Detecting Software Malicious Implant Based on Anomalies Research on Local Area Networks

Andrii Nicheporuk[0000-0002-7230-9475], Vadym Paiuk[0000-0002-7253-893X], Bohdan Savenko[0000-0002-9969-8239], Oleg Savenko[0000-0002-4104-745X] and Olena Geidarova[0000-0002-7253-893X]

Khmelnytsky National University, Khmelnytsky, Ukraine
andrey.nicheporuk@gmail.com, vadympaiuk@gmail.com, savenko_bohdan@ukr.net, savenko_oleg_st@ukr.net, geydarova@ukr.net

Abstract. The paper analyzes malicious software implants that use undocumented software features on local area networks. They can cause significant harm both users of personal computers and enterprises that utilize computer networks and use specialized software. In order to detect this type of malware, its possible models and behavioral scenarios, features, stages of research in local area networks have been proposed. Based on this data, a method for detecting computer anomalies has been developed, which is part of a general process for detecting malicious software implants that use undocumented software features. The result of the method is a division of computers on a local network into classes in purpose for further investigation of behavioral patterns. Thus, groups of computer are highlighted in which similar profiles have been formed, that in the overall scheme allows to improve the accuracy of detection. The adopting of the developed models of software implants that use undocumented software features, as well as a method for detecting computer anomalies, have allowed to carry out experimental researches with the use of distributed detection system. The results of the experiments have shown the correctness of the proposed detection enhancement solutions.

Keywords: Software Malicious Implant, Local Network, Computer, Malware, Czekanowski Diagram.

1 Introduction

According to research [1] of IDC firm and the University of Singapore, malware-related security breaches cause users worldwide damage of at least on $ 500 billion annually. Moreover, the number of malicious software is growing every year [2]. The most relevant for the benefit of attackers are organizations and enterprises that operate information technology on local computer networks. There are many ways to gain entry into the local computer networks of businesses (organizations) for the purpose of unauthorized access to information in them. One way for attackers to access enterprises (organizations) information resources is to use undocumented capabilities in

the software and hardware of computers and peripherals that allows unauthorized access to system resources, typically, via a local network. This is achieved through the using of software implants, which primary purpose is to provide unauthorized access to sensitive information.

Software implant is a secretly implemented program which poses a threat to the information contained on the computer [3]. The software implant can be implemented as a separate malware, or as undocumented software code in the software.

We will consider the software implants that use undocumented software features on the local computer networks of enterprises (organizations) as an object of research. The difficulty of detecting such a secretly functioning software object, which under certain conditions is capable of providing unauthorized access, is due to the absence of its activity during a long time. As a rule, such software implants allow to keep software features and are implemented by some else of the features included in the software package.

An enterprise may use ready-made software that already has software implants, or software made to order, which has been poorly verified upon commissioning. Software that runs inside enterprise LANs is typically a distributed, which makes software implants are active on all computers on the network. This increases the threat to businesses and organizations. Software implants can be used to create botnets [4], implement trojans, or produce metamorphic or polymorphic components [5], and so on. Therefore, the scientific problem of detecting software implants on the local area networks is relevant.

One of the primary tasks that need to be addressed is to develop appropriate methods for creating effective components of a comprehensive system for detecting software implants on local area networks.

2 Related works

Software implants detection studies are presented in many ways that depend on the considering of specific malware types. The main problem that accompanies the process of software implants detection is the discrepancy between what the user sees and what actually happens [1]. To achieve these results, the attackers have developed quite a few tools and approaches.

The most common research in this area is Backdoor malware detection studies [6-12]. In [6] authors proposed a new approach to detecting and removing Backdoor malware using neural networks. The experimental results obtained are based on the classification made. The work focuses on such type of attack that allows attackers to insert a hidden function or hidden program code into a malicious action model. Detection of such types of malware is difficult task, because unexpected behavior occurs only when there is a launch to execute a hidden function or program code known only to the attacker. The adequacy of the proposed solution requires more convincing evidence since a small set of validated and reliable datasets was used [6].

In [8] presents a model used by intruders to hide the invasion path. One of these techniques is done by using multiple hosts on the network, which can be detected

using the approach suggested by the authors. Authors explored the opportunity the use of this approach to detect others type of malware, including Backdoor. The study shows that the proposed approach can produce a very low rate of false negative and false positive and allow to reduce the detection time of the scanning process.

In [9] it is analyzed and substantiated that, due to the complexity of the studied topic, there are few effective solutions. One way analyzed by the authors is to encode code fragments using specially designed interrupts that, when triggered, manipulate the run time state and, under certain conditions, can perform arbitrary computations without fail.

In [10] proposes to solve the problem of formalizing a terminal machine by modeling a program or system using a so-called machine with a designated end state that allows one to treat the software fragment as an emulator. Also authors have developed the concept of a multi-step game where an attacker and a defender get to take turns interacting that allows thinking about it as the system with states and transitions between them.

In works [11, 12] analyze the complexity of the problem with using many tools. This could jeopardize the platform by other means. For example, this is may be a hardware component, a custom program or a piece of malware. And this is a prerequisite for development methods for detecting them. A significant obstacle to the effectiveness of this approach is the lack of test samples.

Another equally malicious tool is the implementation of the software and the launch of the secret exchange feature. The features of such a hidden function are presented in [13]. To address this in [14] developed five steps to identify a hidden function in software.

In [15], models of hidden schemes of function exchange are considered and analyzed. They take into account the availability of the secure protocol of distributed information transmission. The reliability of such a protocol under the conditions of existence of hidden functions is investigated.

Another type of malicious software which can introduce software implants is a botnets [16, 17]. In [16] a technique for botnet detection based on a DNS-traffic is presented. Botnets detection based on the property of bots group activity in the DNS-traffic, which appears in a small period of time in the group DNS-queries of hosts during trying to access the C&C-servers, migrations, running commands or downloading the updates of the malware. The method takes into account abnormal behaviors of the hosts' group, which are similar to botnets: hosts' group does not honor DNS TTL, carry out the DNS-queries to non-local DNS-servers. Method monitors large number of empty DNS-responses with NXDOMAIN error code.

In [17] a DNS-based anti-evasion technique for botnets detection in the corporate area networks is proposed. Authors have combine of the passive DNS monitoring and active DNS probing and have construct BotGRABBER detection system for botnets in corporate area network, which uses such evasion techniques as cycling of IP mapping, "domain flux", "fast flux", DNS-tunneling. BotGRABBER system is based on a cluster analysis of the features obtained from the payload of DNS-messages and uses active probing analysis.

In [18-26] it is shown how the use of metamorphic transformations makes it possible to hide program codes of functions. Along with polymorphic technologies [27], these methods are quite effective and widely used by attackers.

The works in [28-30] analyzed the use of known mathematical methods for processing events related to software operation. The discussed methods can be used to detect software implants that use undocumented software features, but only after the process of processing big data obtained during monitoring has been completed.

Thus, software implants that use undocumented software features [31-36] pose a problem for computer users, especially when they can be distributed on local computer networks. Known detection methods target to specific subclasses or typical classes of malware and are not sufficiently represented in related works. The various mathematical methods used for detection process, means require the initial stages of preparation of the data for processing, which aims to design a comprehensive approach of detection.

3 A method for detecting anomalies in the computer systems based on the search for deviations from the mean values of the behavior profiles

In order to detect software implants that use undocumented software features, let's build profiles of computer systems (CS) based on behavior of software in which it is executed. Because the client side of the software is the same, they should have similar behavioral profiles. System profiles can be clustered in the local area networks (LANs) into CS groups that use typical parts of specialized software. Such profiles can range from one to 5-10, because specialized software has a narrow focus and, therefore, cannot be sprayed for various purposes. We define these profiles as formalized models of software behavior in CS. After some time of functioning of the specialized software and the available software in the CS the statistics is collected, that is use for the refinement of profiles. Developing a formal representation of behavior profiles is based on a numerical expression of the features. After receiving profiles, the system functions and, on a daily start, analyzes the similarity of profiles and results of the functioning of the software in the CS. The presence of clustering profiles allows to more accurately identifying possible anomalies in the CS that belong to certain groups.

To detect software implants that use undocumented software features in LANs, it is important to create a set of their behavioral signatures. In order to form a database of behavioral signatures, appropriate models of software implants were developed based on scenarios of their functioning. The implementation of software implants that use undocumented software features at different stages of the software life cycle can occur by the following scenarios:

1. Work of attackers inside software development team.
2. Creation of dynamically formed commands or parallel computing processes.

3. Implementation of command forwarding and recording malicious information in the memory of information system.
4. Inserting software implants that use undocumented software features into the code.
5. Creating a masked trigger mechanism for software implants that use undocumented software features.
6. Inserting software implants that use undocumented software features into separate routines and into the management program.
7. Preparation of test data for the detection of software implants that use undocumented software features.
8. Hiding software implants that use undocumented software features by making bugs in the software;
9. Placing software implants that use undocumented software features in the branches of the software that are not verified under testing.
10. The participation of the attackers in the software verification;
11. Development of software implants that use undocumented software features during software revision.
12. Development of updates and additions for software.

Scenarios of introducing of software implants that use undocumented software features are directly dependent and affect on their structure, so they may be separate entities or parts of another entity. Let's present all scenarios as graphs and formalize them with respect to the structure of the software implants that use undocumented software features, as well as all possible combinations of them.

As an example, on the fig. 1 has shown a graph of an irregular Markov process.

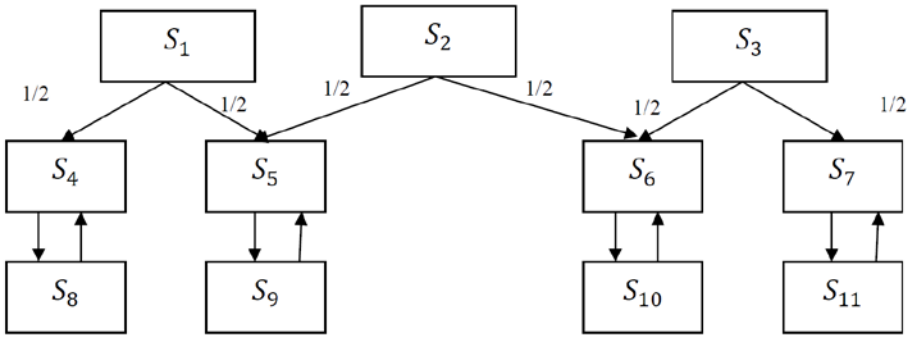

Fig. 1. Graph of irregular Markov process

Here, S_1 - software implants that use undocumented software features in a separate file, S_2 - software implants that use undocumented software features in a working file, S_3 - software implants that use undocumented software features hidden partially in two or more files. Suppose with equal probability $p_i = \frac{1}{2}$ information on S_1, S_2, S_3

arrives in pairs on states S_4, S_5, S_6, S_7. The states S_8, S_9, S_{10}, S_{11} will be counted. For example, if more information is obtained from states S_{10}, S_{11}, then it will indicate that software implants that use undocumented software features are placed in several working files of the program.

We detect software implants that use undocumented software features by manifestations which are based on file analysis and network activity. These manifestations depend on the models of functioning and structure incorporated in them. These features are: presence of software modules that do not meet the purpose of the process; presence of operating system objects that are open by the process but do not conform to the purpose of the process; high intensity of input/output operations for a certain process; a high CPU or internal memory usage from a certain process; the similarity of the file name to the system file name; the operating system process executable file is not in the common directory; the system process is run on behalf of the local user; code enforcement in the data area, which is enabled for all processes, is disabled for the certain process; the system process has a directory other than what it must be for that process; the absence of digital signature in the executable; high network activity of the process, which must run locally, etc. However, to improve detection efficiency, it is need tools that allow to establish the fact of the threat without the intervention of a network administrator who may not be able to process certain attributes for various reasons. Software implants that use undocumented software features may use masking tools on the system, making it difficult to detect them. On the technical side software implants use programming methods that are not common when creating standard software, so these features can also be additional attribute to identify them. In particular, for a Windows executable, these features can be: an additional section at the end of the file; an entry point indicates a transition to the middle of a section that is not a code section; the entry point correspond to a jump command that specifies a jump for the code section; the presence of features that indicates a code section, which is not a code section. Similarly for other environments on the CS, information from which may be related with RAM.

Let's describe the models of software implants that use undocumented software features on LANs:

1. software implants are injected in the software, stores all or selected pieces of software, entered or displayed in the hidden area of local or remote external direct access memory; the object of storage may be keyboard inputs, documents that will be printed; this model requires external storage, which must be organized in such a way as to ensure that it is stored for a specified period of time and can be further removed and hidden from other users or processes;
2. software implants are embedded in network or telecommunication software; As a rule, this software is always active, so software implants control the processing of information on the computer, performing installation and deletion other implants, as well as removing the accumulated information; software implants that use undocumented software features can trigger events for previously implemented implants;

3. software implants that use undocumented software features transmit information stored by an attacker (such as keyboard input) into a communication channel, or store it without relying on the guaranteed possibility of further reception or removal; software implants may also initiate continuous access to information, leading to an increase in signal-to-noise ratio when intercepting side radiation;
4. software implants that use undocumented software features distort data streams that occur during applications running (source streams), or distort input streams of information, or initiate (or suppress) errors that occur when running applications;
5. software implants which do not produce direct affect on software. The main purpose is to maximize the resulting "residual" information for further study; the attacker either obtains these fragments using the implants of the previous models, or directly accesses the computer in the guise of repair or diagnosis.

To detect software implants that use undocumented software features it is need to detect anomalies in a particular CS based on searching of deviations from the mean values of the behavior profiles. Thus, it is need to develop a detection method based on a combination of anomaly detection technologies and behavioral signature matching.

The implementation of the detection of software implants that use undocumented software features is based on the further development of information technology, which will include profile models, models of software implants and method of anomaly detection and theirs applying on LANs to investigate specialized software.

To determine the software profile in the CS, we use the system call monitor [42]. First, let's form API call sequences for each of the processes over a long time and build a software profile in the CS. After profile creation their clusterization is done. And the last step, if it is possible to divide profiles by more than one class, analysis of the obtained classes of CS is conducted.

For the study we use the anomaly search scheme. In order to reduce the number of input, grouping of similar values was done. The resulting profile scheme is a partial case of multidimensional analysis, where multiple objects are considered on many grounds. When processing statistic data in multivariate analysis [4, 5, 6], taxonomic methods have been used that do not require expert evaluation but are based only on observation results. The input for the study is the observation matrix:

$$X = \begin{bmatrix} x_{11} & x_{21} & \dots & x_{1k} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2k} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_{i1} & x_{i2} & \dots & x_{ik} & \dots & x_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_{s1} & x_{s1} & \dots & x_{sk} & \dots & x_{sn} \end{bmatrix} \quad (1)$$

where n is the number of features observed on the objects; s is number of objects; x_{ik} – the number of manifestations of k-th feature in i-th object during the observation period. The features normalization is carried out according to:

$$V_{ik} = \frac{x_{ik}}{\sum_{i=1}^{s} x_{ik}} \tag{2}$$

After applied formula 2 the matrix V was created:

$$V = \begin{bmatrix} V_{11} & V_{21} & \dots & V_{1k} & \dots & V_{1n} \\ V_{21} & V_{22} & \dots & V_{2k} & \dots & V_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ V_{i1} & V_{i2} & \dots & V_{ik} & \dots & V_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ V_{s1} & V_{s1} & \dots & V_{sk} & \dots & V_{sn} \end{bmatrix} \tag{3}$$

Then the isotonic coefficient that shows the position of the S-th object on the set of all features was used:

$$W = \begin{bmatrix} W_1 \\ W_2 \\ \dots \\ W_i \\ \dots \\ W_S \end{bmatrix} \tag{4}$$

where W_i defined as:

$$W_i = \sum_{i=1}^{n} V_{ik} \tag{5}$$

According to matrix (4), it is possible to arrange the objects by isotonic metric, that is, by the rank that characterizes the object by the set of features. Next stage involves structural (isomorphic) ordering of objects. To do this, with the matrix X, the matrix Z is formed by using formulas (4) and (5):

$$Z_{ik} = \frac{x_{ik}}{\sum_{i=1}^{s} x_{ik}} / \sum_{i=1}^{n} \frac{x_{ik}}{\sum_{i=1}^{s} x_{ik}} \tag{6}$$

$$Z = \begin{bmatrix} Z_{11} & Z_{21} & \dots & Z_{1k} & \dots & Z_{1n} \\ Z_{21} & Z_{22} & \dots & Z_{2k} & \dots & Z_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ Z_{i1} & Z_{i2} & \dots & Z_{ik} & \dots & Z_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ Z_{s1} & Z_{s1} & \dots & Z_{sk} & \dots & Z_{sn} \end{bmatrix} \tag{7}$$

The distance matrix in D is then constructed using formulas (4) and (5):

$$d_{im} = \{ \frac{1}{N} \sum_{k=1}^{n} (Z_{ik} - Z_{mk})^2 \}^{\frac{1}{2}} \tag{8}$$

Sometimes the average absolute difference of features values is used, which is determined by the formula:

$$d_{im} = \frac{1}{N} \sum_{k=1}^{n} | Z_{ik} - Z_{mk} | \tag{9}$$

$$D = \begin{bmatrix} 0 & d_{12} & ... & d_{1k} & ... & d_{1s} \\ d_{21} & 0 & ... & d_{2k} & ... & d_{2s} \\ ... & ... & ... & ... & ... & ... \\ d_{i1} & d_{i2} & ... & 0 & ... & d_{is} \\ ... & ... & ... & ... & ... & ... \\ d_{s1} & d_{s2} & ... & d_{sk} & ... & 0_{sn} \end{bmatrix} \tag{10}$$

Thus, the square matrix $S \times S$ is obtained, where the number of rows and the number of columns is equal to the number of objects. In this matrix it is possible to distinguish subsets for homogeneity. The easiest way to do this is to apply the Czekanowski method [5, 6, 7, 8].

The numerical values in the distance matrix are replaced by labels, for example:

× – distance $B < 0.4$

○ – distance $0.4 < B < 0.8$

• – distance $B > 0.8$

As a result a disorderly diagram of Czekanowski will be obtained. To get an ordered diagram, the rows and columns are rearranged so that they are as close to the main diagonal as possible. For example, the ordered Czekanowski diagram is given in table. 1.

Table 1. Ordered Czekanowski diagram

	1	2	3	4	5	6	7	8
1	×	×	×	×	○	○	○	•
2	×	×	×	×	○	○	•	•
3	×	×	×	×	○	○	•	•
4	×	×	×	×	•	•	○	•
5	○	○	○	•	•	•	○	○
6	○	○	○	•	•	×	•	•
7	•	•	•	○	○	•	×	•
8	•	•	•	•	○	•	•	×

The diagram shows that the objects 1, 2, 6, 7 are grouped together and have the smallest deviation (with all group) from the main diagonal. In our case, these may be computers that have specialized software installed. Objects 5 and 3 are form own two subsets that are far from the main diagonal. At the same time the outermost objects belong to numbers 4 and 8.

Thus, it is possible to divide software profiles in the CS into classes and, further, conduct analyzes the deviations in the classes with purpose of anomalies searching.

4 Experiments and evaluation

In order to conduct series of experiments a distributed system [43] for detecting malware was used. Software implants that use undocumented software features were developed as part of each of the typical botnets. The purpose of the experiments was verification of the botnets detection method, the efficiency of the classifier in the structure of the distributed system and determination of the dependence of the percentage of detected nodes of the botnets, which had contained software implants. To perform the experiments, 28 botnets and codes of known detected botnets were constructed [44]. All generated botnets were grouped by classes. From generated botnets 25 structural elements in three stages of operation and 81 functions were allocated.

The experiment was conducted for the classifier without adding instances of the created botnets and with them, that is, the check was performed without training the classifier on the created samples and with the preliminary classification of the samples by classes. The second variant is necessary to check the accuracy of classifying the same samples from which this class is built. This is important because during monitoring API functions may occur errors. The monitoring time of the CS in LAN was 350 hours for each instance of the botnet of each of the two classifiers. It is should be noted that the functionality of the botnet nodes was simplified and did not include the attack option. The botnet nodes only worked in control and support modes of own functioning. Thus, for a distributed system the objects of research were running processes on CS. In order to conduct the experiment botnets that use the strategy of obtaining full control by activating their components were selected. That is software implants that use undocumented software features were presents on each CS. In order to obtain software profile in the CS we perform API monitoring call. Based on the obtained profiles the features vectors are formed. After feature vector creation their clusterization is done. The results of calculation different metrics are presented in table. 2.

The experiments involved determining the following metrics for the detection of bot nodes:

$P_{1,1}$ – the percentage of vectors of malicious actions belonging to certain class relative to all test samples that the system assigned to this class with previous training;

$P_{1,2}$ – similar to metric $P_{1,1}$, but without previous training;

$P_{2,1}$ – the percentage of malicious action vectors belonging to a given subclass of a class relative to all test vectors that the system assigned to that subclass of the class in

the test sample (those that were correctly assigned to the subclasses) with previous training;

$P_{2,2}$ – similar to metric $P_{2,1}$, but without previous training;

$P_{3,1}$ – the percentage of correctly detected botnet nodes with previous training

$P_{3,2}$ – similar to metric $P_{3,1}$, but without previous training;

$P_{4,1}$ – the percentage of incorrectly classified botnet nodes as benign applications (type I error) with previous training;

$P_{4,2}$ – similar to metric $P_{4,1}$, but without previous training;

$P_{5,1}$ – the percentage of incorrectly assigned bot nodes to one of the botnet classes (type III error), with previous training;

$P_{5,2}$ – similar to metric $P_{5,1}$, but without previous training.

The results of evaluating the efficiency of detection the software of botnets' nodes based on the work of two classifiers for the entered classes and subclasses in the classifier are shown in table 2.

Table 2. Results of experiments

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Mean
$P_{1,1}$, %	90,74	84,29	73,66	86,30	94,04	94,18	96,60	89,44
$P_{1,2}$, %	75,93	63,57	60,22	70,32	68,77	67,60	69,36	67,71
$P_{2,1}$, %	85,80	83,57	72,58	85,39	98,88	93,92	96,60	88,42
$P_{2,2}$, %	74,69	63,57	59,14	70,32	67,37	66,58	67,66	66,80
$P_{3,1}$, %	92,11	84,21	71,93	89,47	90,53	88,42	93,68	87,72
$P_{3,2}$, %	76,32	57,89	63,16	64,91	71,58	54,74	75,79	65,89
$P_{4,1}$,%	7,89	14,47	28,07	10,53	7,37	11,58	6,32	11,70
$P_{4,2}$,%	21,05	40,79	36,84	31,58	24,21	44,21	22,11	31,97
$P_{5,1}$,%	0	1,32	0	0	2,11	0	0	0,49
$P_{5,2}$,%	2,63	1,32	0	3,51	4,21	1,05	2,11	2,14

As a result of the experiment correctly clasterized 66% of test samples for the classifier without the entered vectors of artificially generated botnets and 88% for the classifier to which the vectors were previously added by performing its training. The percentage of features that the distributed system used to detect botnets and have related to manifestations of software implants, is approximately 27% of the overall detected. The intensity of manifestations of software implants is significantly lower than typical manifestations of botnets. Thus, software implants that use undocumented software features within botnets can be detected by distributed systems [43] and the direction of such research is promising.

5 Discussion and Future work

Software implants that use undocumented software features used on local area networks can be developed and used in various malicious models. Important task is further developing formal profiles and its behavioral signatures. The combination of these components will allow you to get metrics to investigate the presence of this type of malware.

6 Conclusion

Software implants that use undocumented software features used on local area networks can cause significant harm to users of personal computers, especially to businesses that use computer networks and use specialized software.

The obtained class divisions according to the developed solution will allow to perform further analysis of deviations for anomalies search in the classes. The use of developed models of Software implants that use undocumented software features in distributed detection systems [43] has made it possible to improve the detection efficiency of the botnets they were part of.

The direction of further research is the specification and definition of the many functions that will form elements of Software implants that use undocumented software features, with the aim of representation of theirs behavioral signatures to improve detection efficiency.

References

1. Sanjam, G., Gentr, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Hiding Secrets in Software: A Cryptographic Approach to Program Obfuscation. Communications of the ACM, Vol. 59, No. 5, pp. 113-120 (2016).
2. McAfee Mobile Threat Report Q1, 2019. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf
3. DSTU 3396.2-97 Protection of information. Technical protection of information. Terms and definitions. State Committee of Ukraine, Kyiv (1997) [in Ukrainian]
4. Lysenko, S., Savenko, O., Kryshchuk, A., Kljots, Y. Botnet detection technique for corporate area network. In: Proc. of the 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems, pp. 363-368 (2013).
5. Savenko, O., Lysenko, S., Nicheporuk, A., Savenko, B.: Metamorphic Viruses' Detection Technique Based on the Equivalent Functional Block Search. CEUR Workshop, Vol. 1844, pp. 555–569 (2017).
6. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B. et al: Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. CEUR Workshop, Vol. 2301, (2019)
7. Adups Backdoor. Available: https://www.kryptowire.com/adups_security_analysis.html.

8. Alminshid K., Omar, M. N.: Detecting backdoor using stepping stone detection approach. In: Proc. of 2013 Second International Conference on Informatics & Applications (ICIA), Lodz, Poland, pp. 87-92 (2013)
9. Zaddach, J., Kurmus, A., Balzarotti, D., Blass, E.-O., Francillon, A., et al.: Implementation and Implications of a Stealth Hard-drive Backdoor. In. Proc. of 29th Annual Computer Security Applications Conference, New Orleans, Louisiana, US (2013).
10. Dullien, T. F.: Weird machines, exploitability, and provable unexploitability. IEEE Transactions on Emerging Topics in Computing, No. 99, pp. 1-15 (2017)
11. Thomas, S. L., Chothia, T., Garcia, F. D.: HumIDIFy: A Tool for Hidden Functionality Detection in Firmware. In: Proc. of 14th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Bonn, Germany, pp. 279-300 (2017).
12. Thomas, S. L., Chothia, T., Garcia, F. D.: Measuring the Importance of Static Data Comparisons to Detect Backdoors and Undocumented Functionality. In: Proc. of 22nd European Symposium on Research in Computer Security. Oslo, Norway, pp. 513-531 (2017).
13. Schönegge A.: The Hidden Function Question Revisited. In: Proc. of Algebraic Methodology and Software Technology: 6th International Conference, AMAST '97. Sydney, Australia, pp. 451-464 (1997).
14. The Secret Code of Software Validation….In 5 Easy Steps. Available: https://www.cebos.com/blog/the-secret-code-of-software-validation-in-5-easy-steps/
15. Kawamoto, Y., Yamamoto, H.: Secret function sharing schernes and their applications to the oblivious transfer. In: IEEE International Symposium on Information Theory, pp. 281-295, Yokohama, Japan (2003).
16. Pomorova, O., Savenko, O., Lysenko, S., Kryshchuk, A., Bobrovnikova, K.: A Technique for the Botnet Detection Based on DNS-Traffic Analysis. Communications in Computer and Information Science, Vol. 522, pp.127-138 (2015).
17. Pomorova, O., Savenko, O., Lysenko, S., Kryshchuk, A., Bobrovnikova, K.: Anti-evasion Technique for the Botnets Detection Based on the Passive DNS Monitoring and Active DNS Probing. Communications in Computer and Information Science, Vol. 608, pp.83-95 (2016).
18. Savenko, O., Lysenko, S., Nicheporuk, A., Savenko, B.: Approach for the Unknown Metamorphic Virus Detection. In: 9-th IEEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems. Technology and Applications, pp. 453–458 (2017).
19. Desai, P., Stamp, M.: A highly metamorphic virus generator. Int. J. Multimedia Intelligence and Security, Vol. 1(4), pp. 402-427 (2010)
20. Podlovchenko, R.I., Kuzyurin, N.N., Shcherbina V.S., Zakharov V.A.: Using algebraic models of programs for detecting metamorphic malwares. Journal of Mathematical Sciences, Vol. 172 (5), pp. 740-750 (2011)
21. Anderson, B., Quist, D., Neil, J., Storlie C., Lane, T.: Graph-based malware detection using dynamic analysis. Journal in Computer Virology, 7, pp. 247-258 (2011)
22. Runwal, N., Low, R.M., Stamp, M.: Opcode Graph Similarity and Metamorphic Detection. Journal in Computer Virology, 8, pp. 37-52 (2012)
23. Nagaraju, A.: Metamorphic malware detection using base malware identification approach. Journal Security and Communication Networks, 7, pp. 1719-1733 (2014)
24. Patel, M.: Similarity tests for metamorphic virus detection. Master's thesis, San Jose State University (2011)
25. Wong, W.: Analysis and Detection of Metamorphic Computer Viruses. Master's thesis, San Jose State University (2006)

26. Kuriakose, J., Vinod, P.: Unknown Metamorphic Malware Detection: Modelling with Fewer Relevant Features and Robust Feature Selection Techniques, IAENG International Journal of Computer Science, Vol. 42(2), p139-151 (2015)
27. Pomorova, O., Savenko, O., Lysenko, S., Kryshchuk, A. and Nicheporuk, A.: A technique for detection of bots which are using polymorphic code. In: 21st International Conference, CN, Springer, Brunów, Poland, pp. 265-276 (2014)
28. Tarhio, J., Ukkonen, E.: Approximate BoyerMoore String Matching. SIAM Journal on Computing, Vol. 22, No. 2, pp. 243-260 (1993)
29. Drozd, J., Drozd, A., Antoshchuk, S., Kharchenko, V.: Natural Development of the Resources in Design and Testing of the Computer Systems and their Components. In: 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, pp. 233-237. Berlin, Germany (2013)
30. Kondratenko, Y., Kondratenko, N.: Soft Computing Analytic Models for Increasing Efficiency of Fuzzy Information Processing in Decision Support Systems. Chapter in book: Decision Making: Processes, Behavioral Influences and Role in Business Management, R. Hudson (Ed.), Nova Science Publishers, New York, 41-78 (2015)
31. Proskurin, V.: Software malicious implant in secure systems. Available: http://www.crime-research.ru/library/progwir98.htm [in Ukrainian].
32. Kaarin, O. V.: Program protection theory and practice. MGUL, pp. 450 (2004) [in Russian].
33. Kaarin, O. V. Computer system software security. MGUL, pp. 212 (2003) [in Russian].
34. Shanugin, V. F. Protection of computer information. Effective methods and tools: a textbook. DMK Press, pp.544 (2008) [in Russian].
35. Shanugin, V. F. Protection of information in computer systems and networks. DMK Press, pp. 592 (2012) [in Russian].
36. Balakrishnan, G., Reps, T.: WYSINWYX: What You See Is Not What You eXecute. In: Proc of ACM Transactions on Programming Languages and Systems, Vol. 32, Issue 6, (2010).
37. Igumnov, B. N., Zavgorondnyaya, T. P. Cybernetic basis of construction of economic systems of enterprises. Khmelnitsky, TUP, pp. 344 (2000) [in Russian].
38. Palyata, V. B. Comparative multidimensional analysis in economic research. Statistica, pp. 151 (1980) [in Russian].
39. Shatalkin, A.I. Taxonomy: Grounds, principles and rules. Tovarischestvo nauchnyih izdaniy KMK, pp. 600 (2012) [in Russian].
40. Zhambyu, M. Hierarchical cluster analysis and matching. Finance and statistics, pp. 345 (1988) [in Russian].
41. Ward, J.H.: Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association, Vol. 58, No. 301, pp. 236-244 (1963).
42. Savenko, O., Nicheporuk, A., Hurman I., Lysenko, S.: Dynamic Signature-based Malware Detection Technique Based on API Call Tracing. CEUR Workshop, Vol. 2393, pp. 633–643 (2019).
43. Savenko, O., Lysenko, S., Kryschuk, A.: Multi-agent based approach of botnet detection in computer systems. Communications in Computer and Information Science, Vol. 291, pp.171-180 (2012).
44. Balyk, A., Karpinski, M., Naglik, A., Shangytbayeva, G., Romanets, I.: Using graphic network simulator 3 for DDoS attacks simulation. International Journal of Computing. Vol. 16, Issue 4, pp. 219-225 (2017).