ISSN 2347 - 3983
Volume 8. No. 9, September 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter264892020.pdf
https://doi.org/10.30534/ijeter/2020/264892020

Development of Approaches and Schemes for Proactive Information Protection in Computer Networks

Arzieva Jamila Tileubaevna[1]
[1]Karakalpak State University,
Uzbekistan, Karakalpakstan, jamka-1980@mail.ru

ABSTRACT
This paper proposes approaches to protecting information from the standpoint of the paradigm of their proactive security and a method for proactively protecting information from malicious code based on expert assessments. The architecture of proactive information protection based on scripts and regular expressions and the scheme of the decision-making module against malicious code for the implementation of more complex algorithms for proactive information protection will be constructed.

Key words: correctness, proactive protection, violator, intruder, secure computer systems, reliability, algorithmic approach, malicious software, API function, concatenation, disjunction, iteration, heuristic.

1. INTRODUCTION

Analysis of possible solutions shows that mathematical models of proactive protection against the actions of an intruder at the early stages of the software life cycle most likely cannot be numerous due to both the complexity of formalizing such solutions and the complexity of the solutions themselves. Thus, recently there has been an urgent need to create new software development technologies, initially focused on the creation of secure software products, when intruders act at the design stage.

2. TWO MAIN AREAS OF PROACTIVE SOFTWARE PROTECTION

The first direction is based on the so-called confidential computing protocols. There are n protocol participants or n processors of a computing system connected by a communication network. Initially, each processor knows its own part of some input value x. Calculate required $f(x)$, f – some computable function known to all participants, so that the requirements are met:

- correctness when the value $f(x)$ must be calculated correctly, even if some limited part of the participants arbitrarily deviates from the actions prescribed by the protocol;
- confidentiality, when, as a result of the protocol execution, none of the participants receives any additional information about the initial values of other participants.

It can imagine the following scenario for using this model to develop secure software. There is some process for which you need to implement functionality f. In this case, the consequences of an incorrect implementation are such that it seems appropriate to go on additional costs associated with creating a network of n processors and a distributed algorithm for implementing f. There is one more absolutely reliable participant in the system, which has access to the secret value x and has the ability to allocate its "share" to each processor x. The name Confidential Computing Protocols reflects the fact that the requirement for confidentiality is fundamental; value x must not fall into the hands of an attacker.

This model makes it possible to uniformly interpret both errors arising, for example, as a result of technical failures, and errors arising from their introduction into computational processes. It should be noted that confidential computing protocols refer to protocols that are primarily intended to protect the computing process from the actions of a "reasonable" intruder, i.e. from an attacker who always chooses the worst strategy for us.

The second direction is associated with the development of the so-called self-testing and self-correcting programs. Suppose you want to develop a program that calculates the functions f. Suppose that the implementation of this program is ordered by a contractor who does not enjoy full confidence.

The self-testing program is developed as a combination of two modules. The first one evaluates the function f, the second module tests the first one by feeding some values to it $x_1, \ldots, x_k$ and comparing the obtained result not with the previously calculated values of the function f, but among themselves. For this approach to have a right to exist, the testing module must be simpler than the most efficient algorithm for calculating a function f. It should also be noted that this module must be reliable.

The approach based on the idea of self-testing found its development in the so-called self-correcting programs. Such a program also consists of two modules, the first of which calculates the function f. It is assumed that this module may return erroneous (false) values. However, if this does not happen too often, then the second, corrective, module, choosing some values $x_1, \ldots, x_k$ and feeding them to the input of the first module, it will correct all errors by the obtained values and calculate the correct value of the function [1]. The corrective module is subject to the same performance requirements as the testing one.

The task of developing self-testing and self-correcting programs and their combinations is the following task.

Let it be required to develop software that implements the functionality f. The implementation of this software has been ordered by a contractor who is not fully trusted. However, the consequences of the negative work of this software are such that you can go to the costs associated

with the development of additional test modules, the creation of which is entrusted to a trusted specialist. Thus, self-testing software is a set of programs in which target programs are used as subroutines and which is designed for their effective testing.

The need to develop secure software using protected modules (PM) arises when it is required to ensure the authentication and integrity of complex software systems created by a large team of developers, among which there may be attackers. Such a module is a device protected from the enemy, in which, in the event of unauthorized access to it, the physical destruction of the main components of the module is carried out: processor registers and memory cells.

In order to achieve the required level of protection of such software, the work with confidential parameters entered into the PM is entrusted to trusted developers. In this case, it is proposed to solve the problem of developing secure software using the PM by developing a software and hardware package consisting of the PM, the program to be protected, and protocols of interaction between them.

These areas of protection can be used as the basis for proactive software protection, while initially it is assumed that:

one or several project participants are (or at least may be) intruders;

In the course of development and operation, an attacker can add to programs;

computer facilities, on which programs are executed, are not free of hardware backups.

Thus, the scientific and practical foundations for organizing activities to ensure proactive protection of software are a set of organizational and technical solutions, models and methods considered within the framework of this activity, which allow you to execute the following software development scenario. There is some process for which you need to implement functionality f. Moreover, the consequences of incorrect implementation f are such that it seems appropriate to go on additional costs associated with the creation of a network of n processors and a distributed algorithm to implement f, development of additional oracle programs with a call to the target computation program f, introduction to calculation schemes f protected modules. Then the protection schemes and protocols that implement the proposed methods can provide the correct functionality f, even if there are intruders among the software developers.

3. BASIC PROVISIONS OF PROACTIVE PROTECTION OF COMPUTER SYSTEMS

When creating complex computer systems, as a rule, the prevailing idea of the main stages of its life cycle is used. Such a view is effective in planning work, drawing up work schedules, and managing various projects [2]. It is advisable to divide the life cycle of the system into two parts, occurring at each of the stages, their technical and economic characteristics and factors influencing them.

In the first part of the life cycle, system analysis, design, development, testing and testing of software and hardware of computer systems are carried out. The nomenclature of work at each specified set of stages, their labor intensity, duration and other characteristics significantly depend on the object and the development environment. For these stages of the life cycle of computer systems, it is characteristic and extremely important to introduce certain protective functions into the created computer systems. Such a process is usually called ensuring the technological security of computer systems or, in a broader sense, proactive security of computer systems. It is characterized by the need to prevent the modification of computer systems through the introduction of destructive software and / or the introduction of destructive hardware embedded devices into the technical means of computer systems, as well as the need to introduce mechanisms to prevent malicious investigation of the system and system-wide software of computer systems.

The second part of the life cycle, reflecting the operation, maintenance and modernization of the components of computer systems, is relatively weakly related to the characteristics of the facility and the development environment. The range of work at these stages is more stable, and their labor intensity and duration can vary significantly and depend on the use of computer systems. For any model of the life cycle, ensuring the high quality of computer systems is possible only when using a regulated technological process at each of these stages. The stages of operation and maintenance of computer systems correspond to the process of ensuring operational safety or, in a broader sense, reactive safety of computer systems. This process is characterized by the need to protect programs from computer viruses and software bookmarks of a posteriori type and hardware from partial loss of their functionality. Destructive software can be introduced through the malicious use of software research methods and software specifications. The impact on the hardware of computer systems can be carried out due to various physical fields and / or a posteriori implementation of hardware embedded devices. In addition, vulnerabilities can also exist due to their untimely detection at the stages of testing software and hardware of computer systems or during their autonomous and complex testing.

Under considering the problems of creating proactively secure computer systems, it is initially assumed that “completely” a priori protected computer systems at the operational stage. In this case, the violator simply has nothing to do there [3]. Therefore, he may try to realize his malicious intentions when creating a computer system, and the scenarios of such actions will be characterized by the following "portrait of the intruder."

Two generalized types of violators are considered. The differences between them lie in their ability to interfere with the creation of computer systems.

An offender who interferes with this process, let's call him an active intruder, can:

- introduce intruders into teams developing various components of computer systems;
- introduce intruders who are able to perfectly study the "weak" points of computer systems and the features of the technologies used;

- carry out malicious changes originally defined for computer systems;
- to carry out a malicious choice of irrational work algorithms;
- introduce and use information technologies containing software bookmarks;
- take actions that may make it easier to add bookmarks or make them difficult to find;
- facilitate the implementation of supplies of computer equipment containing software, hardware or hardware-software tabs;
- form software bookmarks in a component of computer systems that affect other components of the system;
- organize the masking of the trigger mechanism of the software bookmark and other similar actions.

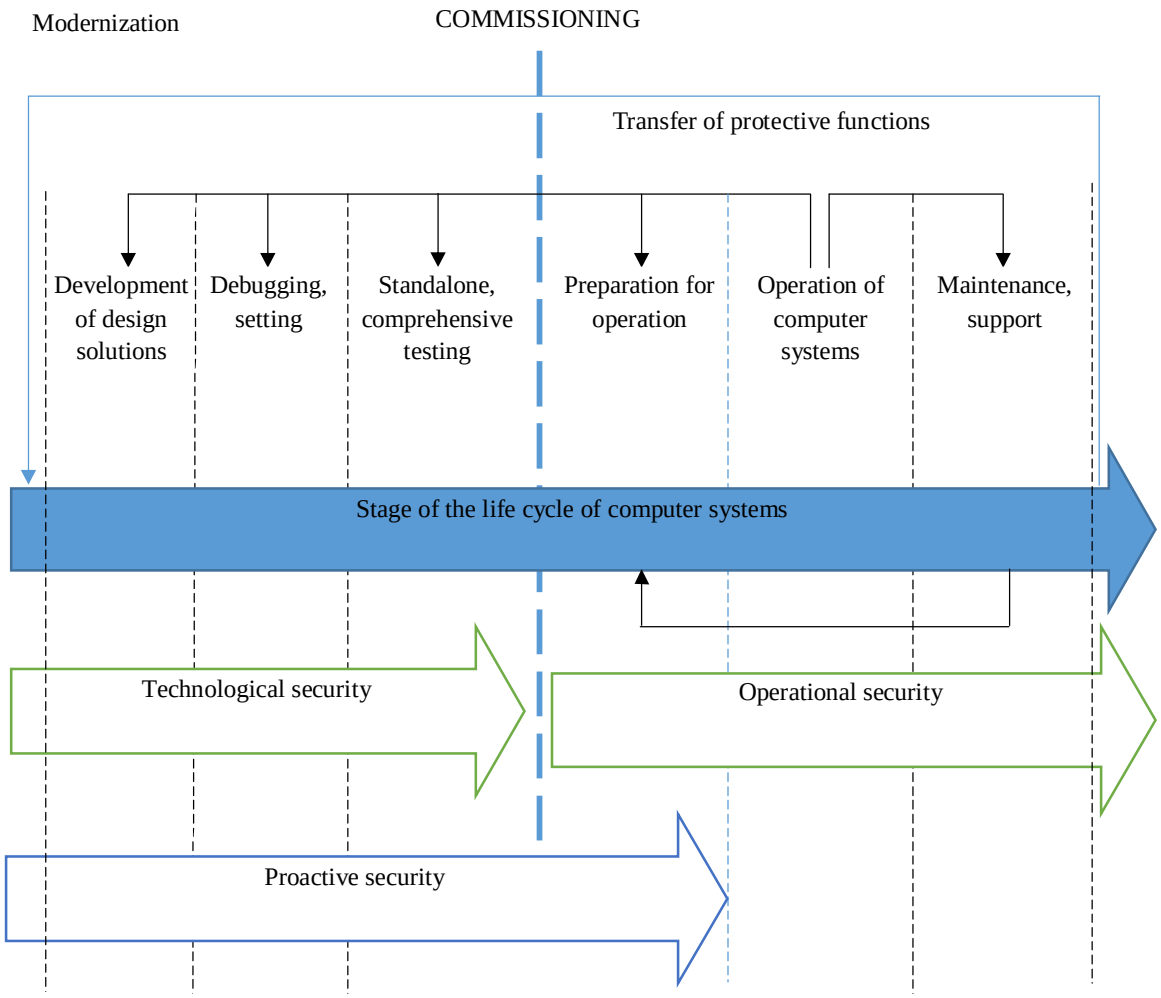

Figure 1: Methodological relationship between proactive security and the life cycle of computer systems

An intruder observing the design and construction of computer systems can:

– receive confidential information about the characteristics of the computer systems development process;
– identify information with specific developers and potential users;
– receive information about the characteristics of traffic interactions in distributed computer systems;
– read passwords, keys, other similar identification and authentication parameters and identify them with specific developers and potential users, etc.

Thus, the violator will be considered as a subject making unauthorized access to information and functional resources when creating computer systems in order to perform a wide range of malicious actions.

4. AN ATTEMPT TO FORMULATE THE PROBLEM OF CREATING PROACTIVELY SECURE COMPUTER SYSTEMS IN ACCORDANCE WITH THE NEW PARADIGM

As noted above, the central link in the creation of proactively secure computer systems is the shift of emphasis in ensuring information security from the operational stage to the earlier stages of the system life cycle (Fig.1).

At the same time, the earlier the introduction of protective procedures into the created computer systems begins, the more effective the protection in general will be [4].

Moreover, taking into account the problems of ensuring reliability and safety at the design stages reduces the level of resource costs for these purposes.

Methodological approach

It is natural to consider the activity of protecting the software of computer systems within the framework of normatively approved or generally accepted in the scientific and technical community concepts. In terms of goals and threats, the latter, as well as the present methodology of proactive security of computer systems, are essentially technologically converged. That is, there is a certain technological similarity in structure and function of concepts that are relatively different in origin.

Table 1 shows an illustrative analogy for proactive information protection.

Table 1: An illustrative analogy of proactive information protection

Concepts	The concept of survivability of computer systems	Trusted computer systems concept	The concept of creating high-confidence computer systems	Methodology of dynamic protection of computer systems	Methodology for proactive security of computer systems
Goal	Ensuring the ability of the system to perform its functions in time	Ensuring that the system performs the required functions	Ensuring well understood and predictable system behavior	Enforcement of proactive protection strategy	Ensuring the functioning of the system without the manifestation of various negative consequences
Threats	Threats to carry out attacks such as intrusions, DoS, DDoS attacks, threats of internal hardware and software failures and failures induced by external events (extraordinary events, disasters)	Threats of attacks by hackers and insiders, external influences, program errors and operator errors	Internal and external threats from natural sources and a "sophisticated and well-financed adversary"	Illegal Access Threats	Illegal research, copying, use and distribution of software

Significant differences in the listed concepts are mainly in the features of the description of the intruder and the taxonomy of vulnerabilities / threats / attacks for computer systems. Table 2 describes the relationship between these phenomena.

Table 2. The relationship between these phenomena

Property violation	**Source of threat**	**Vulnerability**	**Threat**	**Attack**	**Event**
Reliability	Violator, objective external circumstances	The presence of an unintentional defect	The manifestation of an unintentional defect due to design errors, incompetence of developers, etc.	Unauthorized, incompetent actions, emergency situations	Unwanted
Security computer system	Enemy	The presence of a deliberate defect	Introducing deliberate defects	Intensification of Intentional Defects	Malicious

Such an interconnection will make it possible to find "common ground" in conceptual and technological approaches of different origins to the creation of secure computer systems.

System-wide approach

Let be K – computer system designed to solve target problems $\Pi_1, \Pi_2, \ldots, \Pi_l$ There is also a set of supporting the process of functioning K tasks $O_1, O_2, \ldots, O_m$. Set tasks $\{\Pi_i\}, i = 1, \ldots, l$ и $\{O_j\}, j = 1, \ldots, m$ operate using resources $R_1, R_2, \ldots, R_s$.

For development K team is invited $D_1, D_2, \ldots, D_n$, At the same time, “good” and “bad” developers are distinguished, that is, there are two sets G and B respectively. At the same time, the entire background of the choice of developers suggests that the power $|B|$ does not exceed some predetermined limit t. It is clear that $|B| = n - t$. At the same time, bad, as opposed to good, means either incompetent (careless) or malicious developers.

Then some of the problem statements in such an informal scenario, taking into account the above verbal model of behavior (portrait) of a potential offender, may be as follows.

Informal setting 1. It is necessary to develop computer systems K, which, during operation, to develop computer systems would solve target problems Π_i, provided all developers are reliable.

Informal setting 2. Need to develop computer systems K, which would solve target tasks during operation Π_i, provided that the number of unreliable developers does not exceed t.

Informal setting 3. It is necessary to develop computer systems K, which would solve target tasks during operation Π_i, assuming unreliable developers (no more than t) - are passive offenders.

Informal setting 4. It is necessary to develop computer systems K, which would solve target tasks during operation Π_i, provided that the number of unreliable developers (no more than t) - are active offenders.

Informal setting 5. It is necessary to develop computer systems K, which would solve target tasks during operation Π_i, provided that unreliable developers (no more than t) - are both passive and active offenders.

In general, in the conditions of the above formulations of tasks, it is clear that we must solve some proactive protection tasks, when, under a number of conditions, reactive security follows from the proactive nature of the created computer system in terms of ensuring its functional and information security.

Let us abstract further from the presence of unreliable computer system developers. In this case, the general informal formulation of the problem of creating proactively secure computer systems can be formulated as follows.

Informal setting 6. It is necessary to develop computer systems K, which would solve target tasks during operation Π_i, at the same time, if possible, the solution of protection tasks aimed at the stable functioning of computer systems is transferred from the stage of its operation to the stages preceding the commissioning of computer systems.

Such a formulation “looks too informal”, nevertheless, according to the authors, it is precisely such formulations at the early stages of research that can be useful in understanding the proposed informal staging structures.

A very important point related to the resource costs for such a process of transferring protective functions is not considered here [5]. At the same time, the main resource constraints will be financial, and most importantly - temporary, since when ensuring proactive security of computer systems, protection issues are given increased attention at the stage of their creation. In this case, at the operational stage, it is assumed that the users of the CS “can simply not pay attention” to the need to protect its information and functional resources.

Thus, from a system-wide point of view, the paradigm of proactive security comes down to such a conceptual scheme for formulating a problem that would allow us to ensure stable target functioning of a computer system in the presence of a large number of disturbing factors when solving it.

Algorithmic approach

From the above paradigm for creating proactively secure computer systems, the following qualitative statement of the problem of creating algorithmically proactively secure computer systems can be formulated.

Let be Π – complex consisting of programs $\Pi_1, \Pi_2, \ldots, \Pi_i$ having a purpose determined by a specific computer system. At the same time, the issues of program interaction, their correlation, etc. are not considered. Let also n_i and R_i – program size Π_i and the size of the resources allocated by the computing system to protect the program Π_i respectively, and N – total size of the software package Π defined as

$$N = \sum_{i=1}^{i} n_i \quad (2.1)$$

The values n_i and R_i are expressed in either spatial, temporal, or spatial-temporal measure. Value R determines the total cost of protecting a complex of programs Π and is calculated as

$$R = \sum_{i=1}^{i} R_i \quad (2.2)$$

It is defined k_i as a security parameter for the program Π_i, and $K = \max(k_i)$ – as an aggregate safety parameter for the complex Π, where $I = \overline{1.i}$. In general, k_i determines the likelihood that the program will not violate certain security conditions, for example, the probability of the presence (absence) of software defects, the likelihood of successful (unsuccessful) actions of the intruder in the selected protection scenario, etc.

The values k_i and can be directly or indirectly related to each other. For example, the program Π_i size n_i can run faster on a 32-bit processor than on an 8-bit processor, and the security parameter k in a number of cases, it directly depends on the size of the computer word of the computing system.

Let be $E_{ni} = g(k_i)$ and $E_{ki} h(k_i)$ – direct and indirect effect of the implementation of the proposed methods (schemes and protocols) of program protection Π_i. In this case, the direct effect is understood as an increase in the level of software security for specific computer systems, and the indirect effect is understood as obtaining accurate quantitative characteristics that do not violate security functions by programs and the ability to confirm the correctness of the target function. Then the problem statement is as follows.

Informal setting 7. Under the assumptions and restrictions imposed by the conditions of functioning of a particular computer system, it is necessary to develop a set of programs $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_i\}$, which are correct with the probability of the presence of the program Π not higher k_i

calculate the result on a complete system of tests with cumulative costs R, which constitute a polylogarithmic or constant multiplicative factor of size N complex of programs Π.

The exclusion of the possibility of manifestation of destructive software in protected programs is possible due to the introduction of space-time redundancy into protection schemes. K the varieties of redundancy given in this work include:

- processor redundancy;
- temporary redundancy;
- communication redundancy;
- hardware redundancy due to the introduction of specialized protection means.

Thus, the task is to effectively implement the developed protection methods by reducing the space-time redundancy to the minimum possible value. In turn, redundancy is expressed in a certain number of additional processed, stored and transmitted information bits and in a certain additional amount of time for executing algorithms for the operation of programs.

5. Development of a method for proactive protection of information from malicious code

Interception of potentially dangerous actions. To implement the mechanism for intercepting potentially dangerous actions, it is proposed to use the technology of intercepting the SSDT system services table using an additional driver that works at the kernel level. This technology has a number of significant advantages. Working at the kernel level, the interception module allows intercepting all calls to the file subsystem and registry even at the early stages of the operating system boot. At the same time, all intercepted actions can be blocked in a timely manner [6]. The work of the interception module at the lowest level allows you to track the actions of stealth viruses that implement their camouflage at higher levels. Also, the implementation of the module as an OS driver significantly complicates the task of removing and bypassing the interception module for malware.

Scheme of a proactive defense system. Implementation of a proactive defense system based on a two-tier architecture, when a kernel-level driver is responsible for intercepting actions, and a user process for displaying information about events that occur, is not applicable in real conditions. If you transfer the functions of intercepting actions and the functions of making decisions about blocking to the driver, this will either lead to a large number of false positives, or significantly increase the total time of the function call and, as a result, slow down the operation of the entire system. In addition, direct interaction of the driver with the user process can lead to a violation of the stable operation of the operating system. This approach does not allow building a more reliable and correspondingly complex decision-making system and reduces it to the simplest version.

The scheme of a proactive protection system against malicious software is shown in Fig.2.

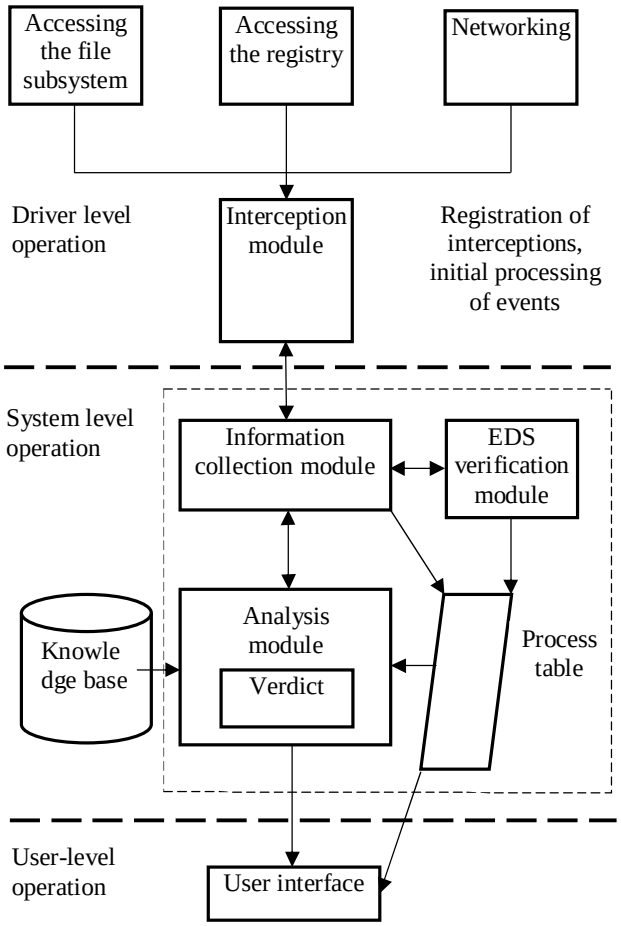

Figure 2: General scheme of a proactive protection system against malicious software

The proposed scheme is based on a three-tier architecture. At the lowest level, the module for intercepting potentially dangerous actions, implemented in the form of an operating system driver, works. He is responsible for intercepting calls to the file subsystem and the registry, and also monitors network communication. The module for intercepting potentially dangerous actions can also provide real-time blocking of unambiguously dangerous actions, the preliminary complex analysis of which is not required. Such actions may include, for example, an attempt to replace or modify the main system files.

The second level of the system is represented by a system service running with operating system rights. At this level, the accumulation and analysis of all potentially dangerous actions is carried out and a decision is made to block the operation of one or another process. The analysis module can be easily modified, can use any required amount of system resources, has the ability to access the local and network knowledge base, as well as the ability to verify the digital signature of all processes running in the system. The work of the analysis module separately from the

mechanism for intercepting potentially dangerous actions makes it possible to implement analysis algorithms based on the accumulation of information about the operation of programs over a certain period of time.

For direct interaction with the user, the user interface is responsible, which works at the highest level of the system. Its main purpose is to notify the user about the occurrence of dangerous events that were detected by the analysis algorithms and/or blocked automatically by the interception module. Note that the user does not take direct part in the decision to block programs within the proposed system. Moving the user out of the protection system can significantly increase the speed of the system's response and reduce the influence of the human factor.

Classification of actions according to the degree of danger. Any proactive malware protection system is based on intercepting potentially dangerous actions and their subsequent analysis. The efficiency of the entire system depends on how well the analysis of potentially dangerous actions is carried out. Obviously, not all potentially dangerous actions are created equal. Some of them are more dangerous, some less. All software actions can be divided into classes according to the degree of danger: safe, low level of danger, medium level of danger, high level of danger and especially dangerous. If the protection system responds in the same way to all unsafe actions, this will lead to a large number of false positives. In order to reduce their number in most modern proactive defense systems, actions belonging to the first three classes are simply ignored. In this case, actions that belong to the last two classes are processed in the same way, which also increases the likelihood of an error of the second kind and increases the load on the module for intercepting potentially dangerous actions.

Lack of control over actions related to safe, low and medium severity levels will not allow tracing the evolution of malware attacks aimed at infecting computer systems over time. Often times, safe actions can be the harbingers of more dangerous actions by malware. So, having entered the system, the virus first of all begins to analyze its new habitat. These actions are not considered dangerous and are ignored.

At the same time, if one of the useful work programs performs an action related to the group of medium or high severity level, it will be blocked by the antivirus. For example, writing data to an executable file can be classified as the action with the highest severity. Most viruses try to write themselves into executable files or create them on disk. But it can be just copying a file from one directory to another.

Thus, one division of software actions by hazard classes is not enough. It will be more effective to use the analysis method that takes into account the entire history of software actions. In this case, not only dangerous and especially dangerous actions should be taken into account, but even such safe actions as reading a directory and checking file attributes.

A large amount of information about the operation of a particular program makes it possible to increase the reliability of the decision-making subsystem and to realize the possibility of creating over time behavioral profiles for each of the programs.

Accumulation of information about software operation. For processing, the data analysis module can receive a huge stream of disparate information about the work of all processes running in the system. In this case, one of the key requirements for the analysis module is the requirement to minimize the use of system resources. This requirement can be met only by reducing the amount of stored operational information and the number of required computing operations. For this reason, the analysis module cannot store information about all actions of all running in computer systems. Instead, it is proposed to use several numerical indicators that could characterize all the previous actions of a particular program.

Under using numerical indicators, the amount of stored information for each program is reduced to only ten bytes, and the analysis of each new action requires only a recalculation of indicators taking into account this action. As a result, it becomes possible to reduce the amount of stored information and the number of calculations.

Application of the method of expert assessments. As noted above, in modern systems of proactive protection against malicious programs, when a potentially dangerous action is intercepted, a message is displayed to the user of the application with a request to allow or block this action. Making a competent decision requires a fairly high qualification and level of knowledge from the user, sometimes at the level of a system programmer or administrator. Obviously, this approach is unacceptable when it comes to the massive use of antivirus.

In the proactive defense system, a decision-making module should be used, which will allow shifting the decision-making onto the software product itself. At the same time, the initial data for its operation should be not only events coming from the module for intercepting the actions of potentially dangerous software, but also a previously prepared knowledge base based on the opinion of experts.

For the formation of the knowledge base, it is proposed to use the method of expert assessments, which will make it possible to streamline the knowledge and opinions of individual specialists. It is assumed that several highly qualified specialists will take part in the formation of the knowledge base, who will be able to give an objective assessment of certain actions taking place in the system. At the same time, instead of classifying potentially dangerous actions according to the degree of danger, it is proposed to use the classification according to how certain actions are characteristic of malicious programs.

To simplify the work of experts in assessing potentially dangerous actions, it is proposed to divide all actions into groups in accordance with the stages of the life cycle of a malicious program [7]. In total, six main stages can be distinguished: first launch, exploration, ensuring a restart, concealment of presence, reproduction and destructive

impact.

Each of the experts fills out the table, independently putting down points for each action on a ten-point scale. First, an independent assessment is put down for each of the actions, as if all the previous and subsequent actions are not important. After filling the first column, the expert proceeds to filling in the remaining columns. They contain an assessment of how unambiguously the considered action can be classified as an action of a malicious program, provided that actions from other groups were performed earlier.

Since the assessment was carried out in points, the final results of the work of the group of experts are averaged and reflected on an expanded scale (for example, by 100 points). The resulting tables are entered into the knowledge base of the program and are the initial data on which the program will rely when making decisions.

6. Constructing an architecture for proactive information protection based on scenarios of malicious code behavior

An expert system based on scenarios of malicious code behavior is proposed for use in proactive protection, namely, in antivirus heuristics and/or behavioral blocker. To simplify the presentation, we will consider it as part of a heuristic.

Why should there be an expert system? Why should it be an expert system, and not an ordinary program, for example, a data processing system? Because the description of behavior is not data, but knowledge. In computer science, there are definitions for both data and knowledge.

Definition 1. Data are separate facts that characterize objects, processes and phenomena of the subject area, as well as their properties.

Definition 2. Knowledge is the patterns of a subject area (laws, connections, rules) that allow experts to pose and solve problems in this area.

Knowledge is transformed during computer processing as follows:

1. Knowledge in human memory as a result of thinking.
2. Material carriers of knowledge - books, manuals, etc.
3. The field of knowledge is a generally accepted description of the basic objects of the domain, their attributes and rules, as well as the relationships between them.
4. A machine-based knowledge base that is close to natural language - understandable to the layman in the field of computer science.

Obviously, descriptions of malware behavior are knowledge, not data, since in our case we have not only facts, but also rules about how malware works.

And since we are dealing with knowledge, their processing requires an expert system that can make decisions about various computer objects based on the knowledge base about the behavior of malware.

As it knows, an expert system consists of three main parts: a knowledge base, a knowledge base management system that implements knowledge input, storage, editing, addition and translation into the internal representation, and a solver for processing knowledge when making decisions.

There are various models of knowledge representation, such as rules (if <conditions> then <action>), semantic networks, frames, scenarios (hierarchies of scenarios), as a special case of frames, and others. Thus, there can be different knowledge base management systems for different types of presentation. There are also various ways of processing knowledge, in particular, direct inference, reverse and combined.

Choosing a way to represent knowledge. You need to choose an appropriate knowledge representation to describe the behavior of malware. A behavior description contains actions, but actions can be represented by algorithms, rules, or scripts.

Usually algorithms are used to describe the behavior of programs. But in our case, it is necessary to describe the behavior of not every program, but entire classes of malware. The first scenario hierarchy is shown in Fig.3. In addition, it should be possible to describe behaviors at different levels of the hierarchy, so that the user can be given explanations not at the level of elementary actions, but using common available concepts.

Thus, we need some generalized hierarchical representation of the algorithms. For this, it is proposed to use scripts. And scripts are preferable to rules because they are closer to the algorithm than rules.

Representing knowledge using scripts. Scripts are designed to describe behavior, and each behavior usually has a purpose. So, in our case, the goal is what the malware is trying to do in order to destroy the system or its components. And thus, the target is explicitly or implicitly displayed in the script name.

How a malicious program could behave in order to achieve its goal, in order to then compare this behavior with the progress of the program under study during detection [8]. For example, if the goal is to infect executable files, the virus first searches for files, opens them, writes to the code section, and modifies the header. These steps will be referred to as sub goals.

Sub goals are intermediate stages in achieving a goal. Thus, they can be viewed as part of a script. Each sub goal can have different conditions. For example, "open file" is always executed after "find file". Some sub goals may be required, some are not. But such conditions can be described in various ways. For example, for the sub goal “write to the code section” we must “go to the code section”, “use the WriteFile API function”, and so on. Thus, as a result of constructions, a hierarchy of scenarios is obtained. The purpose of the top-level script is to infect an executable file, that is, to inject a virus in such a way that the infected program, on the one hand, remains functional, and on the other hand, serves as a source for the propagation of the virus.

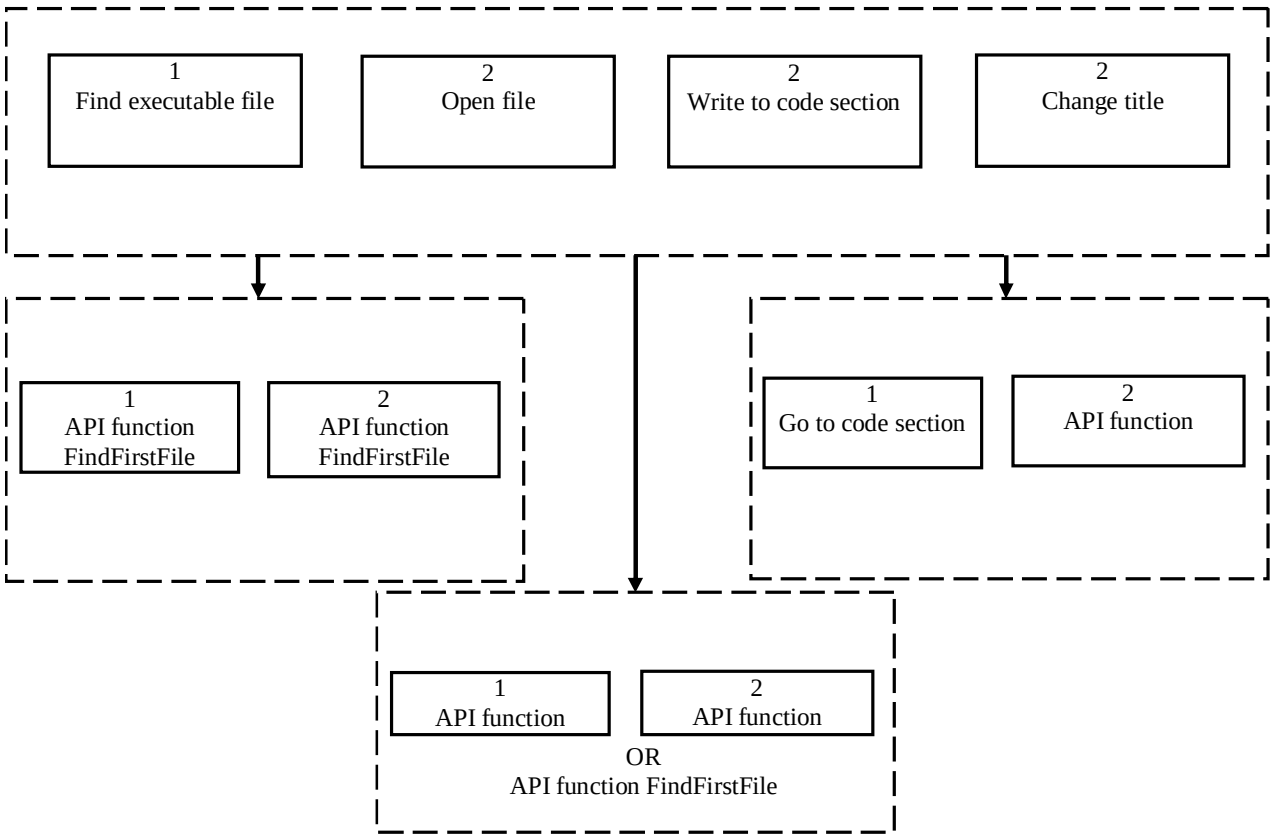

Figure 3: Hierarchy of scripts for describing the behavior of a parasitic virus

There are two types of sub goals: baseline and sub goals based on baseline. Basic sub goals correspond to the actions that we intercept in the system, for example, calling an API function. Thus, moving from the goal, we define its sub goals, then these sub goals through others, and so on until all sub goals are directly or indirectly defined through the base ones.

For the first script hierarchy, shown in Figure 3, the basic sub goals are: the FindFirstFile API, the FindNextFile API, the WriteFile API, the CreateFile API, and the OpenFile API. Some types of basic sub goals and their formal descriptions are listed below.

Thus, each script from the hierarchy consists of a name that indicates the purpose of the script and a list of sub goals.

Now you need to determine how to formally write scripts, and what types of sub goals will be basic. For writing scripts, we suggest using a language similar to the regular expression language. Let's consider three basic operations and two derivatives from them. Basic operations:

1. Concatenation. Denoted by a hyphen «−». $a - b$ means sequential execution of sub goals, first a, then b (e.g. Find File followed by Open File).

2. Disjunction (OR). Indicated by the sign «|» $a \mid b$ means that a or b.

3. Iteration. Indicated by the sign «*». a^* means that scenario a can be repeated any number of times (including zero).

Architecture heuristic. Detection of malicious code using a heuristic goes through three phases: a decoding phase, an investigation phase, and an assessment phase. The first two phases refer to the technical component, the last one to the analytical one.

The purpose of the decoding phase is to emulate the required number of instructions required for a virus to decode its body.

The purpose of the research phase is to emulate, at least once, all code sections available in the program, which can presumably contain viruses.

The purpose of the evaluation phase is to analyze any suspicious actions that were found during decoding and research to determine if the program is infected. This phase is divided into two.

The first is compiling a list of all observable program behaviors using static and dynamic approaches.

The second is the analysis of the identified behaviors. This is where it is determined whether the set of detected behaviors is malware-specific or not. The knowledge base is used for this. The decoding phase, research and the first part of the evaluation phase are outside the research. It is the second

part of the assessment phase that will be an expert system based on scenarios, that is, it will be an ES solver, which, based on inferences, will be able to analyze and compare scanned files with scenarios for each type of malware or

with scenarios of different types of malicious behavior. The complete architecture of the heuristic is shown in Fig.4. It consists of two subsystems:

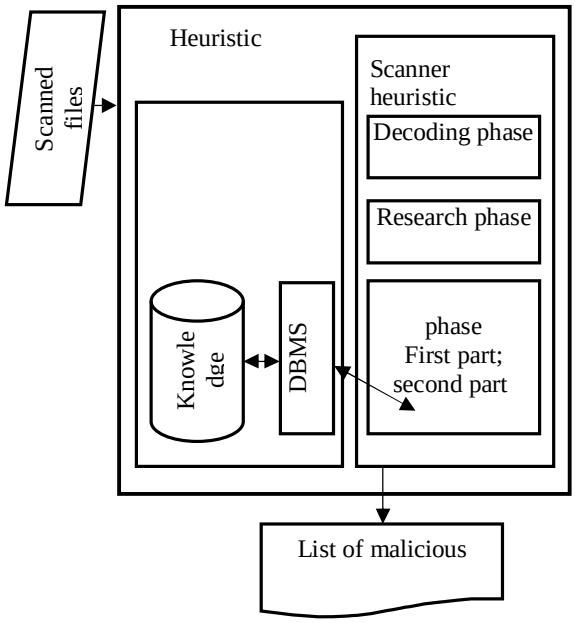

Figure 4: The architecture of a heuristic using an expert system

1. A knowledge base management system with a knowledge base based on hierarchies of malicious behavior scenarios. The knowledge base management system is designed to organize the input of knowledge, store it, edit it, add it and translate it into the internal representation.
2. A heuristic scanner, including an expert system solver, is necessary to determine the infection of the object under study.

7. Scheme of the decision-making module for proactive information protection from malicious code

A scheme of the decision-making module is shown in Fig.5.

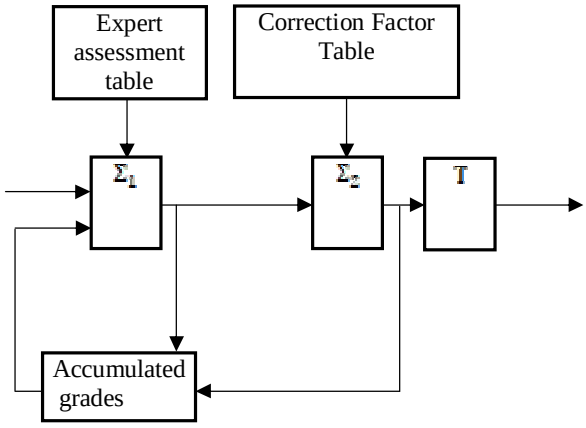

Figure 5: Decision module scheme

Only three tables of limited size are used as stored data. Each of the tables contains no more than a hundred bytes per process. Lists of all accumulated actions are not used. As a result, it is possible to minimize the use of RAM resources. All data operations are performed by two adders, which also significantly reduces the amount of required computing resources. The first adder is used to recalculate the cumulative score table data. The second one is for calculating the current assessment of the harmfulness of the process that performed the action that came to the input of the decision-making module. Accumulated scores s, recalculated for each action according to the formula:

$$s_{i,j} = S_1.S_1(s_{i,j}, t, p, action) = g(s_{i,j}, t) + x_i \cdot q_p \cdot \sum_j v_{i,j} \cdot action_j \cdot g(s_{i,j}, t) \quad (1)$$

where

$g(s_{i,j}, t)$ – function of changing the estimate over time t;

q – process correction factor;

p – process index in the correction factor table;

$action_j$ – assessment of the action according to the corresponding column of the table of expert assessments.

The adjustment factor table reflects the degree of initial trust in a particular process. For example, a lower coefficient can be set for processes whose executable module is equipped with an EDS of a well-known software manufacturer.

The second amount is calculated as the sum of the accumulated scores multiplied by the corresponding coefficients:

$$s_i = S_2.S_2(s_i, t) = g(s_i, t) + \sum_j v_{i,j} \cdot s_{i,j} \quad (2)$$

where s_i – previous value.

Calculation result S_2 used to make a verdict on whether the program is appropriate for the process p to malicious. The decision is made when a certain predetermined decision threshold is exceeded T, simple comparison with a given quantity. The quantity T, at which a program can be recognized as malicious is selected experimentally and strongly depends on the original table of expert assessments.

8. Conclusion

It should be noted that the proposed approaches to information protection from the point of view of the paradigm of their proactive security make it possible to ensure stable target functioning of a computer system in the presence of a large number of disturbing factors, and the developed method and a three-level scheme of proactive information protection from malicious code based on expert assessments makes it possible to significantly reduce the likelihood of false alarms and the level of requirements for the qualifications of service personnel. Also, the constructed architecture of proactive information protection based on scripts and regular expressions made it possible to represent the behavior of

malicious code in a hierarchical form, and the developed scheme of the decision-making module against malicious code allowed making the right decisions independently without contacting the user.

REFERENCES

1. Bojanova, I., Vaulx, F., Zettsu, K., Simmon, E., Sowe, S. (2016, January 21). **Cyber-Physical-Human Systems Putting People in the Loop**. IT Professional.
2. Burning Glass Technologies. (2015). **Job Market Intelligence: Cybersecurity Jobs**, 2015.
3. Gulomov Sherzod, Xoshimova Charos Saidaminovna, Ganiyeva Toxira Irkinovna, Djurayeva Shoxista Tagirovna. **Analysis of Methods for Measuring Available Bandwidth and Classification of Network Traffic**. International Journal of Emerging Trends in Engineering Research. Volume 8. No. 6, June 2020. Indexed in Scopus. – P. 2753-2759
4. Costanzo, J. (2017). Hampton Roads Cybersecurity Education, Workforce, and Economic Development Alliance (HRCyber) Mid-Project Report "**Bridging the cybersecurity talent gap in Hampton Roads**".
5. Karimov Madjid, Gulomov Sherzod, Yusupov Bokhodir. **Method of constructing packet filtering rules**. International conference on information science and communications technologies applications, trends and opportunities (ICISCT). 4-6 November, 2019, Tashkent Uzbekistan
6. Herrmann, A., Rehm, K., Carlini, J., Schkeeper, P., Loken, L., Swanson, J., Maltby, J. (2009, October). **The CIP Report Center for Infrastructure Protection**, Volume 8 number 4.
7. Shark, A., Metzanbaum, S., Barquin, R., Wennergren, D. (2015, August). **Increasing the Effectiveness of the Federal Role in Cybersecurity Education**. National Academy of Public Administration.
8. Gulomov Sherzod, Karimova Dilbar, Akbarova Shokhida Azatovna, Qosimova Gulnora Ismoilovna. **Comparative Analysis of Methods Content Filtering Network Traffic**. International Journal of Emerging Trends in Engineering Research. Volume 8. No. 5, May 2020. Indexed in Scopus. – P. 1561-1569