations of the entire group, that is, when voting, it will have the highest priority and, as a result, influence the result of the calculation.

The use of the trust model in distributed computing issues, that considered in this paper is appropriate, which is why it is proposed to use an additional analysis in addition to classical calculation checks to improve the accuracy of its work. Authorization of the computing element in the system does not give us a full guarantee of his ability to correctly and quickly perform the computing, so it is proposed to use other information that the user leaves about oneself. During the adding to system, the server can record data about the user's address, a unique identifier of the hardware and its features, language, connection time, average working session time, etc. The data set may vary depending on the technology stack used in designing of the computing system.

Behavioral features of the computing element – a set of data about the user that characterize his work during the calculations. The main idea of using such a characteristic is to constantly record information about the user's activity in the system, taking into account his results in voting (Fig. 4.)

Voting settings will allow correctly distribute computing elements between subsystems. In case all customers send different results, it may be advisable to break down the current group and add them all in different subsystems. It is also needed to consider the situation when all trusted computing elements provide one computed answer, and other elements in the subsystem another computed answer. In this case, the trusted elements may be compromised, so the one of the possible scenarios may be: breaking down the current group and set usual status for trusted elements.

It is also necessary to consider the situation when the computational element returns constantly incorrect answers, which may mean the impact on it of viruses that affect the result, thereby slowing down the entire system. In this case, it is necessary to provide mechanisms for blocking its future participation in the system.

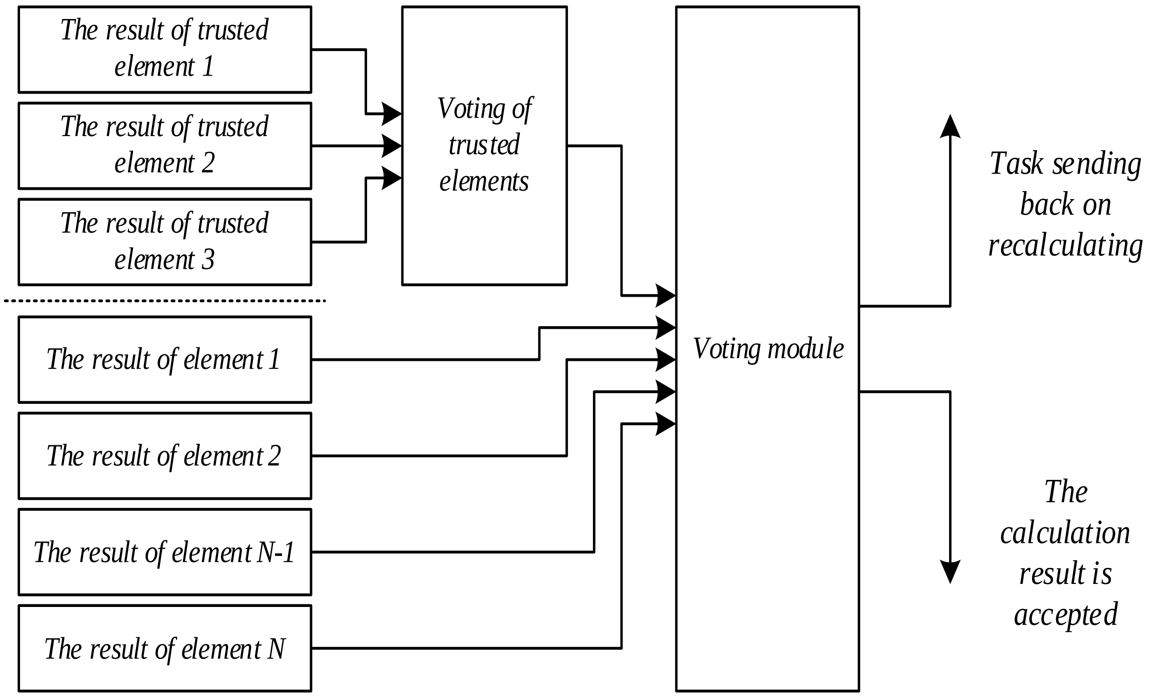
The result of trusted element 1
The result of trusted element 2
The result of trusted element 3
Voting of trusted elements
The result of element 1
The result of element 2
The result of element N-1
The result of element N
Voting module
Task sending back on recalculating
The calculation result is accepted

Figure 4: Computing element voting model.

Usually, when organizing such calculations, the system forms groups of elements with one or more trusted elements. Such clients have the highest trust rating and can significantly influence the outcome of voting. Not only the success of the calculation, but also the rating of all elements in the current iteration of calculations will depend on the result of the vote. Therefore, it is advisable to include three or more trusted clients in the groups so that they first vote among themselves on their calculations, and then consider the results of other users.

Figure 5 shows an abstract computational module, it consists of a voting module that will evaluate the obtained results, a module for storing performed calculations that will be used to add new computing elements, and a computational planning module, the main task of which will be to plan the number of cycles required to perform one task, which will depend on the current state of the computing resources of the entire subsystem.

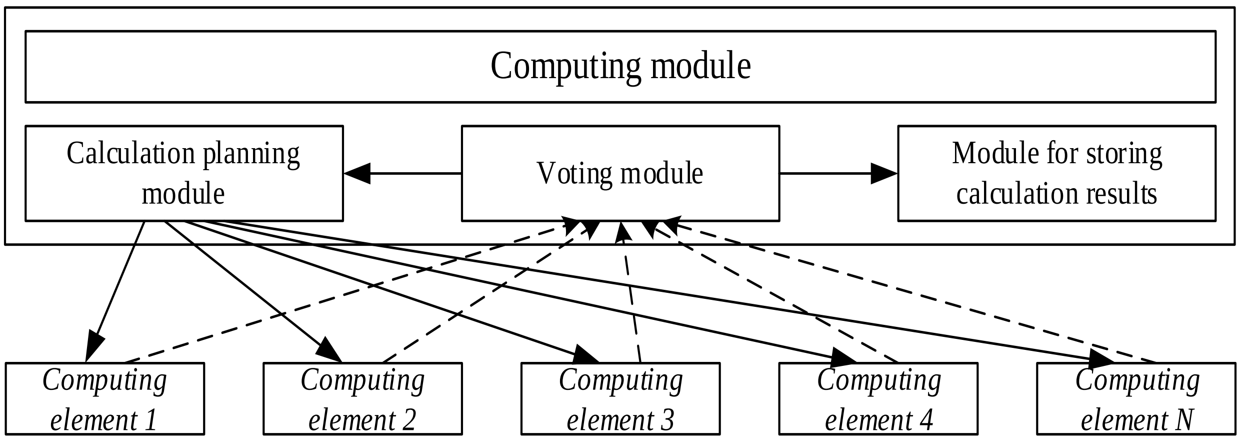

Figure 5: Computing module

The proposed methods and tools will help to solve the issue of protection of needed computing tasks, considering that the system will work with dynamic topology. Basic algorithms for adding elements in computing groups will help to balance the system, because in one hand new element in an already working group will not be able to significantly affect the results of computing with elements with a good trust level, and in another hand the existing elements can either use its computing resources or exclude it from the system by voting. Thus, the described measures will be able to ensure the correct implementation of the tasks for the detection of malware.

6. Experimental research

Presented methods for organizing distributed malware detection provide fault-tolerant working. As part of this work, detection is considered in a limited form, and needs more detail. The sandbox is currently under active development and records changes in its state due to the formation of hash values of all registers. In further work, it is planned to transfer the right to decide on the anomaly of the set of commands to an intermediate server, which will collect the hash values of all computing elements in its subsystem. The voting system at this stage of the system's operation does not consider the rating of computational elements, the decision is made on the basis of most of the answers received are true or false. This aspect requires detailed research defining an algorithm for updating the rating after completing the task, and adjusting it after providing the wrong answer, abnormal behavior, etc.

7. Conclusions

This paper presents a model of system for malware detection, based on the use of distributed computing technology to analyse the state of the sandbox. To identify it, it is proposed to use the concept of the state of the sandbox and monitor its state during the execution of executable files in them. The concept of sandbox states is considered, and ways of its use in malware detection issues are presented. The proposed model eliminates the issue of the computational complexity of the use of sandboxes, by using distributed computing based on heterogeneous computer systems. The main tasks of ensuring the stability and security of calculations, considering the dynamic topology of system, are presented it is proposed to use voting methods and a rating system. It is proposed to consider the identification of users based on behavioural characteristics and their basic information in order to involve them more quickly in the execution of calculations and with the provision of an appropriate level of protection for computing inside system.

8. References

[1] Cyber Threat Report. Sonicwall 2022, 2022. URL: https://www.infopoint-security.de/media/2022-sonicwall-cyber-threat-report.pdf

[2] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, A.Nicheporuk, A Technique for detection of bots which are using polymorphic code, Computer Networks: 21st International Conference, Proceedings 21 ISSN: 1865-0929 (2014). 265-276. doi: 10.1007/978-3-319-07941-7_27

[3] Ö. Aslan, R. Samet, Investigation of Possibilities to Detect Malware Using Existing Tools, IEEE/ACS 14th International Conference on Computer Systems and Applications (2017) 1277 – 1284. doi: 10.1109/AICCSA.2017.24.

[4] S. Talukder, Tools and Techniques for Malware Detection and Analysis, arXiv preprint (2020). doi: 10.48550/arXiv.2002.06819.

[5] X. Gao, C. Hu, C. Shan, W. Han, MaliCage, A packed malware family classification framework based on DNN and GAN, Journal of Information Security and Applications 68 (2022). doi: https://doi.org/10.1016/j.jisa.2022.103267.

[6] J. Singh, J. Singh, A survey on machine learning-based malware detection in executable files, Journal of Systems Architecture 112 (2021). doi: https://doi.org/10.1016/j.sysarc.2020.101861.

[7] C. Raghuraman, S. Suresh, S. Shivshankar, R. Chapaneri, Static and Dynamic Malware Analysis Using Machine Learning, First International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI Springer Singapore (2019) 793-806. doi: https://doi.org/10.1007/978-981-15-0029-9_62

[8] O. Pomorova, O. Savenko, S. Lysenko, A. Nicheporuk. Metamorphic Viruses Detection Technique based on the the Modified Emulators, ICTERI ISSN: 1613-0073 (2016) 375-383.

[9] S. Talukder, Z. Talukder, A survey on malware detection and analysis tools, International Journal of Network Security & Its Applications (IJNSA) Vol. 2020 (2020) 37-57. doi: 10.5121/ijnsa.2020.12203.

[10] O. Savenko, S. Lysenko, A. Nicheporuk, B. Savenko, Metamorphic Viruses' Detection Technique Based on the Equivalent Functional Block Search, ICTERI (2017) 555–569.

[11] Savenko, O., Nicheporuk, A., Hurman, I., Lysenko, S., Dynamic signature-based malware detection technique based on API call tracing, ICTERI Workshops ISSN: 1613-0073 (2019) 633-643.

[12] O. Savenko, S. Lysenko, A. Nicheporuk, B. Savenko, Approach for the Unknown Metamorphic Virus Detection, Proceedings of the 8-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, (2017) 71–76. doi: 10.1109/IDAACS.2017.8095052

[13] CV Liţă et al, 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) 107-126. doi: 10.1007/s11416-017-0291-9

[14] SA Ebad, et al, Measuring Software Obfuscation Quality, A Systematic Literature Review, IEEE Access 2021 (2021) 99024-99038. doi: 10.1109/ACCESS.2021.3094517.

[15] T. Korkishko, A. Melnyk, Cryptographic processor architectures for DES algorithm, 1999 IEEE Africon. 5th Africon Conference in Africa (Cat. No.99CH36342) vol.1 (1999) 175-180. doi: 10.1109/AFRCON.1999.820788.

[16] S. Liu, P. Feng, S. Wang, K. Sun, J. Cao, Enhancing malware analysis sandboxes with emulated user behavior, Computers & Security 115 (2022) 102613. doi: 10.1016/j.cose.2022.102613

[17] G. P. Kachare, G. Choudhary, S. K. Shandilya, V. Sihag, Sandbox Environment for Real Time Malware Analysis of IoT Devices, Computing Science, Communication and Security: Third International Conference, COMS2 2022 (2022) 169-183. doi: 10.1007/978-3-031-10551-7_13

[18] F. Alhaidari, NA Shaib, M. Alsafi, H. Alharbi, M. Alawami, R. Aljindan, Atta-ur Rahman, Rachid Zagrouba, ZeVigilante: Detecting Zero-Day malware using machine learning and sandboxing analysis techniques, Computational Intelligence and Neuroscience 2022 (2022). doi: 10.1155/2022/1615528

[19] J. N. Odii, J. A. C. Hampo, F. O. Nwokoma, and T. U. Onwuama, Comparative Analysis of Malware Detection Techniques using Signature, Behavior and Heuristic, International Journal of Computer Science and Information Security (IJCSIS) 17, no. 7 (2019) 33-50.
[20] A. Khalimov, S. Benahmed, R. Hussain, S.M.A Kazmi, A. Oracevic, F. Hussain, F. Ahmad, CA Kerrache, Container-based sandboxes for malware analysis: A compromise worth considering, Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing (2019) 219-227. doi: 10.1145/3344341.3368810
[21] C. Hagleitner, D. Diamantopoulos, B. Ringlein, C. Evangelinos, C. Johns, R. N. Chang, B. D'Amora, J. A. Kahle, J. Sexton, M. Johnston, E. Pyzer-Knapp, C. Ward, Heterogeneous Computing Systems for Complex Scientific Discovery Workflows, 2021 Design, Automation & Test in Europe Conference & Exhibition (2021) 13-18. doi: 10.23919/date51398.2021.9474061.
[22] Folding@Home Network Breaks the ExaFLOP Barrier In Fight Against Coronavirus, 2020. URL: https://www.tomshardware.com/news/folding-at-home-breaks-exaflop-barrier-fight-coronavirus-covid-19
[23] N. A. Ignatev, E. R. Navruzov, Estimates of the Complexity of Detecting Types of DDOS Attacks. International Journal of Computing, 21 4 (2022) 443-449. doi: 10.47839/ijc.21.4.2779
[24] R. S., Kanavalli, A. Gupta, A. Pattanaik, S. Agarwal, Real-time DDoS Detection and Mitigation in Software Defined Networks using Machine Learning Techniques, International Journal of Computing, 21, 3 (2022), 353-359. doi: 10.47839/ijc.21.3.2691
[25] K. Censor-Hillel, R. Gelles, B. Haeupler, Making asynchronous distributed computations robust to noise, Distributed Computing 32 (2019) 405-421. doi: 10.1007/s00446-018-0343-5.
[26] M. Dinitz, J.T. Fineman, S. Gilbert, C. Newport, Smoothed analysis of dynamic networks, Distributed Computing, 31 (2017) 273-287. doi: 10.1007/s00446-017-0300-8
[27] C. Lenzen, J. Rybicki, Near-optimal self-stabilising counting and firing squads. Distributed Computing 32.4 (2019) 339-360. doi: 10.1007/s00446-018-0342-6
[28] N. Ramu, P. Vijayakumar, DL Jegatha, R. Sivakumar, A novel trust model for secure group communication in distributed computing, Journal of Organizational and End User Computing (JOEUC) 32, no. 3 (2020). 1-14. doi: 10.4018/JOEUC.2020070101
[29] W. She, Q. Liu, Z. Tian, J.-S. Chen, B. Wang and W. Liu,, Blockchain trust model for malicious node detection in wireless sensor networks, IEEE Access 7 (2019) 38947-38956. doi: 10.1109/ACCESS.2019.2902811
[30] S. Guo, X. Hu, S. Guo, X. Qiu, F. Qi, Blockchain meets edge computing: A distributed and trusted authentication system, IEEE Transactions on Industrial Informatics 16, no. 3 (2019) 1972-1983. doi: 10.1109/TII.2019.2938001
[31] PS. Almeida, C. Baquero, Scalable eventually consistent counters over unreliable networks, Distributed Computing 32, no. 1 (2017) 69-89. doi: https://doi.org/10.1007/s00446-017-0322-2
[32] S. Slimani, T. Hamrouni, F.B. Charrada, Service-oriented replication strategies for improving quality-of-service in cloud computing: a survey, Cluster Computing 24 (2021) 361-392. doi: 10.1007/s10586-020-03108-z