ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC 004.492.3 **doi: 10.32620/reks.2024.1.13**

Antonina KASHTALIAN[1], Sergii LYSENKO[1], Oleg SAVENKO[1], Andrii NICHEPORUK[1], Tomas SOCHOR[2], Volodymyr AVSIYEVYCH[1]

[1] ***Khmelnitsky National University, Khmelnitsky, Ukraine***
[2] ***Prigo University, Havirov, Czech Republic***

MULTI-COMPUTER MALWARE DETECTION SYSTEMS WITH METAMORPHIC FUNCTIONALITY

The need to develop new systems for detecting and counteracting malware remains relevant. In addition to malware detection methods, the need to develop new systems for detecting and counteracting malware has become increasingly important. The use of various detection systems and the formation of a variable architecture in them significantly improves the effectiveness of detection, since both for attackers in computer attacks and for malware, understanding the system is significantly complicated. In addition, such systems may contain baits, traps, and, accordingly, modifiable operating environments to deceptively execute programs for research. This paper develops a conceptual model of multicomputer systems, which is designed to ensure the functioning of antivirus bait and traps to detect malware and computer attacks in corporate networks. The proposed approach is intended to prevent and counteract metamorphic virus penetration. This paper presents the conceptual model of multi-computer systems and introduces a defining characteristic responsible for the control of decisions and other defining characteristics of the system. Methods for detecting metamorphic viruses with the possibility of their implementation in the architecture of multi-computer systems with bait and traps are developed so that the system directly joins the detection procedure through its components and decides on the presence of metamorphic code in the executable file. An implementation of a multi-computer malware detection system with metamorphic functionality was developed to prove the feasibility of the proposed conceptual architecture model and the developed methods for detecting metamorphic viruses. An experiment on the functioning of a multi-computer malware detection system was set up, and experimental studies were conducted. The conducted experiments included metamorphic virus detection. In addition, an experiment on the effectiveness of detecting the metamorphic code of viruses was conducted. The efficiency of detecting metamorphic virus code using the developed multi-computer system was also investigated, and the presence of improved detection was established. The directions of further work are to extend the results of this work to new types of malware.

Keywords: *metamorphic code; multi-computer systems; cybersecurity; computer viruses; malware; malware detection.*

1. Introduction

1.1. Motivation

The creation and distribution of malware continues apace. Attackers build various functionalities into their architecture, including those that ensure the concealment of malicious code. Computer viruses that are designed in such a way that they contain obfuscation mechanisms and use them to embed into executable PE files are quite difficult to detect. Each time they are embedded in executable PE files, they change the location of their commands, the order of placement, and replace them with alternative command. The variety of such techniques used to avoid detection is quite large. Therefore, for each of these techniques, it is necessary to develop separate methods for detecting viruses. The use of methods based on signature search is possible as an additional tool. It is more important to search for new viruses with obfuscation functionality. A set of such computer viruses is the set of metamorphic viruses. Methods using obfuscation analysis are effective in detecting metamorphic viruses [1]. However, attackers are constantly improving mechanisms based on metamorphic transformations; therefore, the development of methods for detecting such viruses continues. At the same time, there is a need for fundamental changes in their development so that they can improve detection efficiency and significantly outpace the improvement of metamorphic techniques developed and used by attackers. Such detection methods should provide proactive detection of new metamorphic techniques that could be used in metamorphic viruses by attackers.

Existing methods for detecting metamorphic viruses could be improved and enhanced by implementing them in distributed systems to organize their functioning within computer networks. Such technologies have been used and presented in [2, 3]. Then, in order to improve the detection of metamorphic viruses, if the place of de-

tection is considered to be the nodes of a computer network, it is necessary to improve and simultaneously develop two components that are used in the detection process. On the one hand, these detection systems must operate in computer network nodes and their components are located in the nodes. In addition, to maintain the integrity of the system, such systems have to respond in conjunction with specialized functionality responsible for detecting metamorphic viruses and must be multi-computer. Another possibility of the proposed system is the use of methods for the direct detection of metamorphic viruses. The detection methods should be improved in terms of their possible combination and implementation in a multi-computer system to form one sensor in a computer network. This makes it possible to scale the multi-computer system to all nodes of the computer network and expand the scope of detection methods compared with their use in individual computer stations. In this embodiment, the detection methods have enhanced capabilities, which gives them an advantage over the metamorphic techniques used by attackers in computer viruses.

There are not enough approaches and methods devoted to the development of multi-computer systems for this purpose among scientific researchers. As a rule, researchers on detecting computer viruses in host computer systems or in computer networks, including directly in their nodes, focus mainly on the development of effective detection methods. To implement detection methods in a particular system, method developers pay insufficient attention to the architecture of systems. The architecture of systems should also focus on the specifics of the tasks assigned to the system, since attackers take advantage of the shortcomings of known architectures. In addition, the effectiveness of detection methods can be leveled due to the imperfection of the architecture of the systems in which they are implemented.

Therefore, the aim of this study is to improve the efficiency of detecting metamorphic viruses by developing effective multi-computer systems and methods for their detection.

1.2. Previous works

When designing multi-computer systems for detecting malware and computer attacks, such systems with baits and traps have proven to be quite effective. They can be used to implement and implement detection methods. The number of traps and baits for malicious software and computer attacks is constantly growing, which creates problems for attackers. Such multi-computer systems are becoming multifunctional. Their architectural features are hidden from attackers. This does not allow them to understand their essence and, accordingly, bypass their countermeasures. Among such multi-computer systems, systems with a controller are promising [4]. Thanks to it, a multicomputer system, after developing a decision on a particular event, selects options from those developed, considering its previous decisions on the same events. This makes it possible to confuse the attacker and, accordingly, improve the effectiveness of detecting and countering malware. The principle of synthesizing such multi-computer systems is presented in [4].

The operating environment in each component of the system is taken as a trap in such multicomputer systems. Each operating environment in different components of the system is different. Accordingly, the research to detect metamorphic viruses is carried out in parallel in different nodes of the computer network, which in particular improves the time efficiency. In addition, different environments allow different variants of metamorphic transformations embedded in the architecture of metamorphic viruses to manifest themselves.

Let's consider methods of using baits and traps implemented in multi-computer systems for detecting and counteracting malware and computer attacks.

1.3. State of the art

The purpose of deception systems and their components is to improve attack prevention, attack detection, and mitigate the effects of successful attacks. Deception systems can be classified according to various features, including the purpose of deception, the level of the deception system, the type of deployment of the deception system [5], the type of deception [6], and the level of behavior and responses of the deception system [7].

The type of deception is characterized by its information structures, actions, duration, and concept. The main types of deception include disturbance, confusion, mixing, involvement, moving target protection, and honey-x.

Disturbance involves the breach of confidential data [8], including for users who send information to an unreliable source [9]. Obfuscation hides valuable information with noise, adding irrelevant data to relevant data [10], and sending confidential and false data through a certain number of nodes [11]. Connectivity prevention methods use the idea of mixing for security and privacy [12]. The active defense strategy involves engaging the attacker in interaction using various approaches, such as using an attack graph to represent the attacker's strategy and the location of baits in the network [13] and modeling the attacker's penetration into the network using a game-theoretic model [14].

Moving target defense and honey-x are more common types of deception than others. Moving target defense uses the concepts of flexibility, changing the attack surface, and random customization, which are called mu-

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

tations or cyber mutations. Among the types of cyber mutations, random host mutations, random route mutations, random port mutations, fingerprint mutations, and variable virtual networks are widely used [15]. In the study [16], we propose a dynamic host mutation architecture based on moving target protection that can actively counteract cyberattacks. The dynamic host mutation strategy includes address mutation, fingerprint mutation, attraction operations, key management, and authentication. To turn end hosts into unpredictable moving targets, methods have been developed to intelligently and randomly transform their IP addresses or ports without reducing network performance [17]. Mutations are also used to improve the security of cloud services, a deceptive defense mechanism based on the idea of a moving target centered on frequent virtual machine migrations, whose strategies are determined based on a signaling game [18]. Some methods have been proposed to determine the optimal time for mutating network addresses [19].

The honey-x type of deception is by far the most common; the class of honey-x deceptive objects includes objects that can be named with the beginning honey-, such as honeypots, honeynets, honeytokens, honeymails, honeyurls, etc.

Baits and traps can be classified in different ways according to their key characteristics [20]. According to the purpose of using the bait, they can be used for research and production. According to the role of the bait, there are client baits that actively initiate interaction with attackers and server baits that passively wait for attacks [21]. According to the level of interaction, the baits are divided into low-, medium-, and high-level baits. Low-interaction baits imitate simple functions of one or more services, thus not requiring significant development and implementation costs. Baits with a high level of interaction can imitate an unlimited set of services for attackers [22, 23]. To optimize bait resources, combined low and high-interaction baits are used [24]. The scalability of the baits reflects their ability to increase the number of bait nodes and traps in the system. The simplest baits are not scalable. However, modern bait and trap systems provide the ability to change the number of nodes according to the needs of the network and the specified parameters. It is proposed to implement a distributed system of baits with their dynamic location. In such a network, an attacker cannot distinguish real services from baits, and efficiency is ensured by a game-theoretic approach [25]. Methods for the dynamic configuration of baits, a strategy for their deployment and maintenance based on machine learning methods that allow the autonomous deployment of baits [26], and a methodology for determining the optimal number of baits in a network [19] are proposed. Such bait systems allow fighting distributed attacks [27] and analyzing data in networks of geographically distributed baits to identify attack patterns and build attacker profiles [28]. By the type of resources used, baits are divided into physical, virtual, and combined. Physical baits and their networks run on physical machines designed to deploy these nodes. Virtual baits are deployed on virtual machines [29, 30]. Combined baits support deployment on physical and virtual machines [31]. According to the availability of the code, baits are open source [32] and closed source.

To protect confidential data, the automated generation of honeytokens is used to analyze user behavior and their interaction with honeytokens [33]. Honey-x objects are also used to protect credentials, including improving the security of hashed passwords, which is done with the help of additional false passwords (honey-words), the use of which allows generating an alarm [34].

A deception system can be applied at different levels, which reflects the level of the protected system to which deception is applied. A deception system can be applied at the network, system, application, and data levels. At the network level, a deception system is designed to deal with network attacks such as scanning, eavesdropping, penetration, and propagation. At the system level, deception techniques are applied to nodes and are intended to combat external and internal malicious activities[35, 36]. The application level involves the use of specialized deception systems developed for specific services and applications, such as web services and databases. Deception systems designed for data protection use various types of false data to attract attackers and are designed to combat data leaks, privacy breaches, and credential theft [38].

The deployment of a deception system characterizes how this system is integrated into the target system [5]. Depending on the method of deployment, there are: built-in deception systems, deception systems installed before the target system, deception solutions added during the operation of the target system, and isolated deception systems. Some solutions offer built-in integration of deception with the system at the development stage. Work [39] proposes a multi-paradigm approach to defining deception tactics during software development, which is implemented by a set of deception objects integrated with system components. Deception systems installed before the target system mostly contain deception nodes placed between the attacker and the target system, which allows the intercepting of network traffic and influencing processes [7]. Deception systems added at runtime integrate with the target system during its operation. Such systems mostly use honey-x objects such as honey-assets, honey-files, honey-passwords, honey profiles, and honey hyperlinks [40]. The most common component of isolated deception systems are baits, which can operate separately from the target system.

The level of behavior and responses of a deception system characterizes the level of deception of the system and objects according to the responses to queries. The deception behavior of a system can vary from the simplest predictable truthful behavior to intellectually deceptive behavior [41]. Systems with truthful behavior always respond to any query with complete "truthfulness", so the answers reflect the actual internal states of the system. In systems with naively deceptive behavior, processes try to deceive the attacker with artificial responses; however, if the attacker knows this deceptive behavior from previous interactions, the deception can be exposed and the attacker is warned about the presence of deception [17, 42]. The best protection can be provided by deception systems that have intellectually deceptive behavior. The functioning of such a system for an attacker does not differ from the functioning of the target system, even if there has been previous interaction. Systems with intellectually deceptive behavior use machine learning and artificial intelligence methods [43]. The article [44] presents the consistency issue and related trade-offs in distributed replicated systems and databases. In the study [45], an overview of cyber threats and vulnerabilities is presented.

A cyberattack detection system based on information-extreme machine learning is presented in the work [46].

Methods and technologies for ensuring cybersecurity of industrial and web-oriented systems and networks are presented in the study [47]. A model and training method for malware traffic detection based on a decision tree ensemble presented in the work [48].

An approach devoted to the problem of malware detection using evolutionary algorithms is presented in the study [49]. Research that highlights IoT malware detection based on control flow graph analysis is presented in [50]. The technique for malware detection via distributed systems is described in the work [51]. In the study [52], the mean of malware detection is the multi-agent systems.

1.4. The purpose and tasks of research

Considering the shortcomings in the methods of detecting metamorphic viruses and the need to improve the architecture of multi-computer systems for detecting malicious software that use baits and traps to improve the efficiency of detecting metamorphic viruses in computer networks and their nodes, it is necessary to conduct research and solve several problems.

The peculiarity of the architecture of multi-computer systems with baits and traps for detecting malware with metamorphic functionality is the implementation of detection methods and, accordingly, its construction as a single sensor for functioning in a computer network.

Therefore, to achieve the goal of improving the effectiveness of metamorphic viruses, the following tasks need to be solved:

1) to identify the features of the synthesized architecture of multi-computer systems with bait and traps for detecting malware through its properties and use them as a basis for building such a system;

2) to develop a conceptual model of the architecture of multi-computer systems with baits and traps for detecting malicious software, considering the possibility of implementing methods for detecting metamorphic viruses, as well as their features as decision controllers;

3) to develop methods for detecting metamorphic viruses with the possibility of their implementation in the architecture of multi-computer systems with baits and traps in such a way that the system directly joins the detection procedure through its components and decides on the presence of metamorphic code in the executable PE file;

4) to develop an implementation of a multi-computer malware detection system with metamorphic functionality to prove the feasibility of the proposed conceptual architecture model and the developed methods for detecting metamorphic viruses;

5) to set up an experiment on the functioning of a multi-computer malware detection system and conduct experimental studies on the process of processing metamorphic code to confirm the possibility of implementing the steps of the developed methods for detecting metamorphic viruses;

6) to set up an experiment on the effectiveness of detecting the metamorphic virus code and conduct relevant experimental studies;

7) to investigate the effectiveness of detecting metamorphic virus code using the developed multi-computer system and determine whether there is an improvement in detection, as well as to determine the directions for further research and development of the proposed solution and its extension to other types of malware.

The aim of this study is to develop a multi-computer malware detection system with metamorphic functionality to improve the efficiency of detecting metamorphic viruses.

The paper structure is as follows: Section 1 presents motivation, previous work, and state-of the art – a brief analysis of the very modern and the latest ideas and methods addressed to solve the problem of malware detection with its advantages and disadvantages. Sections 2 discusses the main idea of the research: the development of multi-computer malware detection systems with metamorphic functionality. Section 3 describes the experimental results of this research. In addition, conclusions present the obtained results of the research.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

2. Multicomputer systems and methods for detecting malware with metamorphic functionality in corporate networks

2.1. Concept of multi-computer systems of combined antivirus bait and trap and decision controller for malware and computer attack detection

To synthesize multicomputer systems for detecting malware and computer attacks in corporate networks, in the architecture of which detection methods can be implemented that corresponds to part of its specialized functionality, let us use the principle of synthesis P of such systems, which was presented in [1]. According to this principle, it is possible to synthesize systems in which detection methods are implemented, and together with them, such systems function as sensors in corporate networks, whose tasks are to detect malware and computer attacks in corporate networks using combined antivirus baits and traps. Such systems can also be used as sandboxes. Such features are supported by the developed architecture of multi-computer systems. The system components have variable environments that provide variable operating environments for the study of computer virus program codes. The variable operating environments provided in the system components should improve the efficiency of detecting obfuscated virus codes. This can be achieved using baits placed in the components. Their use forms variable operating environments that allow virus program codes to be manifested. Such studies are particularly relevant for metamorphic and polymorphic viruses or executable programs infected with them. Let us consider the concept of creating such systems.

The concept of creating multicomputer systems of combined antivirus bait and traps and a decision controller to improve the efficiency of detecting malware and computer attacks in corporate networks is based on a combination of the following defining properties and their synthesis in systems according to the synthesis principle $\mathfrak{P}$ [1], such as:

- variability in the type of system architecture;
- variability of the system centers;
- system adaptability according to changes in external conditions;
- characteristic changes in the system center
- self-organization of the system
- ability to detect malware.

The considered systems are denoted further by systems of class $\mathfrak{S}$.

Let us consider the peculiarities of the synthesis of systems by combining their defining properties. Display graph of the defining characteristics for systems of type $\mathfrak{S}$ in their architecture at the vertices corresponding to the elements of the sets $\mathfrak{V}_i$ ($i = 1,2,\dots,n_{\mathfrak{V}}n_{\mathfrak{V}}$ – the number of subsets) is shown in Fig. 1. Any closed route in the display graph of defining characteristics in the architecture of systems of the class $\mathfrak{S}$ shown in Fig. 1 always includes a vertex $\mathfrak{v}_{10,1} \in \mathfrak{V}_{10,1}$ from the set $\mathfrak{V}_{10,1}$ [1]. This means that the graph reflects the architecture of various systems according to the principle of synthesis $\mathfrak{P}$. Vertices that correspond to the elements of a certain set $\mathfrak{V}_i$ ($i = 1,2,\dots,n_{\mathfrak{V}}n_{\mathfrak{V}}$ – the number of subsets) when defining a closed route can belong to it, that is, from one set, several elements can be included in the route, not just one. This reflects other options in the architecture of systems of the class $\mathfrak{S}$.

For example, the system may not have an exclusively centralized decentralized, or mixed architecture. However, in the variant of mixed architecture with respect to centralization it can also be centralized and decentralized. For example, at certain time intervals, the type of architecture can change to mixed, then to centralized and then return to mixed or decentralized architecture.

Furthermore, the level of centralization and its features may differ. Similarly, the remaining defining characteristics in the architecture of type systems $\mathfrak{S}$ can have the same features. That is, in type systems $\mathfrak{S}$ there may be several elements from a certain set $\mathfrak{V}_i$ ($i = 1,2,\dots,9,11,\dots,n_{\mathfrak{V}}, i \neq 10$, $n_{\mathfrak{V}}$ – the number of subsets). The graph presented in Fig. 1 outside the boundaries of existing edges and vertices may contain other vertices and edges. However, a closed route also covers them and accordingly includes vertices and edges. As a result, the vertices covered by the route reflect the defining characteristics synthesized in systems of the class $\mathfrak{S}$. From the set $\mathfrak{V}_{10}$, only one vertex belonging to this set is included in the route. The remaining peaks are isolated and cannot be included in any route. Thus, the number of systems of class $\mathfrak{S}$ according to the synthesis principle $\mathfrak{P}$ is different, but according to formula (1) all of them are united by the presence of the controller in their architecture. The number of subsets $\mathfrak{V}_i$ ($i = 1,2,\dots,n_{\mathfrak{V}}n_{\mathfrak{V}}$ – the number of subsets) can be different, in particular less than $n_{\mathfrak{V}}$, but the presence of a one-element set $\mathfrak{V}_{10,1}$ and the set $\mathfrak{V}_{11}$ [1] in the direct product of sets is mandatory.

Let us detail the concept of creation systems of type $\mathfrak{S}$ of their conceptual model, which is necessary to specify the features determined by the synthesis principle $\mathfrak{P}$.

2.2. Conceptual model of the architecture of multi-computer systems with combined baits and traps and a decision controller for detecting and counteracting malicious software and cyberattacks

According to the synthesis principle $\mathfrak{P}$ of systems of class $\mathfrak{S}$, a conceptual model of architecture $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ of

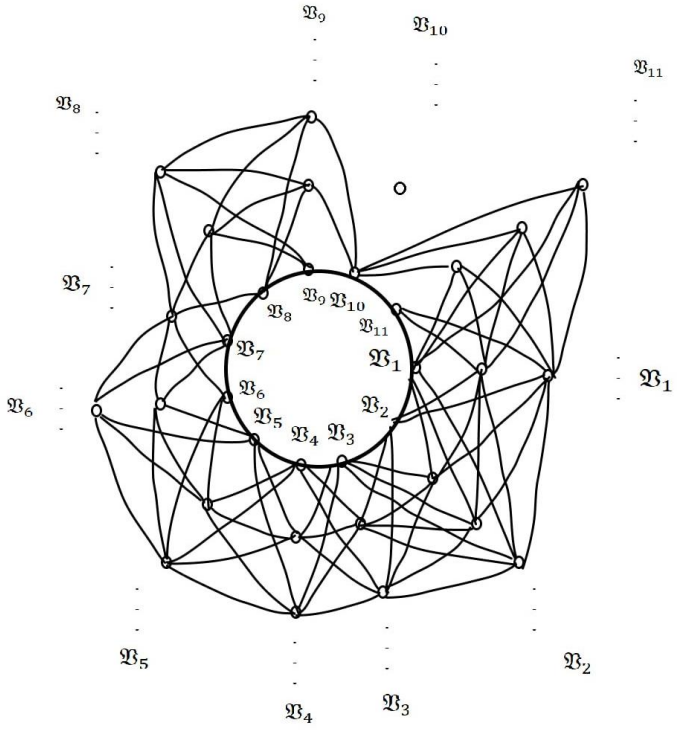


Fig. 1. Display graph of the defining characteristics for in the architecture of systems of the class $\mathfrak{S}$

multi-computer systems with combined baits and traps and the decision-making controller for detection and countermeasures of anti-aircraft and anti-aircraft vehicles, we set as follows:

$$\mathfrak{A}_{\mathfrak{M},\mathfrak{S}} = \langle \mathfrak{P}(\mathfrak{M}_{\mathfrak{S}}), \mathfrak{P}_{\mathfrak{M}} \rangle:$$

$$1)\ \forall\, \mathfrak{k}\colon \mathfrak{V}_{10,1} \subset \mathfrak{P}_{\mathfrak{k}}(\mathfrak{M}_{\mathfrak{S}});\ \mathfrak{P}_{\mathfrak{k}}(\mathfrak{M}_{\mathfrak{S}}) \in \mathfrak{P}(\mathfrak{M}_{\mathfrak{S}});\quad \mathfrak{k} = 1,2,\dots,\mathfrak{n}_{\mathfrak{P}(\mathfrak{M}_{\mathfrak{S}})};$$

$$2)\ \mathfrak{M}_{\mathfrak{S}} = \bigcup_{\mathfrak{i}_1=1}^{\mathfrak{n}_{\mathfrak{M}_{\mathfrak{S}}}} \mathfrak{M}_{\mathfrak{i}_1};\quad \mathfrak{M}_{\mathfrak{i}_1} = \left\{\mathfrak{m}_{\mathfrak{i}_1,1}, \mathfrak{m}_{\mathfrak{i}_1,2}, \dots, \mathfrak{m}_{\mathfrak{i}_1,\mathfrak{n}_{\mathfrak{M}_{\mathfrak{i}_1}}}\right\};\quad \mathfrak{I} = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}_{\mathfrak{i}_1}};\ \mathfrak{M}_{10} = \{\mathfrak{m}_{10,1}\};$$

$$3)\ \mathfrak{P}(\mathfrak{M}_{\mathfrak{S}}) \xrightarrow{\mathfrak{P}_{\mathfrak{M}}} \mathfrak{W}(\mathfrak{M}_{\mathfrak{S}});\ \mathfrak{w}_{\mathfrak{j}} \in \mathfrak{W}(\mathfrak{M}_{\mathfrak{S}});\quad \mathfrak{w}_{\mathfrak{j}} = \left(\mathfrak{w}_{\mathfrak{j},1}, \mathfrak{w}_{\mathfrak{j},2}, \dots, \mathfrak{w}_{\mathfrak{j},\mathfrak{i}_2}, \dots, \mathfrak{w}_{\mathfrak{j},\mathfrak{n}_{\mathfrak{M}_{\mathfrak{v}}}}\right);$$

$$\mathfrak{w}_{\mathfrak{j},\mathfrak{i}} = \begin{cases} 0, \text{if element does not belong} \\ \quad \text{to the set } \mathfrak{P}_{\mathfrak{j}}(\mathfrak{M}_{\mathfrak{S}}), \\ 1, \text{if element belongs} \\ \quad \text{to the set } \mathfrak{P}_{\mathfrak{j}}(\mathfrak{M}_{\mathfrak{S}}); \end{cases}$$

$$\mathfrak{w}_{10,1} = 1;\ \mathfrak{j} = 1,2,\dots,\mathfrak{n}_{\mathfrak{W}(\mathfrak{M}_{\mathfrak{S}})};\ \mathfrak{i}_2 = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}_{\mathfrak{v}}};$$

$$4)\ \mathfrak{G}_2 = (\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,1}, \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,2}, \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3}, \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},2});$$

$$\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},2} \subseteq \left(\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,1} \times \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3}\right) \cup \left(\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3} \times \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,2}\right) \cup \left(\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3} \times \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,1}\right) \cup \left(\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,2} \times \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3}\right);$$

$$\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2} = \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,1} \cup \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,2} \cup \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3};$$

$$5)\ \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1} = \left\{\mathfrak{m}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1,1}, \mathfrak{m}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1,2}, \dots, \mathfrak{m}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1,\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1}}}\right\};$$

$$\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}} = \bigcup_{\mathfrak{I}_1=1}^{4} \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},\mathfrak{I}_1};\ \mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}}} = \sum_{\mathfrak{I}_1=1}^{4} \mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},\mathfrak{I}_1}};$$

$$\mathfrak{F}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1}\colon \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}} \to \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1};$$

$$6)\ \mathfrak{P}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1} = \left\{\mathfrak{p}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}, \mathfrak{p}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}\right\};$$

$$\left(\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K},\mathfrak{I}_3,1}, \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}}\right) \xrightarrow{\mathfrak{p}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}} \mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1};$$

$$\left(\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K},\mathfrak{I}_3,1}, \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}}\right) \xrightarrow{\mathfrak{p}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}} \mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2};$$

$$\mathfrak{I}_3 = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K}}};\ \mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1,0} = \mathfrak{v}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K},\mathfrak{I}_3,1};$$

$$\mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}} = \mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}}};\ \mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}} = \mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}}};$$

$$\text{if } \mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1,\mathfrak{k}_1} = 0\ \forall\, \mathfrak{k}_1 = 1,2,\dots,\mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}}, \text{ то}$$

$$\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2,\mathfrak{k}_1} = 0;$$

$$\forall\, \mathfrak{k}_2 = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1}}\colon \mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1,\mathfrak{k}_2} = 1;$$

$$7)\ \mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v}} = \mathfrak{F}^{\mathfrak{S}}_{\mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v}}}\left(\mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v},\mathfrak{a}}, \mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v},\mathfrak{p}}, \mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v},\mathfrak{v}}\right);$$

$$8)\ \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3} \xrightarrow{\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}} \mathfrak{M}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3};$$

$$\mathfrak{m}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3,\mathfrak{I}_4} = \frac{\sum_{\mathfrak{k}_3=1}^{\mathfrak{n}_{\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}}} \left(\mathfrak{v}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3,\mathfrak{k}_3} \cdot \mathfrak{f}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3,\mathfrak{k}}\left(\mathfrak{m}^{\mathfrak{S}}_{\mathfrak{V},2,3,\mathfrak{I}_4}\right)\right)}{\sum_{\mathfrak{k}_3=1}^{\mathfrak{n}_{\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}}} \mathfrak{v}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3,\mathfrak{k}_3}};$$

$$9)\ \forall\, \mathfrak{g}_1, \mathfrak{g}_2\colon \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},\mathfrak{g}_1} \cap \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},\mathfrak{g}_2} = \emptyset;\quad \mathfrak{g}_1 = 1,2,3,4;\ \mathfrak{g}_2 = 1,2,3,4;\ \mathfrak{g}_1 \neq \mathfrak{g}_2, \tag{2}$$

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

where $\mathfrak{M}_{\mathfrak{S}}$ – the set of elements and components in the system architecture, which are formed according to the given defining properties; $\mathfrak{P}(\mathfrak{M}_{\mathfrak{S}})$ – set of subsets; $\mathfrak{P}_{\mathfrak{M}}$ – the set of predicates given on the set $\mathfrak{P}(\mathfrak{M}_{\mathfrak{S}})$; $\mathfrak{P}_{\mathfrak{k}}(\mathfrak{M}_{\mathfrak{S}})$ – $\mathfrak{k}$-th element of the set $\mathfrak{P}(\mathfrak{M}_{\mathfrak{S}})$; $\mathfrak{k} = 1,2,\dots,\mathfrak{n}_{\mathfrak{P}(\mathfrak{M}_{\mathfrak{S}})}$; $\mathfrak{n}_{\mathfrak{P}(\mathfrak{M}_{\mathfrak{S}})}$ – the number of elements in the set $\mathfrak{P}(\mathfrak{M}_{\mathfrak{S}})$; the set $\mathfrak{V}_{10,1} = \{\mathfrak{v}_{10,1}\}$; $\mathfrak{V}_{10,1}$ – one-element set; $\mathfrak{M}_{\mathfrak{i}_1}$ – $\mathfrak{i}_1$-th subset for certain elements and components in the system architecture, $\mathfrak{i}_1 = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}_{\mathfrak{S}}}$; $\mathfrak{n}_{\mathfrak{M}_{\mathfrak{S}}}$ – number of subsets; $\mathfrak{m}_{\mathfrak{i}_1,\mathfrak{I}}$ – i-th element of the subset $\mathfrak{M}_{\mathfrak{i}_1}$, $\mathfrak{I} = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}_{\mathfrak{i}_1}}$; $\mathfrak{n}_{\mathfrak{M}_{\mathfrak{i}_1}}$ – the number of elements of the subset $\mathfrak{M}_{\mathfrak{i}_1}$; $\mathfrak{W}(\mathfrak{M}_{\mathfrak{S}})$ – set of vectors; element $\mathfrak{w}_{\mathfrak{j}} \in \mathfrak{W}(\mathfrak{M}_{\mathfrak{S}})$, $\mathfrak{j} = 1,2,\dots,\mathfrak{n}_{\mathfrak{W}(\mathfrak{M}_{\mathfrak{S}})}$; $\mathfrak{w}_{\mathfrak{j},\mathfrak{i}_2} – (\mathfrak{j},\mathfrak{i}_2)$ – coordinate of the vector $\mathfrak{w}_{\mathfrak{j}}$, $\mathfrak{i}_2 = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}_{\mathfrak{v}}}$; $\mathfrak{G}_2$ – graph of connections of computer network environment components $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2} = \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,1} \cup \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,2} \cup \mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3}$; $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2}$ – the set of all vertices of the graph $\mathfrak{G}_2$; $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},2}$ – the set of edges of the graph; $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1}$ – set of typical components of systems of class $\mathfrak{S}$; $\mathfrak{m}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1,\mathfrak{I}_2}$ – $\mathfrak{I}_2$- element of the set of typical components $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1}$; $\mathfrak{I}_2 = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1}}$; $\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1}}$ – the number of elements of the set $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{K},\mathfrak{T},1}$; $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}}$ – a set of elements from which the components of the system is formed; $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1}$ – a subset of elements to ensure presence in the node of the corporate network, to ensure the performance of functions specified by certain elements, to supplement with new elements and ensure communication with the center of the system; $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},2}$ – a subset of elements to ensure the functioning of the system center; $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},3}$ – a subset of elements to ensure the functioning of the controller; $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},4}$ – a subset of elements to ensure presence in the node of the corporate network, defined to ensure the performance of functions specified by certain elements, and to supplement with new elements and ensure communication with the center of the system; $\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}}}$ – the number of elements in the set $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T}}$; $\mathfrak{F}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1}$ – set of functions; $\mathfrak{P}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1}$ – a set of predicates for establishing information in the system about the state of available active and passive and missing elements in the components; $\mathfrak{p}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}, \mathfrak{p}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}$ – predicates from the set $\mathfrak{P}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1}$; $\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K}}}$ – the number of vectors in the set of vectors $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K}}$; $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1,0}$ – the first coordinate of the vector $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}$, which is the value of the component number from the system; $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K},\mathfrak{I}_3,1}$ - the value of the component number according to the system list, $\mathfrak{I}_3 = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K}}}$; $\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K}}}$ – the number of vectors in the set of vectors $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},\mathfrak{E},\mathfrak{K}}$,; $\mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}} = \mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}}$; $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2,\mathfrak{r}_1}$ – coordinate of the vector $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}$; $\mathfrak{r}_1 = 1,2,\dots,\mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}}$; $\left(\mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}} + 1\right)$ – the number of coordinates of the vector $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,1}$; $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2,\mathfrak{r}_2}$ – coordinate of the vector $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}$; $\mathfrak{r}_2 = 1,2,\dots,\mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}}$; $\left(\mathfrak{n}_{\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}} + 1\right)$ – the number of coordinates of the vector $\mathfrak{v}^{\mathfrak{S}}_{\mathfrak{E},\mathfrak{K},\mathfrak{T},1,2}$; $\mathfrak{F}^{\mathfrak{S}}_{\mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v}}}$ – a function that determines the value of the integrated indicator of the hardware and software of the computer station according to the values of its structural and parametric characteristics, which are taken into account when determining the indicators $\mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v},\mathfrak{a}}, \mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v},\mathfrak{p}}, \mathfrak{I}^{\mathfrak{S}}_{\mathfrak{p},\mathfrak{v},\mathfrak{v}}$ as the arguments of the function; $\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}$ – set of functions $\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3} = \left\{\mathfrak{f}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3,1}, \mathfrak{f}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3,2}, \dots, \mathfrak{f}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3,\mathfrak{n}_{\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}}}\right\}$, $\mathfrak{I}_4 = 1,2,\dots,\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3}}$; $\mathfrak{n}_{\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}}$ – the number of functions in the plural; $\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}$; $\mathfrak{n}_{\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3}}$ – the number of elements of the sets $\mathfrak{M}^{\mathfrak{S}}_{\mathfrak{V},2,3}$ and $\mathfrak{M}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}$; $\mathfrak{y}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3,\mathfrak{f}_3}$ – coefficients for correlation of weights of all functions from the set $\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}$; $\mathfrak{f}_3 = 1,2,\dots,\mathfrak{n}_{\mathfrak{F}^{\mathfrak{S},\mathfrak{V}}_{\mathfrak{V},2,3}}$.

Such a task of the conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ of the architecture of systems of class $\mathfrak{S}$ according to formula (2) considers the multiplicity of elements that are in relations and connections with each other and form a certain integrity and unity of parts. Given formula (2), the relationships between the parts of the system and its internal organization have specific properties that change according to external and internal influences and the purpose of using the system.

In the conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ of the architecture of systems of class $\mathfrak{S}$, components and elements are distinguished. Moreover, some components also contain smaller components and elements. Therefore, the synthesized systems of class $\mathfrak{S}$ are complex. The selected elements of the system in the model have a purpose, a hierarchy is defined between them and they are interconnected. Accordingly, systems of class $\mathfrak{S}$ are synthesized according to their defining properties, which are generally not obtained by combining the properties of their components, and the functional capabilities of the system as a whole are greater than the functional capabilities of its parts. Let us consider the functional orderliness of the system of class $\mathfrak{S}$, i.e. the expressiveness of how the elements, components and hierarchical levels of the system are connected and interact with each other, as well as what properties have integrity formed by them.

A picture of the placement and connections of the elements and components of the model in the generalized architecture of the operating environment of the system of class $\mathfrak{S}$, considering the task of the conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ (formula (2)) is presented in Fig. 2.

Correlation of elements and components of conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$, multicomputer systems of class $\mathfrak{S}$ and

system functioning environment in fig. 2 shows the nodes of the corporate network in which system components can be installed. At the same time, they correspond to the elements of the set $\mathfrak{M}_{\mathfrak{V},2}^{\mathfrak{S}}$ of all vertices of the graph $\mathfrak{G}_2$. The components of the multicomputer systems of class $\mathfrak{S}$ may not be present in all nodes of the corporate network. Thus, the components of multicomputer systems of class $\mathfrak{S}$ in the context shown in Fig. 2 and in formula (2) confirm the possibility of scaling the system by an arbitrary number of elements of the set $\mathfrak{M}_{\mathfrak{V},2}^{\mathfrak{S}}$. Conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ of multicomputer architecture of systems of class $\mathfrak{S}$ is complete, because it covers the defining properties of such systems, the principles according to which they are synthesized, and the relations defined in 1)-9) of formula (2). Confirmation of the synthesis of systems according to conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ is the presence of generally accepted signs of systems: integrity, latency, strengthening of system efficiency due to the set of components, division into parts with connections between them, the presence of properties not characteristic of subsystems and blocks, self-regulation, multivariability, and interaction with the external environment. These features require detail in the context of presenting the organization of the functioning of the systems of class $\mathfrak{S}$.

Let us define the main elements of the conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ of multicomputer systems of class $\mathfrak{S}$:

- conditions of system functioning, which are determined by the nature of interaction between the system and its environment, and between system elements;

- system control capabilities and the composition of controlled system variables are highlighted.

In addition, the purpose of the study of systems of class $\mathfrak{S}$ is to improve their characteristics in the part of the architecture itself and in the part of the specialized functionality, which must be organically combined into a single whole.

In the conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ of multicomputer systems of class $\mathfrak{S}$, a defining characteristic responsible for the control of decisions is introduced. This distinguishes it from the well-known models of multi-computer systems designed to ensure the functioning of antivirus baits and traps to detect malware and computer attacks, as well as to prevent and counteract their penetration. In addition, the model specifies the rest of the defining characteristics that should form the system architecture in the process of functioning of systems of class $\mathfrak{S}$. This action must be performed independently. Synthesis of a set of individual defining characteristics occurs according to a closed route in the graph of the display of defining characteristics in the architecture of systems of class $\mathfrak{S}$ (see Fig. 1). In addition, it identifies specialized functionality that with the general part of the system, forms the system as a single sensor.

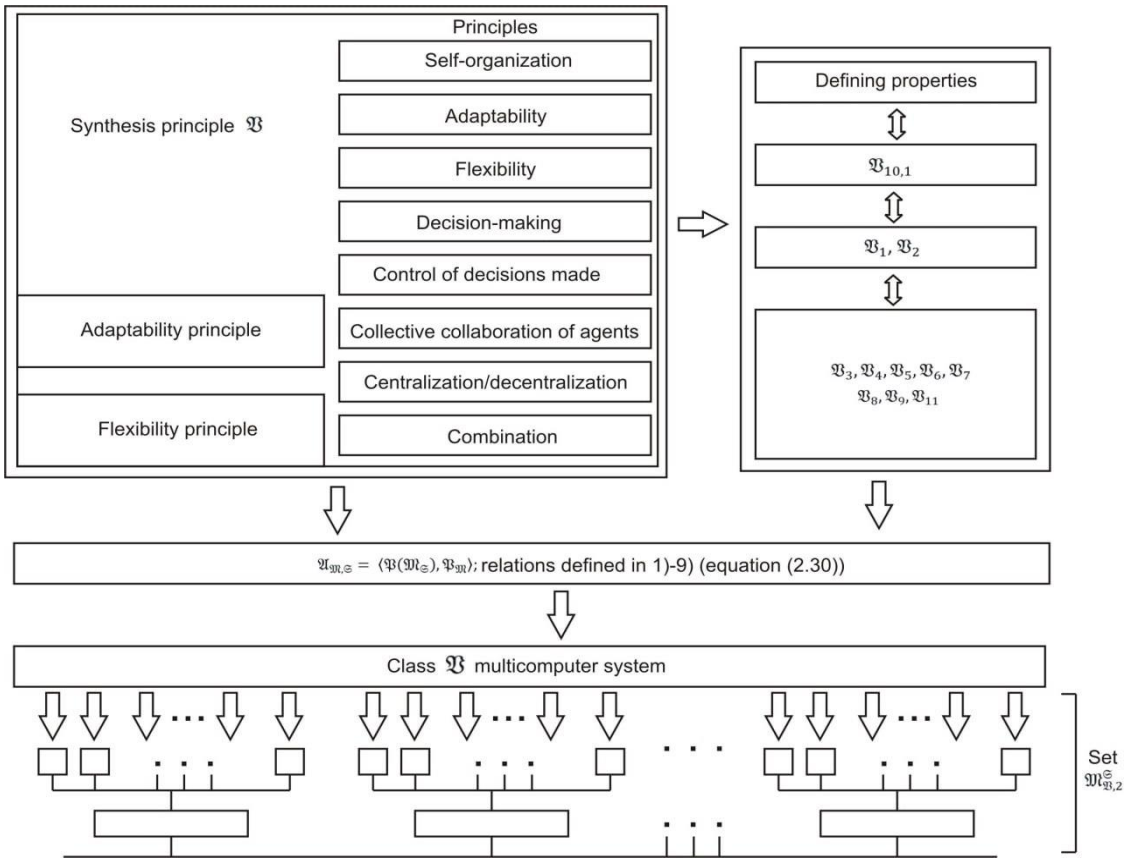


Fig. 2. Correlation of elements and components of conceptual models $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$, multicomputer systems of class $\mathfrak{S}$ and the systems functioning environment

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

The defining characteristic responsible for the control of decisions is made. This is highlighted by a separate element in the conceptual model, which actually defines the property of artificial consciousness with limited capabilities, which are defined by tasks of systems of class $\mathfrak{S}$. This makes it possible to provide various options for responses to the influence of intruders, cyberattacks, and the functioning of malware. As a result, understanding the behavior of the system is significantly complicated.

Other defining characteristics allow you to increase the number of architecture options. Therefore, their use when rebuilding the systems of class $\mathfrak{S}$ allows not only to diversify the system due to its various changing architectures and ensures its stability when removing individual nodes of the corporate network.

The combination of specialized functionality with the general part of the system forms the system as the single sensor, which improves the effectiveness of malware and cyberattack detection.

An important feature of setting the conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ of multicomputer systems of class $\mathfrak{S}$ according to formula (2) is to consider the defining properties. They allow the system to be formed according to certain principles. That is, to perform its actual vertical presentation. This is ensured directly through the presentation of systems in relation to the environment of their operation in corporate networks. That is, its horizontal presentation is performed. These two representations in the model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ are implemented in a coordinated manner. Thus, the conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ adequately reflects systems of class $\mathfrak{S}$ due to the defining properties and environment of operation.

The developed conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ of multicomputer systems of class $\mathfrak{S}$ contains the necessary elements defining models. It is an abstract model that reflects the defining characteristics and features of systems of class $\mathfrak{S}$ regarding the environment of corporate networks. Its special element is the decision controller. It implements a dynamic synthesis of the architecture according to its defining properties. In addition, it implements the combination of specialized functionality with a part of the system into a single sensor. Conceptual model $\mathfrak{A}_{\mathfrak{M},\mathfrak{S}}$ is the basis for creating multicomputer systems of class $\mathfrak{S}$ from combined antivirus bait and traps and the decision controller for the detection of malware and cyberattacks in corporate networks and the implementation of methods for the detection of malware and cyberattacks using combined antivirus baits and traps to improve detection efficiency.

2.3. Detection of metamorphic virus code using multi-computer malware detection systems

The proposed multicomputer system detection of malicious software can implement methods for the detection of metamorphic codes of viruses.

Let us consider the detection process using the developed methods. The process of detecting metamorphic viruses is implemented in the form of two methods, in which the method of detecting the metamorphic code of viruses includes the method of forming a vector of features of the similarity of the code sample to the metamorphic virus. Both methods are based on the concept of comparing copies of metamorphic viruses, the result of which is the definition of a set of features used to detect metamorphic viruses.

Consider the proposed method for detecting metamorphic viruses in a multicomputer system deployed in a local network. The use of the network is dictated by the presence, in addition to obfuscation techniques, of anti-emulation tools that prevent the execution of the emulation process. This is one of the main methods of detecting metamorphic viruses, which, in turn, leads to low detection efficiency. Therefore, it is not always possible to detect metamorphic viruses that use anti-emulation technologies by means of one computer system; therefore, it is suggested to involve a multi-computer system.

Thus, the main task of the system itself is to dispatch data flows between network nodes. The initial settings for each component of a multicomputer system include an isolated virtual environment and a whitelist and blacklist of behaviors.

The method for detecting the metamorphic code of viruses involves the following steps:

1. Analysis of the behavior of executable files on each component using the program's suspiciousness monitor. The check is carried out on the basis of heuristic rules, which are based on the API calls executed by the program.

2. Determination of the need for further research. In the event that suspicious activity was detected on any component of a multi-computer system, a search for matches is carried out in the black and white lists. If the behavior is not in these lists, further investigation of the suspicious program is conducted.

3. Disassemble the suspicious program for execution in an isolated trap environment and obtain a list of opcodes for execution.

4. Execution of the suspicious program in an isolated virtual trap environment, obtaining a list of API calls, redisassembling the suspicious program, and obtaining a list of opcodes after execution.

5. Implementation of a method for forming a vector of signs of similarity of a code sample to a metamorphic virus. As input data for this method, there are listings of opcodes before and after execution, and lists of API calls.

6. Formation in the system component of the result about the degree of similarity of the suspicious program to the metamorphic virus with the involvement of the Fuzzy Inference System using Mamdani Fuzzy Inference

Systems [53]. The result of the work of is the indicator "Level of Similarity to Metamorphic Virus" (LSMV) with the linguistic values *High*, *Medium* and *Low.*

7. Analysis of the formed result and decision-making in accordance with the obtained LSMV indicator:

if the LSMV received a value of *High*, then the suspicious program was blocked on this component, and the behavior of the suspicious program (in the form of a list of API calls) was entered into the black list, followed by updating the black lists on all components of the multi-computer system;

if the LSMV of the suspicious program received the value *Low* - a decision is made about the legitimacy of the suspicious program and its behavior is whitelisted;

if the LSMV receives the *Medium* value, then the most suspicious program is propagated to other system components to run them in isolated virtual trap environments. Waiting for further response from the server.

8. Sequential execution of steps 4-6 in each component of the multicomputer system. Sending a message with the result of the LSMV indicator from each component to the component that initiated the check (initiator).

9. Analysis using the initiator component, obtained from the remaining components of the LSMV indicator system. If at least one value of the LSMV indicator value is *High*, the suspicious program is blocked on the component that initiated the additional analysis of the suspicious program, and the behavior of this executable file is included in the black list of all system components. Otherwise, a conclusion is made about the legitimacy of the suspicious program's behavior.

The steps of the method for the metamorphic code detection system are shown in Fig. 3.

2.4. Method for forming a vector of signs of similarity of a code sample to a metamorphic virus

To determine the characteristic features that would allow determining whether a suspicious program belongs to a metamorphic virus, a method of forming a vector of signs of similarity of a code sample to a metamorphic virus is proposed.

The method is based on comparing listings of the suspicious program before and after execution using the Damerau–Levenshtein distance [54].

The method consists of the following steps:

1. Identification of the search location for code fragments with opcode listings before and after execution.
2. Definition of code fragments.
3. Eliminating uncertainty when defining code fragments.
4. Determining the degree of dangerous behavior of a suspicious program based on a comparison of the program's behavior with a database of behavioral patterns.
5. Pairwise comparison of fragments. Formation of a vector of features of the similarity of a code sample to a metamorphic virus based on the comparison of pairs of fragments of a suspicious program before and after execution in an isolated trap environment.

Let's consider the steps of the proposed method in more detail.

The primary task is to localize the search location for code fragments in the opcode listings of the suspicious program before and after execution.

The choice of search location for code fragments is determined according to the following rules:

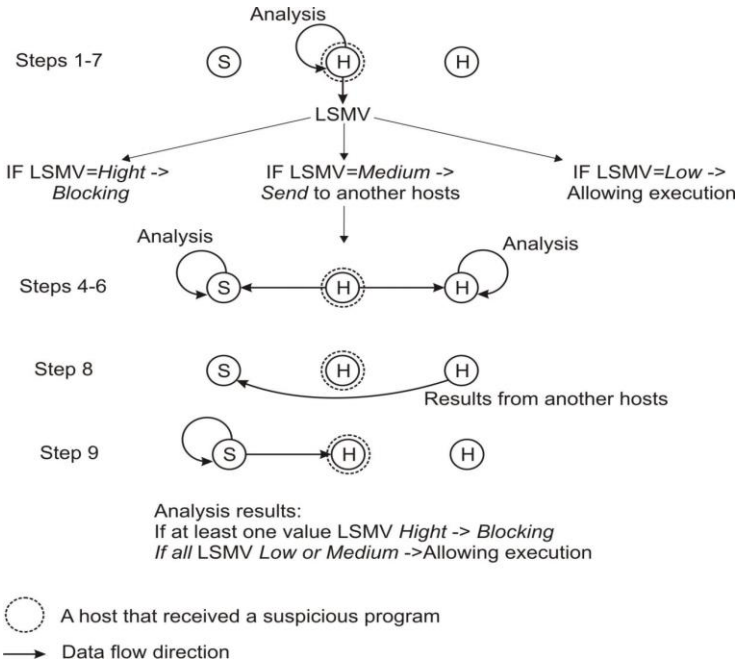


Fig. 3. Steps of the method of detection of the metamorphic code of viruses using a multicomputer system

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

$$f\begin{cases}\text{target\_section, if } \left((s_n \neq .\text{data} \ \vee s_n \neq .\text{code} \ \vee \ldots) \wedge (s_a = executable)\right),\\ \text{target\_section, if } \left(s_{\text{call}} = a_{\text{last}} \ \vee \ s_{\text{jump}} = a_{\text{last}}\right),\\ \text{target\_section} = N_s - 1, \ otherwise,\end{cases} \quad (3)$$

where s_n – the name of the section in which the entry point to the suspicious program is present, s_a – section attribute, s_{call}, s_{jump} – call and jump instructions of the corresponding section, the operand of which contains the address of the last section a_{last}.

After identifying the search location, the next stage of the method involves splitting the opcode listings for code samples before and after execution (we denote these listings as Q and respectively Q′) into code fragments $\{q_l\} \in Q$ and .$\{q'_k\} \in Q'$ based on conditional transition instructions. The TF-IDF metric is used to determine the importance of an opcode in the context of each code fragment in $\{q_l\}$ and $\{q'_k\}$. As a result, two matrices Q and Q′ are formed, in which the rows correspond to the code fragments present in this listing ($\{q_l\}$ or $\{q'_k\}$), and the columns correspond to the opcodes present in the corresponding code fragment.

Each cell of the matrix defines TF-IDF estimation of the appearance of the i-th opcode in the j-th code fragment. Next, to find code fragments from $\{q_l\}$, corresponding to code fragments with $\{q'_k\}$ the following metric is applied:

$$r(\{q_l\},\{q'_k\}) = \sum_{i=0,j=0}(q_{li} - q'_{kj})^2,$$

where q_{li} - the estimate of the occurrence of the i-th opcode in the fragment $\{q_l\}$,q'_{kj} – evaluation of the appearance of the j-th opcode in the fragment $\{q'_k\}$.

If the value of the similarity score of two fragments is less than the threshold value σ, then the similarity score is recalculated for the i-th code fragment from q_{li} and the next j+1-th fragment from q'_{kj}. The value σ is constant, which is selected empirically.

As a result, there may be a situation in which one fragment of code from $\{q_l\}$ corresponds to several fragments from $\{q'_k\}$. Therefore, to eliminate uncertainty and form an unambiguous correspondence between fragments, code fragments from $\{q'_k\}$, which have the maximum evaluation value r, is selected for fragment q_{li}.

All the above-mentioned actions were performed to determine code fragments from $\{q_l\}$ and $\{q'_k\}$ that correspond to each other as closely as possible. Therefore, the last stage of the method is the formation vector of signs of similarity of a code sample to a metamorphic virus. This vector is formed based on a pairwise comparison of the corresponding fragments from $\{q_l\}$ and by the Damerau-Levenshtein distance.

Let us define the feature vector as follows:

$$F = \langle {}^{L_{mod}(E), L_{med}(E), X_{mod}(E), X_{med}(E), D_{mod}(E),}_{D_{med}(E), I_{mod}(E), I_{med}(E), M_{mod}(E), M_{med}(E), Y} \rangle \quad (4)$$

where $E = \{\varepsilon_i\}_{i=1}^{n}$ – pairs of code fragments from $\{q_l\}$ and $\{q'_k\}$; n – the total number of pairs of fragments; L_{mod} – the modal value of the Damerau-Levenshtein metric between ε_i; L_{med} – the median value of the Damerau-Levenshtein metric between ε_i; X_{mod} – the modal value of the number of necessary opcode exchange operations for ε_i; X_{med} – the median value of the number of necessary opcode exchange operations for ε_i; D_{mod} – the modal value of the number of necessary operations of removing operational codes for ε_i; D_{mod} – the median value of the number of necessary opcode removal operations for ε_i; I_{mod} – the modal value of the number of necessary opcode insertion operations for ε_i; I_{med} – the median value of the number of necessary opcode insertion operations for ε_i; M_{mod} – modal value of the number of opcode matches for ε_i; M_{med} – median value of the number of opcode matches for ε_i; Y – the degree of danger of the program's behavior.

To assess the degree of danger of behavior, its behavior is compared with a defined set of harmful behavioral patterns. If there is a match between the actions of the suspicious program and one of the malicious patterns, the similarity vector property for metamorphic viruses takes the suspiciousness value of *Low*, *Medium*, or *High*.

In this way, two methods are presented for detecting metamorphic viruses in a multicomputer system. The first method is based on the identification of code fragments and the formation of a vector with signs of similarity to a metamorphic virus. The second method uses a multicomputer system to coordinate the analysis and share results between components of the multicomputer system. The presented methods make it possible to detect not only known metamorphic viruses but also new ones.

Thus, these two methods were implemented in the architecture of a multicomputer system. Its operation provided several different operating environments for examining programs for the presence of metamorphic virus code. Let's consider the implementation of a multicomputer system and its methods. The implementation of the developed methods and the system itself are necessary for conducting experimental studies on the effectiveness of the proposed solution in terms of improving detection efficiency.

3. Multicomputer malware detection system with metamorphic functionality

3.1. Architecture of the multicomputer system

A multicomputer malware detection system (named AMJS), which consists of a set of interacting nodes - components, was developed to implement the process of detecting metamorphic malware. AMJS is implemented as a set of modules, the main ones of which are:

– AMJSP (Amjs Publisher) - sends level 0 messages (user login/logout, file replacement, administrator installation).

– AMJSS (Amjs Subscriber) – receives level 0 messages (user login/logout, file replacement, admin installation), event logging.

– AMJSAC (Amjs Admin Checker) – Checks if the current admin is active and can be an admin.

– AMJSAP (Amjs Admin Publisher) – sends messages between administrators, works only on hosts selected by administrators.

– AMJSAS (Amjs Admin Subscriber) – receives messages between administrators, sends a request to install a new administrator, logs events, and works only on components selected by administrators.

– AMJSDB (Amjs DataBase) – retrieves data from the database.

– AmjsMPILib - a library of functions for distributed work.

– AMJSMS (Amjs MPI Scheduler) is a task scheduler (sends the executable task file to work participants, starts the task and records its execution data in the database).

– AMJSMR (Amjs MPI Runner) – executable task file.

– AMJSCE (Amjs Config Editor) - receives a message about the need to change the configuration.

Interaction of these modules is shown in Fig. 4. In the process of implementing AMJS, such architectures as Event-Driven architecture (Amjss, Amjsas, Amjsce, Amjs UI modules), distributed architecture (Amjsms, Amjsmr, Amjss, Amjsas, Amjsp, Amjsap modules), client-server architecture (graphic component of Amjs UI), and monolithic architecture (modules amjsp, amjsap, amjsac, amjsjc, amjsdb).

A distributed architecture was used to combine the components into a cluster. Cluster members exchange messages over the network while maintaining a defined system state. Modules amjss, amjas, amjsp, and amjsap provide the exchange of these messages (generate events). The configuration parameters are defined in the fileconfig.ini, including the settings of the cluster name, the number of hosts in the cluster, the prefix of the domain name of hosts in the network, data for MPI work, and ports that were used to establish a connection between hosts.

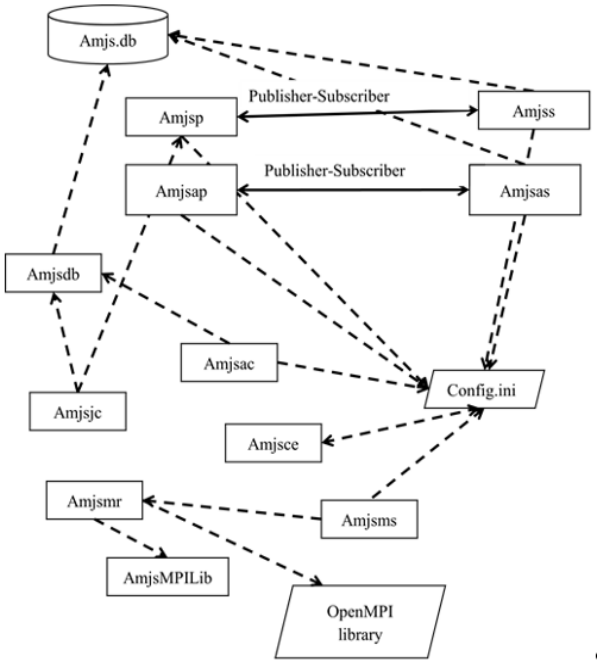


Fig. 4. Architecture of the multicomputer system

The main interaction occurs between amjsp and amjss, amjsap and amjsas modules. These modules receive and send messages about changes in the status of clients, as well as changes in the administrator. Every minute all hosts send these messages. In addition, amjsap sends a message when a new administrator is selected. Pairs of applications work according to the Publisher – Subscriber pattern. Distributed tasks are run using the amjsms module, and the amjsmr program is an executable for MPI. Functions from the AMJS MPI Lib library are used to perform tasks. Before the job, the amjsvc program is run, which clears the job table if the oldest job was more than a week ago.

The amjsce daemon service is always running on the AMJS system, waiting for configuration change notifications. It also checks every minute whether the current administrator meets the requirements.

The interface window displaying the status of hosts in the system is presented in Fig. 5.

3.2. Implementation of the detection function of metamorphic viruses in a multicomputer system

The main function assigned to the developed multicomputer system AMJS, is the identification of the malware metamorphic code.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

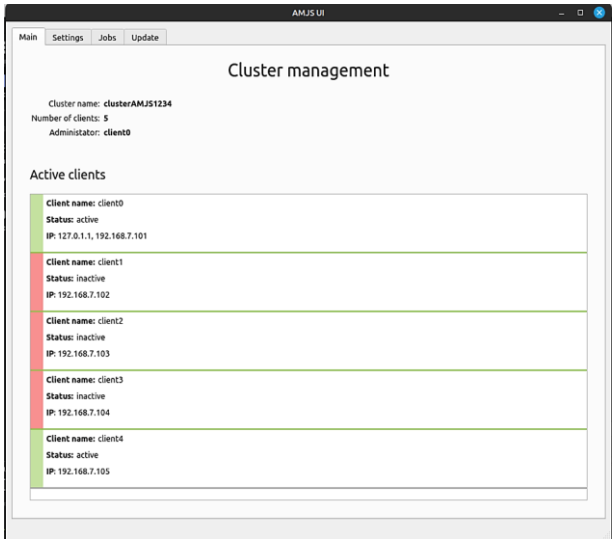


Fig. 5. Interface window displaying the status of the hosts in the system

For this purpose, the method of detecting the metamorphic code of viruses and the method of forming a vector of signs of the similarity of a code sample to a metamorphic virus were implemented in the AMJS system.

The basis of the proposed method is the concept of comparing copies of metamorphic viruses by their opcodes, which involves obtaining two disassembled listings of opcodes - before and after the execution of the suspicious program. This process was achieved by creating an isolated environment, passing the suspicious file to it, and running it for execution. The functionality of the isolated environment is implemented using a Docker container built on the Ubuntu 22.04 image. To simplify the work with the Docker container, management was implemented through the docker command interface, using the system function. The GDB [54] disassembler was used to perform the disassembly.

The implementation of data exchange functions between the components of the AMJS multicomputer system was implemented using the MPI interface. MPI_Recv and MPI_Send functions were used for data exchange between processes.

Each process has a rank from 0 to the number of running processes per task, inclusive of 1. To obtain all results, a process with rank 0 is used. The first stage of transmission involves sending a list of API functions, and the second stage involves the listing of opcodes.

In turn, we perform symmetric actions on all processes whose number is different from 0: disassembly of the binary file, sending to process 0 the list of API functions and the listing of opcodes.

After the container with the isolated virtual environment is started, the suspicious file is copied to it, it is started, and the new executable file is copied again to the host system. Furthermore, disassembly was again carried out with the result being sent in the form of a listing of opcodes after execution to the main process. Thus, the distributed task of analyzing a binary file for similarity to a metamorphic virus is performed.

Thus, a multi-computer system for detecting malicious software with metamorphic functionality has been implemented. This implementation corresponds to the concept and conceptual model of multicomputer systems (formula (2)).

Experimental studies were conducted to confirm the improvement of the effectiveness of detection of metamorphic viruses in computer networks.

4. Experiments

4.1. Setting up the experiment and its results for the functioning of the developed multi-computer system

The main purpose of the experiment was to check the effectiveness of the detection of MWOR metamorphic malware using the developed multi-computer system AMJS.

The experiment included a series of tests in which the metamorphic malware was launched on one of the components (the task initiator) of the AMJS system. Every time on the initiator component, the analysis procedure of this file was started according to the method of detection of metamorphic viruses.

With the help of the GDB disassembler [54], the malware was disassembled and a sample code was obtained to be executed in an isolated virtual environment (in the form of a list of opcodes and a set of API functions). To obtain a sample code, after execution, the malware was launched in an isolated environment. After executing the executable file of the malware in an isolated environment, its repeated disassembly was performed.

Next, the formation of the vector of signs of similarity of the code sample to the metamorphic virus was carried out based on the implemented method of forming the vector of signs of similarity of the code sample to the metamorphic virus.

Then, with the help of the fuzzy inference system built into the proposed system [53], the identification of malware as a metamorphic virus was carried out.

If the indicator “Level of Similarity to Metamorphic Virus” (LSMV) received a value of *Medium* (that is, it was not possible to uniquely identify the test sample's belonging to one of the classes of metamorphic malware using the means of one host), then the test sample was sent to other components of the AMJS multi-computer system to run them in isolated environments and manifest metamorphic properties.

Fig. 6 shows an example of two listings of opcodes for one instance of malware obtained after starting and disassembling on different hosts of the system. Modified blocks are marked with red squares.

After performing similar symmetrical actions on each of the system components (step 8 of the method of detecting metamorphic viruses), the results of the analysis were sent to the initiator component.

```
endbr64
push   %rbp
mov    %rsp,%rbp
push   %rbx
sub    $0x218,%rsp
mov    %fs:0x28,%rax
mov    %rax,-0x18(%rbp)
xor    %eax,%eax
lea    0xd99(%rip),%rax        # 0x2008
mov    %rax,%rsi
lea    0x2dc7(%rip),%rax       # 0x4040 <_ZSt4cout@GLIBCXX_3.4>
mov    %rax,%rdi
call   0x10f0 <_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
lea    -0x220(%rbp),%rax
mov    $0x10,%edx
lea    0xd7c(%rip),%rcx        # 0x2010
mov    %rcx,%rsi
mov    %rax,%rdi
call   0x1100 <_ZNSt14basic_ofstreamIcSt11char_traitsIcEEC1EPKcSt13_Ios_Openmode@plt>
lea    -0x220(%rbp),%rax
mov    %rax,%rdi
call   0x10d0 <_ZNSt14basic_ofstreamIcSt11char_traitsIcEE7is_openEv@plt>
test   %al,%al
je     0x12f5 <main+172>
lea    -0x220(%rbp),%rax
lea    0xd60(%rip),%rdx        # 0x2020
mov    %rdx,%rsi
mov    %rax,%rdi
call   0x10f0 <_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
lea    -0x220(%rbp),%rax
mov    %rax,%rdi
call   0x1120 <_ZNSt14basic_ofstreamIcSt11char_traitsIcEE5closeEv@plt>
lea    0xd5f(%rip),%rax        # 0x2040
mov    %rax,%rsi
lea    0x2d55(%rip),%rax       # 0x4040 <_ZSt4cout@GLIBCXX_3.4>
mov    %rax,%rdi
call   0x10f0 <_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
jmp    0x1315 <main+204>
lea    0xd7c(%rip),%rax        # 0x2078
mov    %rax,%rsi
lea    0x2e5a(%rip),%rax       # 0x4160 <_ZSt4cerr@GLIBCXX_3.4>
```

a)

```
endbr64
push   %rbp
mov    %rsp,%rbp
sub    $0x10,%rsp
mov    %fs:0x28,%rax
mov    %rax,-0x8(%rbp)
xor    %eax,%eax
lea    0xddc(%rip),%rax        # 0x2017
mov    %rax,%rsi
lea    0x2dfb(%rip),%rax       # 0x4040 <_ZSt4cout@GLIBCXX_3.4>
mov    %rax,%rdi
call   0x10c0 <_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
lea    -0x10(%rbp),%rax
mov    $0x0,%ecx
lea    -0x74(%rip),%rdx        # 0x11e9 <_Z14threadFunctionPv>
mov    $0x0,%esi
mov    %rax,%rdi
call   0x10f0 <pthread_create@plt>
mov    -0x10(%rbp),%rax
mov    $0x0,%esi
mov    %rax,%rdi
call   0x10b0 <pthread_join@plt>
mov    $0x0,%eax
mov    -0x8(%rbp),%rdx
sub    %fs:0x28,%rdx
je     0x1294 <main+123>
call   0x10d0 <__stack_chk_fail@plt>
leave
ret
```

b)

Fig.6. Two listings of opcodes of the investigated malware after start-up and disassembly on different hosts of the system: a) Host 1 and b) Host 3

A fragment of the log on the initiator component is shown in Fig. 7.

After additional verification with other components of the system, the final conclusion was formed (step 9 of the detection method).

Fig. 8 presents the interface window for receiving information by the component that initiated the check about the results of checking the suspicious program by other (Host1-Host4) hosts in the system. It should be noted that in this case, the result of the rest of the hosts in the system coincides, and the RPM is set as High. As a result, Host0 decides to block the suspicious program and add its behavior to the behavior blacklist.

```
e37d93bde495257fa0e7a488c6000e60fd115ec030de4aea006371f4682a71bf
bb4c91f2a819527c756074509af3c1af6bee0691dc81de9b770bc44dfe85e77e
e06cbab64638f1cdfe0ce5be7c99580653413754ceb78e109353f9943a8a293d
0843221ff80ef294a6611c45b90084092abc8d800baccb6c1b55e771ec84b93b
Host 1 started isolated container.
Host 1sent new functions vector to Host 0.
Host 1sent new assembly code to Host 0.
Host 4 started isolated container.
Host 2 started isolated container.
Host 4sent new functions vector to Host 0.
Host 3 started isolated container.
Host 2sent new functions vector to Host 0.
Host 3sent new functions vector to Host 0.
Host 1 functions : _GLOBAL__sub_I_main
__do_global_dtors_aux
__static_initialization_and_destruction_0(int, int)
_fini
_init
_start
deregister_tm_clones
frame_dummy
main
register_tm_clones

Host 2 functions : _GLOBAL__sub_I_main
```

Fig. 7. A fragment of the log illustrating the reception of the list of API functions by the Host 0 component from other components in a multicomputer system

4.2. Setting of the experiment and results of experimental studies on the detection of metamorphic viruses by the developed multicomputer system

Setting up the experiment. In the developed multicomputer system, files are available in each component, which contain information about the actions performed and functions launched to perform certain tasks, as well as about events that affect the system directly and cause it to react to them in a certain way. These files must be constantly updated, and outdated information is removed from them according to certain criteria. In addition, according to these files, the system creates separate tables in which information about the system's reactions to the given effects is stored for a long time. Such information enables the center of the system to develop options for decisions regarding further steps of the system, rate them, and agree with the controller. The controller uses the information from the corresponding table of effects to finalize the decision regarding further steps of the system according to one effect. Thus, the internal tables of the system components contain information about the previous decisions of the system as decisions on the defined impacts. In other words, the developed multicomputer system is positioned as a system with memory elements. Since at the initial stage of its operation it does not have

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

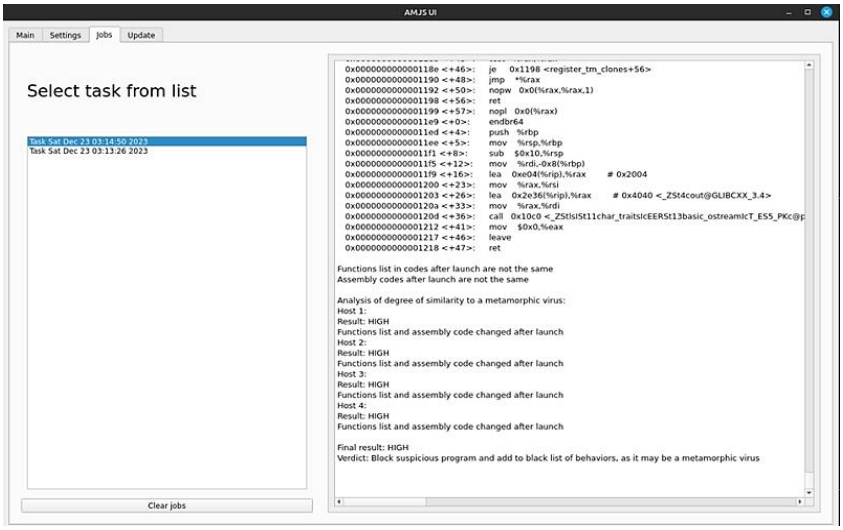


Fig. 8. Interface window for receiving information by the component

information about previous events, we consider it in the first experiment as one that does not contain a fully functional controller of the decisions made. In this case, the controller approves the decision of the system center [4] according to the rating (highest) of the proposed options. Accordingly, for each series of the first stage of the experiment, the conditions were the same.

The second important factor in conducting an experiment is metamorphic viruses and their variety [1]-[3]. Let's highlight the following features: writing completely into an executable PE-file while preserving the volume of the original file, i.e. replacing part of the bits and, accordingly, losing the functionality of the original file and saving part of the bits of the original file without using them; recording completely in the executable PE-file without saving the volume of the initial file, i.e. adding part of the bits of the metamorphic virus to the initial file and, accordingly, saving the functionality of the initial file and obtaining access to execute the first command of the virus when the infected PE-file is launched; replacement of the initial PE file with a metamorphic virus with a change or preservation of the size of the file in bits by supplementing with inactive commands; writing the metamorphic virus to an initial PE file and copying and storing this initial file under a different name with the possibility of calling it for execution after executing the virus commands. These features form the basis for the formation of four subsets of metamorphic viruses. The selection of these subsets is decisive in the context of the study of their code according to the results of the previous experiment on obtaining disassembled commands and the logic of the studied programs. For example, if a virus is embodied in a file such that the entry point to the program is the first command of the virus and there is an unconditional transition through several commands, then such a mechanism of the virus forms one template. In the absence of an unconditional transition, a second template is formed. Therefore, these features should be considered when detecting the metamorphic code of viruses.

Let us denote the subsets of metamorphic viruses as follows: $\mathfrak{H}_1$ – replacement of bits, the volume of the initial file in bits is saved, the loss of the functionality of the initial PE file, the presence of an unconditional transition at the beginning of the PE file (from the point of entry into the file), the absence of the function of saving the initial PE file in a separate file; $\mathfrak{H}_2$ – replacement of bits, the volume of the initial file in bits is preserved, the loss of the functionality of the initial PE-file, the presence of an unconditional transition at the beginning of the PE-file (from the point of entry into the file), saving the initial PE-file in a separate file with the possibility of calling it for execution after execution metamorphic virus commands; $\mathfrak{H}_3$ – replacement of bits, the volume of the initial file in bits is preserved, the loss of the functionality of the initial PE-file, the presence of an unconditional transition at the beginning of the PE-file (from the point of entry into the file), saving the initial PE-file in a separate file without calling it for execution after executing commands metamorphic virus; $\mathfrak{H}_4$ – replacement of bits, the volume of the initial file in bits is not saved, the loss of functionality of the initial PE file, the presence of an unconditional transition at the beginning of the PE file (from the point of entry into the file), the absence of the function of saving the initial PE file in a separate file; $\mathfrak{H}_5$ – replacement of bits, the volume of the initial file in bits is preserved, the loss of the functionality of the initial PE file, the presence of an unconditional transition at the beginning of the PE file (from the point of entry into the file), saving the initial PE-file in a separate file with the possibility of calling it for execution after executing the commands of the metamorphic virus; $\mathfrak{H}_6$ – replacement of bits, the volume of the initial file in bits is not saved, the loss of the functionality of the initial PE file, the presence of an unconditional transition at the beginning of the

PE file (from the point of entry into the file), saving the initial PE file in a separate file without calling it for execution after execution metamorphic virus commands; $\mathfrak{H}_7$ – replacement of bits, the volume of the initial file in bits is saved, loss of functionality of the initial PE file, absence of an unconditional transition at the beginning of the PE file (from the point of entry into the file), absence of the function of saving the initial PE file in a separate file; $\mathfrak{H}_8$ – replacement of bits, the volume of the initial file in bits is preserved, loss of the functionality of the initial PE file, no unconditional transition at the beginning of the PE file (from the point of entry into the file), saving the initial PE file in a separate file with the possibility of calling it for execution after execution metamorphic virus commands; $\mathfrak{H}_9$ – replacement of bits, the volume of the initial file in bits is preserved, loss of the functionality of the initial PE file, no unconditional transition at the beginning of the PE file (from the point of entry into the file), saving the initial PE file in a separate file without calling it for execution after executing commands metamorphic virus; $\mathfrak{H}_{10}$ – substitution of bits, volume of the initial file in bits is not saved, loss of functionality of the initial PE file, lack of unconditional transition at the beginning of the PE file (from the point of entry into the file), lack of function to save the initial PE file in a separate file; $\mathfrak{H}_{11}$ – replacement of bits, the volume of the initial file in bits is preserved, loss of the functionality of the initial PE file, no unconditional transition at the beginning of the PE file (from the point of entry into the file), saving the initial PE file in a separate file with the possibility of calling it for execution after execution metamorphic virus commands; $\mathfrak{H}_{12}$ – replacement of bits, the size of the initial file in bits is not saved, the loss of the functionality of the initial PE file, the absence of an unconditional transition at the beginning of the PE file (from the point of entry into the file), saving the initial PE file in a separate file without calling it for execution after execution metamorphic virus commands.

Thus, in the set of metamorphic viruses, 12 subsets separated by certain criteria were selected. These subsets include most types of metamorphic viruses, but their union is not an exhaustive set of metamorphic viruses. To conduct the experiment, we divided the data in 12 subsets, and each subset was defined by an element or elements in the experiment.

Let's define a set of metamorphic viruses as follows:

$$\mathfrak{H} = \bigcup_{\mathfrak{k}=1}^{12} \mathfrak{H}_{\mathfrak{k}}. \quad (5)$$

Let us denote the number of elements in the set $\mathfrak{H}$ as $\mathfrak{N}_{\mathfrak{H}}$, and similarly for subsets $\mathfrak{H}_{\mathfrak{k}}$ we denote $\mathfrak{N}_{\mathfrak{H}_{\mathfrak{k}}}$ ($\mathfrak{k} = 1,2,\dots,12$). These designations were used when processing the results of the experiment and indicated the number of metamorphic viruses of a certain subset and the set as a whole, which were used for experimental research.

The purpose of the first stage of the experiment is to establish the reliability of the detection of metamorphic viruses developed by a multi-computer malware detection system with metamorphic functionality, provided there is no information in the memory elements for the previous steps and, accordingly, without the involvement of the controller. That is, in the first experiment with the system, only the center of the system was involved in determining the detection of a metamorphic virus [4]. In contrast to host systems, in the developed multi-computer malware detection system with metamorphic functionality, efficiency improvement was achieved by increasing computing resources, using computer network capacity and features in the architecture of the multi-computer system itself. Metamorphic viruses can be detected by host systems without involving the rest of the computing resources of the network, and can also be detected by components of a multicomputer system without involving part of the entire system and the center of the system. In this case, the system components and their sensors performed decision-making functions.

Strategy at work [52] involved researching the activities of malicious software in hosts, comparing them with each other, and making a general decision according to the received decisions. It includes a mechanism according to which the system components assign a suspicious status to the executed process, which is then considered by the rest of the components. The decision about whether a suspicious process is running on a host or on an individual host is made at a single system center. If an executable is running on only one host and not on the others, it is difficult to determine whether malicious activity is present. However, such options were not investigated in the setting of this experiment. Thus, in the first stage of the experiment, the multicomputer system itself was tested for the detection of metamorphic viruses, positioning it as a single sensor without a controller.

The purpose of the next stages of the experiment is to establish the reliability of the detection of metamorphic viruses by the developed multi-computer malware detection system with metamorphic functionality, provided that there is information in the memory elements for the previous steps and, accordingly, with the involvement of the controller. The second stage of the experiment included the capabilities of the developed multicomputer system using previous experience in detecting metamorphic viruses; however, this experience, which is formed on the information in the memory elements, was insignificant. Therefore, it is advisable to conduct a certain number of stages of experiments to achieve constant reliability in the detection of metamorphic viruses. Enough stages was defined as the number of stages at

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

which the detection reliability results were no longer improved. When conducting experimental research, the number of stages needed to be determined. It was influenced by the particularity of the architecture of the controller and the mechanism for approving the decision of the system center embedded in it.

During the experiment, the controller was configured so that every third and sixth stage, when the effects were repeated, the second-ranked answer was chosen, and every ninth time the third-ranked answer was chosen. Then, the minimum number of stages of experiments from the initial one should be equal to ten because it was included in the first stage. In order to continue the research after the tenth stage, nine experiments should be conducted to ensure compliance with stages two through ten, that is, to perform a full iteration. However, the number of stages may not necessarily be 10, 19, or 28. It can be different and determined by the fact that at two or three adjacent stages, depending on the formed requirements, the appropriate level of reliability of detection of metamorphic viruses was achieved.

An important part of the experiment is a set of test samples of metamorphic viruses and their correct use because they all have functionality that involves implementation into executable PE files. To conduct the experiment, we chose five different, but at the same time, those that have characteristic features of the class, metamorphic viruses for each of the subsets $\mathfrak{H}_{\mathfrak{k}}$ ($\mathfrak{k} = 1,2,\dots,12$). Their construction was performed artificially without including malicious functionality. Thus, the number of samples of different metamorphic viruses used in the experiment was sixty. The general pattern of all sixty metamorphic viruses was searched for an executable PE file and infect it with a type that defines one of the classes $\mathfrak{H}_{\mathfrak{k}}$ ($\mathfrak{k} = 1,2,\dots,12$) and it must be of the same type as the executable metamorphic virus.

Benign software, i.e., user programs, must also be executed during the experiment. We selected twenty-five computer stations in the corporate network that were located in at least three of its segments. We turned off the rest of the computer stations.

We installed the system components in each of them and performed its initial settings and start-up. When conducting a series of experiments at different stages of the installation of system components, we repeated the experiments to ensure their independence for all twelve types of metamorphic viruses, which are specified by the corresponding classes $\mathfrak{H}_{\mathfrak{k}}$ ($\mathfrak{k} = 1,2,\dots,12$).

For each type of metamorphic virus, the multi-computer system was not reinstalled. It accumulates knowledge in memory elements for use by the controller. Thus, the independence of the experiment was applied exclusively to classes $\mathfrak{H}_{\mathfrak{k}}$ ($\mathfrak{k} = 1,2,\dots,12$).

In each of the twenty-five nodes in which the components of the multicomputer system were installed, only system processes and one researched process were launched. In twenty-four nodes, twenty-four different processes were launched for execution, which were oriented to the use of input–output to ensure the duration of execution.

After a given custom process is fully executed, it is repeated until the experiment series is completed at the set time. A metamorphic virus was launched in a node. The time for conducting a series of experiments was determined by considering the time spent on the execution of the metamorphic virus and the processing of events related to it in a multicomputer system.

The time for all series of the experiment was the same and was determined experimentally during the previous series of the experiments. For a defined set of initial data of the planned experiment, we set the time for conducting a series of experiments lasting 30 min.

For the metamorphic virus, the same executable PE files were used as baits and as the targets of its attacks for further infection. These node-specific bait files had the same bit set but different names. They were placed in all directories and the root directory.

All research was conducted in an isolated environment of the corporate network. After all experiments and experiments with classes were completed, the system software was reinstalled with the same settings in the computer stations.

Conduct the experiment and the results of the experimental research. To conduct experimental studies with the developed multi-computer system, the specified executable programs were launched at the same time in each specified node of the corporate network. For this, the same launcher was used with the launch time set.

The results of all series of experiments were separately saved with log files and tables of memory elements.

To evaluate the efficiency of the detection approach *TPR* – True Positive Rate, *FPR* – False Positive Rate, *Precision*, *Recall*, *F1-score*, and *MCC* metrics were involved [56, 57]:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP+FN}} \cdot 100, \quad (6)$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN+FP}} \cdot 100, \quad (7)$$

$$\mathrm{S_P} = \frac{\mathrm{TN}}{\mathrm{TP+FN}} \cdot 100, \quad (8)$$

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}, \quad (9)$$

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}, \quad (10)$$

$$\mathrm{F1} = \frac{2*\mathrm{Recall}*\mathrm{Precision}}{\mathrm{Recall+Precision}}, \quad (11)$$

$$\mathrm{MCC} = \frac{\mathrm{TP}*\mathrm{TN}-\mathrm{FP}*\mathrm{FN}}{\sqrt{(\mathrm{TP+FP})(\mathrm{TP+FN})(\mathrm{TN+FP})(\mathrm{TN+FN})}}, \quad (12)$$

Matthews correlation coefficient (MCC) [56] is defined from -1 to 1, where ±1 means full agreement or disagreement, and a value of zero means no relationship. The phi coefficient has a maximum value determined by the distribution of two variables if one or both variables can take more than two values.

The examples for 1st and 21st results of a series of experimental studies (total 46 stages) are presented in Table 1. The experiment is stopped after the full completion of step 46 because the values obtained in steps 44-46 are close and reflect an improvement in the reliability of detection compared with the previous steps. That is, the requirements for stopping the experiment have been met. To process the results of the experimental studies, we used the ROC analysis technique [55].

Examples of results of ROC analysis for 1st and 21st stages (total 46 stages) are given separately in Table 2.

Table 1

The results of the study on the accuracy of detection of metamorphic viruses at certain stages, considering all the features of artificially created class instances

Stage	Detection result	Metamorphic virus class												Total
		1	2	3	4	5	6	7	8	9	10	11	12	
1	TP	11	10	13	13	12	12	13	12	12	12	11	12	143
	FN	4	5	2	2	3	3	2	3	3	3	4	3	37
	FP	34	27	20	20	19	33	35	32	29	21	28	27	325
	TN	326	333	340	340	341	327	325	328	331	339	332	333	3995
...														
21	TP	10	13	13	11	11	12	10	11	9	13	13	12	138
	FN	5	2	2	4	4	3	5	4	6	2	2	3	42
	FP	33	21	34	22	32	19	36	42	40	26	28	25	358
	TN	327	339	326	338	328	341	324	318	320	334	332	335	3962
	FN	4	4	3	6	2	4	2	3	2	4	6	4	44
...														

Table 2

Examples of results of ROC analysis for 1st and 21st stages (total 46 stages)

Stage	TPR	FPR	S_p	Precision	Recall	F1	MCC
1	79.44	7.52	92.48	0.31	0.79	0.44	0.46
...							
21	76.67	8.29	91.71	0.28	0.77	0.41	0.43
...							

Table 2 shows the evaluation of the MCC value. The obtained results in table 2 confirm the appropriate level of the developed classifier of metamorphic viruses.

The demonstration of the change in the values of the main indicators of the ROC analysis when performing the 46 stages is shown in Fig. 9.

In Fig. 9, a, the graph of the function shows the growth of the reliability value of detection of metamorphic viruses in the process of filling the system with information about its previous steps.

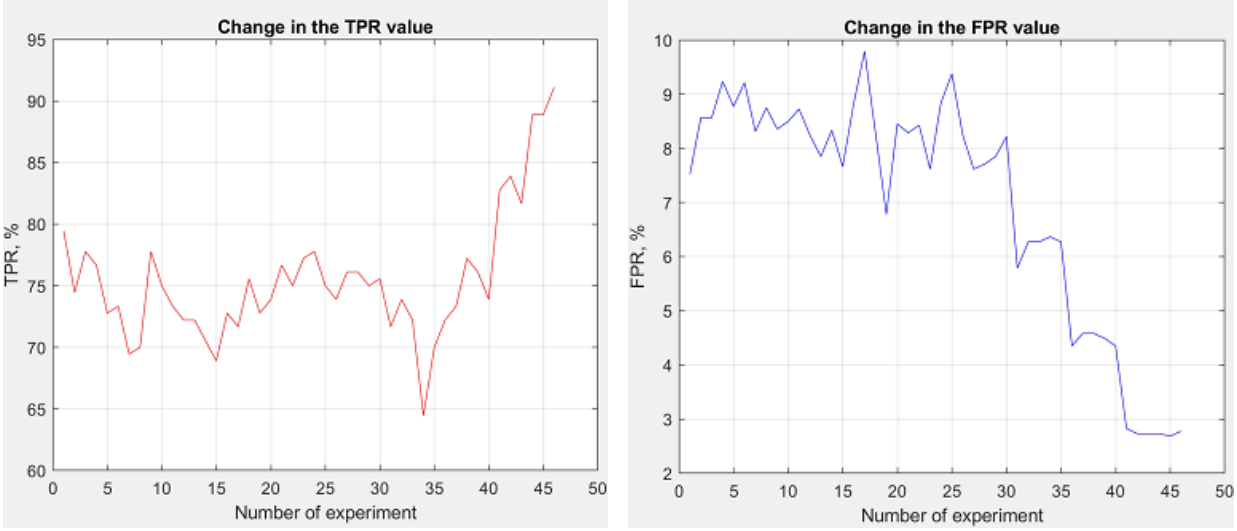
The detection result of the developed multi-computer system reached 90%, which is a sufficient basic indicator.

Fig. 9, b shows a graph for the false positive rate. The graph is descending, demonstrating the improvement in the system's classification of programs that do not contain metamorphic functionality.

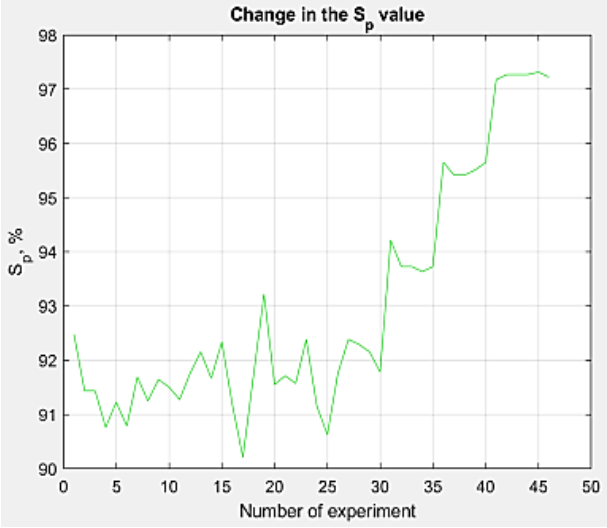
The percentage of false positives was 3%, and it improved by about 6%, which confirms the need to use information about the results of the previous stages of detection of metamorphic viruses.

Similarly, Fig. 9, b and Fig. 9, c show the result of classification of programs without metamorphic functionality and the result is about 97%, which is acceptable for further improvements of the developed multicomputer system.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)



a) b)



c)

Fig. 9. Graphs for functions from Table 2

4.3. Discussion

Let's summarize all the results obtained at all stages into a single table of values from stages 1-46. As a result, we have received the sum of the values for each class of metamorphic viruses according to the indicators of the ROC analysis.

At the first stages, these values were smaller, and at the last stages, they were larger, but this made it possible to assess the quality of the test samples of classes of metamorphic viruses.

This was achieved by comparing the deviation of the indicator results from the average value.

If the deviation is significant, then it is necessary to analyze instances of metamorphic viruses of the class in which the deviation is found, or to finalize the corresponding subsystem of the multicomputer system.

Table 3 shows the summary values of the results of the experiment.

The arithmetic mean value of detection reliability for all classes and all series and stages of the experiment is equal $TP = 75.46$ %. The deviation from this value of all the values of the twelve classes does not exceed 3%; therefore, the created instances of classes of metamorphic viruses are correlated with each other. This is also confirmed by the values of the remaining indicators of the ROC analysis.

Thus, according to the results of the conducted experiment, it was established that the presence of the controller in a multi-computer system provides improved detection of metamorphic viruses with increasing number of applications.

Table 3

Results of the study of individual types of artificially created instances of classes at all stages

Detection result	Classes of metamorphic viruses												Total
	1	2	3	4	5	6	7	8	9	10	11	12	
TP	510	520	519	511	517	515	535	519	520	540	501	541	6248
FN	180	170	171	179	173	175	155	171	170	150	189	149	2032
FP	1132	1151	1167	1115	1096	1214	1132	1228	1161	1164	1178	1117	13855
TN	15428	15409	15393	15445	15464	15346	15428	15332	15399	15396	15382	15443	184865
TPR	73.91	75.36	75.22	74.06	74.93	74.64	77.54	75.22	75.36	78.26	72.61	78.41	75.46
FPR	6.84	6.95	7.05	6.73	6.62	7.33	6.84	7.42	7.01	7.03	7.11	6.75	6.97
Precision	0.31	0.31	0.31	0.31	0.32	0.30	0.32	0.30	0.31	0.32	0.30	0.33	0.31
Recall	0.74	0.75	0.75	0.74	0.75	0.75	0.78	0.75	0.75	0.78	0.73	0.78	0.75
F1	0.44	0.44	0.44	0.44	0.45	0.43	0.45	0.43	0.44	0.45	0.42	0.46	0.44
MCC	0.45	0.45	0.45	0.45	0.46	0.44	0.47	0.44	0.45	0.47	0.43	0.48	0.45

Conclusion and Future Work

Thus, a conceptual model of multi-computer systems has been developed, in which, unlike known models of multi-computer systems, which are designed to ensure the functioning with their support of antivirus baits and traps for detecting malware and cyberattacks in corporate networks, as well as for prevention and countermeasures their penetration, in the conceptual model of multi-computer systems a defining characteristic is introduced, which is responsible for the control of the decisions made, and the rest of the defining characteristics, which in the process of functioning of the systems should form the architecture of the system by independently synthesizing a set of separate defining characteristics according to a closed route in the graph of defining characteristics in architecture of the systems, as well as specialized functionality is allocated, which, compatible with the general part of the system, forms the system as the single sensor, which makes it possible to provide a variety of options for responses to the influence of intruders, cyberattacks and the functioning of malware, and also makes it possible not only to diversify the system due to its different architecture, but also ensures its stability when removing certain nodes in the corporate network and when combining specialized functionality with the general part of the system forms the system as the single sensor, which generally improves the effectiveness of countering malware and cyberattacks.

Methods for detecting metamorphic viruses have been developed with the possibility of their implementation in the architecture of multicomputer systems with baits and traps in such a way that the system directly participates in the detection procedure through its components and makes a decision about the presence of metamorphic code in the executable PE file.

The implementation of a multi-computer system for detecting malicious software with metamorphic functionality has been developed to demonstrate the ability to implement the proposed conceptual architecture model and the developed methods of detecting metamorphic viruses. An experiment was set up regarding the functioning of a multi-computer system for detecting malicious software, and experimental studies were conducted with it in the part of studying the metamorphic code development process to confirm the possibility of implementing the steps of the developed methods of detecting metamorphic viruses. In addition, an experiment was conducted regarding the effectiveness of detecting the metamorphic code of viruses, and relevant experimental studies were conducted.

Based on the results of the work performed, the effectiveness of the detection of the metamorphic code of viruses by the developed multi-computer system was investigated and the presence of an improvement in detection was established.

The directions of **further work** according to the conceptual model to ensure the functioning of multi-computer systems from combined antivirus baits and traps and the decision-making controller for detecting malware and cyberattacks in corporate networks, it is necessary to develop a method of organizing the functioning of such systems, as well as a method of organizing the functioning of combined antivirus baits and traps and methods of detecting malware and cyberattacks using combined antivirus baits and traps implemented in the architecture of such systems to improve detection efficiency, as well as spread the results of work to new types of malicious software.

Authors Contribution

Antonina Kashtalian analyzed the known methods of developing bait and traps, developed a conceptual model of multi-computer malware detection systems, participated in the development of the implementation of the detection system with metamorphic functionality, and designed and conducted experiments.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Sergii Lysenko participated in the formulation of the problem and purpose of the study, determined the strategy for developing methods for detecting metamorphic viruses, participated in the experimental design and verification of scientific results.

Oleg Savenko formulated the research problem, defined the conceptual provisions for the development, participated in the experiment design, processed the experiment results in terms of malware detection via the system, and verified the scientific and practical results.

Andrii Nicheporuk developed two methods for detecting metamorphic viruses and participated in the design and processing of the experimental results.

Tomas Sochor participated in the malware detection approach creation and verification of scientific results.

Volodymyr Avsiyevych participated in the development of the implementation of a multi-computer system for detecting malware with metamorphic functionality and conducted experiments to obtain the results of the study of metamorphic code.

Conflict of interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

This study was conducted without financial support.

Data availability

The manuscript has no associated data.

Use of Artificial Intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current study.

All the authors have read and agreed to the published version of this manuscript.

References

1. Markowsky, G. Savenko, O., Lysenko, S., & Nicheporuk, A. The technique for metamorphic viruses' detection based on its obfuscation features analysis. *CEUR-WS*, 2018, vol. 2104, pp. 680–687.

2. Savenko, O., Lysenko, S., Nicheporuk, A., & Savenko, B. Metamorphic Viruses' Detection Technique Based on the Equivalent Functional Block Search. *CEUR-WS*, 2017, vol. 1844, pp. 555–569.

3. Savenko, O., Lysenko, S., Nicheporuk, A., & Savenko, B. Approach for the Unknown Metamorphic Virus Detection. *Proceedings of the 8-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS 2017)*, Bucharest (Romania), September 21–23, 2017, Bucharest, 2017, pp. 71–76. DOI:10.1109/IDAACS.2017.8095052.

4. Kashtalian, A., Lysenko, S., Savenko, B., Sochor, T., & Kysil, T. Principle and method of deception systems synthesizing for malware and computer attacks detection. *Radioelectronic and Computer Systems*, 2023, no. 4, pp. 112-151. DOI: 10.32620/reks.2023.4.10.

5. Han, X., Kheir, N., & Balzarotti, D. Deception Techniques in Computer Security. *ACM Computing Surveys (CSUR)*, 2018, vol. 51, pp. 1-36. DOI: 10.1145/3214305.

6. Pawlick, J., Colbert, E., & Zhu, Q. A Game-theoretic Taxonomy and Survey of Defensive Deception for Cybersecurity and Privacy. *ACM Computing Surveys (CSUR)*, 2018, vol. 52, pp. 1-28. DOI: 10.48550/arXiv.1712.05441.

7. Almeshekah, M. H., & Spafford, E. H. Cyber Security Deception. *In: Jajodia, S., Subrahmanian, V., Swarup, V., Wang, C. (eds) Cyber Deception*, Springer, Cham, 2016. DOI: 10.1007/978-3-319-32699-3_2.

8. Chessa, M., Grossklags, J., & Loiseau, P. A Game-Theoretic Study on Non-monetary Incentives in Data Analytics Projects with Privacy Implications, *2015 IEEE 28th Computer Security Foundations Symposium*, Verona, Italy, 2015, pp. 90-104. DOI: 10.1109/CSF.2015.14.

9. Shokri, R. Privacy games: Optimal user-centric data obfuscation. *Proc. Privacy Enhancing Technologies*, 2015, vol. 2, pp. 299–315. DOI 10.1515/popets-2015-0024.

10. Pawlick, J., & Zhu, Q. A Stackelberg Game Perspective on the Conflict Between Machine Learning and Data Obfuscation. *In IEEE Workshop on Inform. Forensics and Security*, 2016. Available at: https://arxiv.org/abs/1608.02546. (accessed 12.12.2023).

11. Clark, A., Zhu, Q., Poovendran, R., & Basar, T. Deceptive routing in relay networks. *In Decision and Game Theory for Security*. Springer, 2012, pp. 171–185. DOI: 10.1007/978-3-642-34266-0_10.

12. Lu, R., Lin, X., Luan, T. H., Liang, X., & Shen, X. Pseudonym changing at social spots: An effective strategy for location privacy in vanets. *IEEE Trans Vehicular Technol*, 2012, vol. 61, iss. 1, pp. 86–96. DOI: 10.1109/TVT.2011.2162864.

13. Durkota, K., Lisy, V., Bosansky, B., & Kiekintveld, C. Optimal Network Security Hardening Using Attack Graph Games. *In Intl. Joint Conf. on Artificial Intelligence*, 2015, pp. 526–532. Available at: https://www.semanticscholar.org/paper/Optimal-Network-Security-Hardening-Using-Attack-Durkota-Lis%C3%BD/114c35ed4e6be9e556f36bed7af3bfe9fe9209d9. (accessed 10.12.2023).

14. Horak, K., Zhu, Q., & Bosansky, B. Manipulating Adversary's Belief: A Dynamic Game Approach to Deception by Design in Network Security. *In Decision and Game Theory for Security*. Springer, 2017, pp. 273–294. DOI: 10.1007/978-3-319-68711-7_15.

15. Al-Shaer, E. A Cyber Mutation: Metrics, Techniques and Future Directions. *In Proceedings of the 2016 ACM Workshop on Moving Target Defense (MTD '16)*. Association for Computing Machinery, New York, NY, USA, 2016, vol. 1. DOI: 10.1145/2995272.2995285.

16. Park, K., Woo, S., Moon, D., & Choi, H. Secure Cyber Deception Architecture and Decoy Injection to Mitigate the Insider Threat. *Symmetry*, 2018, vol. 10, iss. 1, article no. 14. DOI: 10.3390/sym10010014.

17. Kechao, L., & Xinli, X. OpenHIP Random Host Hopping in Network Layer. *In International Conference on Education, Management and Information Technology (ICEMIT 2019)*, 2019. DOI: 10.25236/icemit.2019.048.

18. Adili, M. T., Mohammadi, A., Manshaei, M. H. & Rahman, M. A. A cost-effective security management for clouds: A game-theoretic deception mechanism. *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Lisbon, Portugal, 2017, pp. 98-106. DOI: 10.23919/INM.2017.7987269.

19. Reti, D., Fraunholz, D., Elzer, K., Schneider, K., & Schotten, H. D. Evaluating Deception and Moving Target Defense with Network Attack Simulation. *In Proceedings of the 9th ACM Workshop on Moving Target Defense (MTD'22)*. Association for Computing Machinery, New York, NY, USA, 2022, pp. 45–53. DOI: 10.1145/3560828.3564006.

20. Franco, J., Aris, A., Canberk, B., & Uluagac, A. S. A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. *arXiv:2108.02287v1* [cs.CR] 4 Aug 2021. Available at: https://arxiv.org/pdf/2108.02287.pdf. (accessed 12.12.2023).

21. Zielinski, D., & Kholidy, H. A. An Analysis of Honeypots and their Impact as a Cyber Deception Tactic arXiv:2301.00045v1 [cs.CR] 30 Dec 2022. Available at: https://doi.org/10.48550/arXiv.2301.00045. (accessed 12.12.2023).

22. Sochor, T., & Zuzcak, M. High-Interaction Linux Honeypot Architecture in Recent Perspective. *In: Gaj, P., Kwiecień, A., Stera, P. (eds) Computer Networks. CN 2016. Communications in Computer and Information Science*, 2016, vol. 608. Springer, Cham. DOI: 10.1007/978-3-319-39207-3_11.

23. Chovancová, E., & Ádám, N. A Clustered Hybrid Honeypot Architecture. *Acta Polytechnica Hungarica*, 2019, vol. 16, iss. 10, pp. 173-189. DOI: 10.12700/APH.16.10.2019.10.11.

24. Baykara, M., & Das, R. A novel honeypot based security approach for real-time intrusion detection and prevention systems. *Journal of Information Security and Applications*, 2018, vol. 41, pp. 103-116. DOI: 10.1016/j.jisa.2018.06.004.

25. Li, Y., Shi, L., & Feng, H. A Game-Theoretic Analysis for Distributed Honeypots. *Future Internet*, 2019, vol. 11, iss. 3, article no. 65. DOI: 10.3390/fi11030065.

26. Fraunholz, D., Zimmermann, M., & Schotten, H. D. An adaptive honeypot configuration, deployment and maintenance strategy. 2017 19th *International Conference on Advanced Communication Technology (ICACT)*, 2017, pp. 53-57. DOI: 10.23919/ICACT.2017.7890056.

27. Wang, K., Du, M., Maharjan, S., & Sun, Y. Strategic Honeypot Game Model for Distributed Denial of Service Attacks in the Smart Grid. *In IEEE Transactions on Smart Grid*. Sept. 2017, vol. 8, no. 5, pp. 2474-2482, DOI: 10.1109/TSG.2017.2670144.

28. Nasr, M., Zolfaghari, H., & Houmansadr, A. The Waterfall of Liberty: Decoy Routing Circumvention that Resists Routing Attacks. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017. DOI: 10.1145/3133956.3134075.

29. Sadasivam, G. K., & Hota C. Scalable Honeypot Architecture for Identifying Malicious Network Activities. *2015 International Conference on Emerging Information Technology and Engineering Solutions*. Mahashtra, India, 2015, pp. 27-31. DOI: 10.1109/EITES.2015.15.

30. Kumar, S., Janet, B., & Eswari, R. Multi Platform Honeypot for Generation of Cyber Threat Intelligence. *2019 IEEE 9th International Conference on Advanced Computing (IACC)*. Tiruchirappalli, India, 2019, pp. 25-29. DOI: 10.1109/IACC48062.2019.8971584.

31. You, J., Lv, S., Sun, Y., Wen, H., & Sun, L. HoneyVP: A Cost-Effective Hybrid Honeypot Architecture for Industrial Control Systems. *ICC 2021 - IEEE International Conference on Communications*, Montreal, QC, Canada, 2021, pp. 1-6, DOI: 10.1109/ICC42927.2021.9500567.

32. Ilg, N., Duplys, P., Sisejkovic, D., & Menth, M. A survey of contemporary open-source honeypots, frameworks, and tools. *Journal of Network and Computer Applications*, 2023, vol. 220, article no. 103737, ISSN 1084-8045, DOI: 10.1016/j.jnca.2023.103737.

33. Shabtai, A., Bercovitch, M., Rokach, L., Gal, Y., Elovici, Y., & Shmueli, E. Behavioral Study of Users When Interacting with Active Honeytokens. *ACM Trans. Inf. Syst. Secur.*, 2016, vol. 18, iss. 3, article no. 9, pp. 1-21. DOI: 10.1145/2854152.

34. Juels, A., & Rivest, R. L. Honeywords: Making password-cracking detectable. In Proceedings of the 2013 *ACM SIGSAC conference on Computer & communications security*, 2013, pp. 145–160. DOI: 10.1145/2508859.2516671.

35. Rrushi, J. L. NIC displays to thwart malware attacks mounted from within the OS. *Comput. Secur.*, 2016, vol. 61, pp. 59–71. DOI: 10.1016/j.cose.2016.05.002.

36. Kaghazgaran, P., & Takabi, H. Toward an Insider Threat Detection Framework Using Honey Permissions. *Journal of Internet Services and Information Security (JISIS)*, 2015, vol. 5, iss. 3. DOI: 10.22667/JISIS.2015.08.31.019.

37. Efendi, M. A., Ibrahim, Z. B., Zawawi, M. N., Rahim, F. A., Pahri, N. A., & Ismail, A. A Survey on Deception Techniques for Securing Web Application. *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. 2019,

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

pp. 328-331. DOI: 10.1109/BigDataSecurity-HPSC-IDS.2019.00066.

38. Onaolapo, J., Mariconti, E., & Stringhini, G. What Happens After You Are Pwnd: Understanding the Use of Leaked Webmail Credentials in the Wild. *In Proceedings of the 2016 Internet Measurement Conference (IMC '16)*. Association for Computing Machinery, New York, NY, USA, 2016, pp. 65–79. DOI: 10.1145/2987443.2987475.

39. De Faveri, C., Moreira, A., & Amaral, V. Multi-Paradigm Deception Modeling for Cyber Defense. *The Journal of Systems & Software*, 2018, vol. 141, pp. 32-51. DOI: 10.1016/j.jss.2018.03.031.

40. De Cristofaro, E., Friedman, A., Jourjon, G., Ali Kaafa, M. A., & Shafiq, M. Z. Paying for Likes? Understanding Facebook Like Fraud Using Honeypots. *In Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*. Association for Computing Machinery, New York, NY, USA, 2014, pp. 129-136. DOI: 10.1145/2663716.2663729.

41. Almeshekah, M. H., & Spafford, E. H. Planning and Integrating Deception into Computer Security Defenses. *In Proceedings of the 2014 New Security Paradigms Workshop (NSPW '14)*. Association for Computing Machinery, New York, NY, USA, 2014, pp. 127–138. DOI: 10.1145/2683467.2683482.

42. Bercovitch, M., Renford, M., Hasson, L., Shabtai, A., Rokach, L., & Elovici, Y. HoneyGen: An automated honeytokens generator. *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics*, Beijing, China, 2011, pp. 131-136. DOI: 10.1109/ISI.2011.5984063.

43. Matin, I. M. M., & Rahardjo, B. Malware Detection Using Honeypot and Machine Learning. *2019 7th International Conference on Cyber and IT Service Management (CITSM)*. Jakarta, Indonesia, 2019, pp. 1-4. DOI: 10.1109/CITSM47753.2019.8965419.

44. Ahmed, J., Karpenko, A., Tarasyuk, O., Gorbenko, A., & Sheikh-Akbari, A. Consistency issue and related trade-offs in distributed replicated systems and databases: a review. *Radioelectronic and Computer Systems*, 2023, no. 2. pp. 171-179. DOI: 10.32620/reks.2023.2.14.

45. Fursov, I., Yamkovyi, K., & Shmatko, O. Smart Grid and wind generators: an overview of cyber threats and vulnerabilities of power supply networks. *Radioelectronic and Computer Systems*, 2022, vol. 4. pp. 50-63. DOI: 10.32620/reks.2022.4.04.

46. Dovbysh, A., Liubchak, V., Shelehov, I., Simonovskiy, J., & Tenytska, A. Information-extreme machine learning of a cyber attack detection system. *Radioelectronic and Computer Systems*, 2022, no. 3, pp. 121-131. DOI: 10.32620/reks.2022.3.09.

47. Morozova, O., Nicheporuk, A, Tetskyi, A., & Tkachov, V. Methods and technologies for ensuring cybersecurity of industrial and web-oriented systems and networks. *Radioelectronic and Computer Systems*, 2021, no. 4, pp. 145-156. DOI: 10.32620/reks.2021.4.12.

48. Moskalenko, V., Zarets'kyy, M., Moskalenko, A., Kudryavtsev, A., & Semashko, V. Multi-layer model and training method for malware traffic detection based on decision tree ensemble. *Radioelectronic and Computer Systems*, 2020, no. 2, pp. 92-101. DOI: 10.32620/reks.2020.2.08.

49. Lysenko, S., Bobrovnikova, K., Shchuka, R., & Savenko, O. A Cyberattacks Detection Technique Based on Evolutionary Algorithms. *11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 2020, vol. 1, pp. 127-132. DOI: 10.1109/DESSERT50317.2020.9125016.

50. Bobrovnikova, K., Lysenko, S., Savenko, B., Gaj, P., & Savenko, O. Technique for IoT malware detection based on control flow graph analysis. *Radioelectronic and Computer Systems*, 2022, no. 1, pp. 141–153. DOI: 10.32620/reks.2022.1.11.

51. Savenko, B., Kashtalian, A., Lysenko, S., & Savenko, O. Malware Detection By Distributed Systems with Partial Centralization. *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Dortmund, Germany, 2023, pp. 265-270. DOI: 10.1109/IDAACS58523.2023.10348773.

52. Savenko, O., Lysenko, S., & Kryschuk, A. Multi-agent based approach of botnet detection in computer systems. *CCIS*, 2012, vol. 291, pp. 171–180. DOI: 10.1007/978-3-642-31217-5_19.

53. Kleshch, K., & Shablii, V. Comparison of fuzzy search algorithms based on Damerau-Levenshtein automata on large data. *Technology audit and production reserves*, 2023, vol. 4, no. 2/72, pp. 27-32. DOI: 10.15587/2706-5448.2023.286382.

54. *GDB: The GNU Project Debugger*. Available at: https://www.sourceware.org/gdb/ (accessed 06.12.2023).

55. Powers, D. *Evaluation: From Precision. Recall and F-Measure to ROC. Informedness. Markedness & Correlation.* arXiv 2020. Available at: 10.48550/arXiv.2010.16061. (accessed 06.12.2023).

56. Chicco, D., & Jurman, G. The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification. *BioData Mining*, 2023, vol. 16, iss. 1, pp. 1-23. DOI: 10.1186/s13040-023-00322-4.

57. Savenko, B., Lysenko, S., Bobrovnikova, K., Savenko, O., & Markowsky, G. Detection DNS Tunneling Botnets. *Proceedings of the 2021" IEEE 11th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, IDAACS'2021, Cracow, Poland, September 22-25, 2021. DOI: 10.1109/IDAACS53288.2021.9661022.

Received 05.01.2024. Accepted 20.02.2024

МУЛЬТИКОМП'ЮТЕРНІ СИСТЕМИ ВИЯВЛЕННЯ ЗЛОВМИСНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З МЕТАМОРФНИМ ФУНКЦІОНАЛОМ

Антоніна Каштальян, Сергій Лисенко, Олег Савенко, Андрій Нічепорук, Томаш Сочор, Володимир Авсієвич

Потреба в розробці нових систем виявлення та протидії шкідливому програмному забезпеченню залишається актуальною. Окрім методів виявлення шкідливого програмного забезпечення, все більшої актуальності набуває потреба в розробці нових систем виявлення та протидії шкідливому програмному забезпеченню. Використання різноманітних систем виявлення та формування в них змінної архітектури значно підвищує ефективність виявлення, оскільки як для зловмисників при комп'ютерних атаках, так і для шкідливого програмного забезпечення розуміння системи значно ускладнюється. Крім того, такі системи можуть містити приманки, пастки і, відповідно, модифіковані операційні середовища для обманного виконання програм з метою дослідження. У статті розроблено концептуальну модель багатокомп'ютерних систем, яка розроблена для забезпечення функціонування антивірусних приманок і пасток з метою виявлення шкідливого програмного забезпечення та комп'ютерних атак у корпоративних мережах. Запропонований підхід спрямований на запобігання та протидію проникнення метаморфних вірусів. Представлено концептуальну модель багатокомп'ютерних систем та введено визначальну характеристику, яка відповідає за контроль рішень та інші визначальні характеристики системи. Розроблено методи виявлення метаморфних вірусів з можливістю їх реалізації в архітектурі мультикомп'ютерних систем з приманками та пастками таким чином, що система безпосередньо через свої компоненти долучається до здійснення виявлення та приймає рішення про наявність метаморфного коду у виконуваному файлі. Здійснено реалізацію багатокомп'ютерної системи виявлення шкідливого програмного забезпечення з метаморфним функціоналом для доведення спроможності реалізації запропонованої концептуальної архітектурної моделі та розроблених методів виявлення метаморфних вірусів. Поставлено експеримент з функціонування багатокомп'ютерної системи виявлення шкідливого програмного забезпечення та проведено експериментальні дослідження. Проведені експерименти включали виявлення метаморфних вірусів. Крім того, було поставлено експеримент щодо ефективності виявлення метаморфного коду вірусів та проведено відповідні експериментальні дослідження. Також було досліджено ефективність виявлення метаморфного коду вірусів розробленою багатокомп'ютерною системою та встановлено наявність покращеного виявлення. Напрямки подальшої роботи полягають у поширенні результатів роботи на нові типи шкідливого програмного забезпечення.

Ключові слова: метаморфічний код; мультикомп'ютерні системи; кібербезпека; комп'ютерні віруси; шкідливе програмне забезпечення; виявлення шкідливого програмного забезпечення.

Каштальян Антоніна Сергіївна – канд. техн. наук, доц. каф. фізики та електротехніки, докторантка, Хмельницький національний університет, Хмельницький, Україна.

Лисенко Сергій Миколайович – д-р техн. наук, проф., проф. каф. комп'ютерної інженерії та інформаційний систем, Хмельницький національний університет, Хмельницький, Україна.

Савенко Олег Станіславович – д-р техн. наук, проф., декан факультету інформаційних технологій, проф. каф. комп'ютерної інженерії та інформаційний систем, Хмельницький національний університет, Хмельницький, Україна.

Нічепорук Андрій Олександрович – канд. техн. наук, доц., доц. каф. комп'ютерної інженерії та інформаційний систем, Хмельницький національний університет, Хмельницький, Україна.

Сочор Томаш – доц. каф. економіки та економічної політики, Університет Пріго, Чеська Республіка.

Авсієвич Володимир – студент, Хмельницький національний університет, Хмельницький, Україна.

Antonina Kashtalian – PhD, Associate Professor at the Department of Physics and Electrical Engineering, Doctoral Staff, Khmelnytskyi National University, Khmelnytskyi, Ukraine,
e-mail: yantonina@ukr.net, ORCID: 0000-0002-4925-9713.

Sergii Lysenko – Dr.S., Full Professor, Professor at the Computer Engineering & Information Systems Department, Khmelnytskyi National University, Khmelnytskyi, Ukraine,
e-mail: sirogyk@ukr.net, ORCID: 0000-0001-7243-8747.

Oleg Savenko – Dr.S., Full Professor, Dean of Information Systems, Professor of Computer Engineering & Information Systems Department, Khmelnytskyi National University, Khmelnytskyi, Ukraine,
e-mail: savenko_oleg_st@ukr.net, ORCID: 0000-0002-4104-745X.

Andrii Nicheporuk – PhD, Associate Professor at the Department of Computer Engineering & Information Systems Department, Khmelnytskyi National University, Khmelnytskyi, Ukraine,
e-mail: andrey.nicheporuk@gmail.com, ORCID: 0000-0002-7230-9475.

Tomáš Sochor – Associated Professor for Cybersecurity and Quantitative Methods, Department of Economics and Economic Policies, Prigo University, Czech Republic,
e-mail: tomas.sochor@prigo.cz, ORCID: 0000-0002-1704-1883.

Volodymyr Avsiyevych – Student in Khmelnytskyi National University, Khmelnytskyi, Ukraine,
e-mail: kovalleonid4@gmail.com.