Methods of information systems protection

УДК 651.34 **doi: 10.20998/2522-9052.2018.2.16**

I. Mamusiĉ [1], D. Lysytsia [2], A. Lysytsia [2]

[1] University of Zagreb, Zagreb, Croatia
[2]National Technical University "Kharkiv Polytechnic Institute", Kharkiv

MODEL OF DATA PREPARATION FOR ALLOCATION OF ALGORITHM FROM BINARY CODE FOR THE SAFETY ANALYSIS OF THE SOFTWARE

The **subject** of the study in the article is the use of technology of recovery of the algorithm for the allocation of binary attractors in the machine-independent form for the safety analysis of the software. **The purpose** is the first stage of the method of allocation of the algorithm from the binary code with the use of additional attractors - preparatory, which includes the task of allocating a set of attractors with simillar features and synthesis of information about the studied system. The following **results** are obtained. During the cource of the research the analysis of specialized simulators was performed. Such simulators allow to solve the problems of allocation (removal) of some algorithms from binary code. It was determined that additional attractors of the binary code of the program are required in order to increase he accuracy of software security testing. The general structure of the algorithm extraction from binary code is presented. A set of algorithms were developed. **Conclusions**. When combined they create the model of the first stage of data allocation of the algorithm from binary code for the analysis of software security. The key feature of development of this stage is the possibility of constructing a graph for arbitrary attractors, without restriction of the static nature of the code. This will allow a significant expansion of the spectrum of the program code under investigation, including codes with signs of a dynamic change. The further development of this research is to study the whole scheme and develop an appropriate method for allocating a binary code algorithm for software security analysis.

Keywords: software security testing; binary attractor; ethical hacking.

Introduction

Formulation of the problem. An analysis of recent world-wide events related to information security has shown that virtually every modern IT structure has certain vulnerabilities to cyberattacks. At the same time, there is a certain tendency to increase cyberattacks that have succeeded in their malicious purpose. In the opinion of the authors, this is largely due to the lack of attention to the testing of software security (SOA), as well as the discrepancy in opinions of software developers of the very essence of the term and functions of software security testing.

The analysis of popular information sources on the Internet [1, 2, 4, 6] showed that most authors connect the issue of software security testing to the purpose of finding and neutralizing existing risks which present a clear threat to the quality functioning of computer or computerized systems of various purposes.

It is stated that the basic principles of software security are confidentiality, integrity and accessibility [5].

Without diminishing the importance of these principles and without limiting the main strategic goal of software security testing indicated in these sources, it should be noted that some software development organizations, when testing security, focus only on known external factors and simulate various situations that use, for example, the same methods of hacking [2,4]:

– attempts to find out the password using external means;

– attacking the system using special tools that analyze software protection;

– suppression, overloading of the system (with the assumption that it will refuse to serve other clients);

– purposeful introduction of errors in the hope of penetrating the system during the recovery;

– reviewing and analyzing unclassified data in hopes of finding a key for logging into the system.

But at the same time, due to some objective and subjective reasons, testers often ignore the wide possibilities of reverse engineering technology, unfortunately. At the same time, in opinion the authors, some of these technologies can significantly improve the quality of software security testing, reduce the risk of successful cyberattacks, and generally improve the information security.

One of such technologies is the technology of recovery of the algorithm for the allocation of binary attractors in the machine-independent form [3, 9]. This technology helps to solve complex issues of search for non-declared features of the software (mostly malicious ones), as well as errors in implementation and detects the malicious code (computer virus), etc.

The analysis of literature [7, 8, 10] showed that at present times there are a number of specialized simulators which allow solving the issues of selection (removal) of some algorithms from binary code**.** But these programs mostly reaserch only that part of the program that is used during the launche of the analysis process and leave certain "traces" - attractors. An increase in the volume of the researched code is possible with the use of additional code execution attractors, which combine more application execution scenarios.

This determines the actuality of developing a method for allocating an algorithm from a binary code using additional attractors for software security analysis. The overall structure of the allocation of the algorithm from the binary code is schematically presented in fig. 1.

Studies conducted [3, 6] have shown that when analyzing simple programs it is often enough to apply the procedure once. In complex cases, it should be applied iteratively.

As can be seen on the picture, the restoration of the algorithm begins with the preparation of the initial data and the allocation of a set of attractors with simillar features. Then follows a synthesis of information about the researched system from source attractors using the graph approach of representation in the system. After that, the code is exported to a machine-independent representation on the basis of the part of the code that relates to the investigated algorithm. During this operation, optimization solutions are used to simplify the received presentation and, finally, the result is presented in a form suitable for viewing by the analyst and using in the decision support system which uses artificial intelligence. During this moment, the analyst takes a decision whenever the next iteration of the analysis sould be performed.

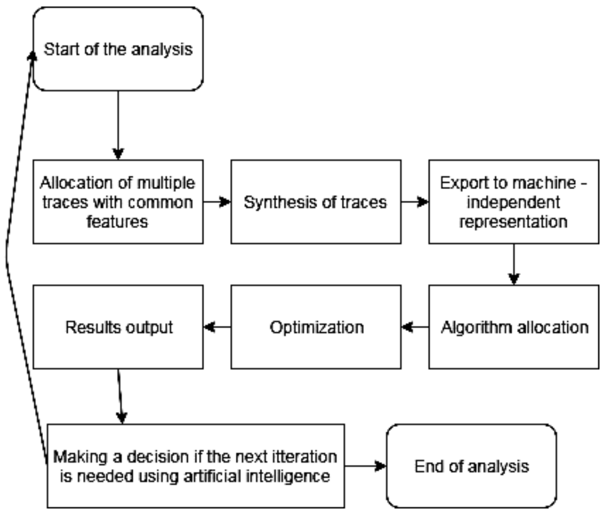
Start of the analysis
Allocation of multiple traces with common features
Synthesis of traces
Export to machine - independent representation
Algorithm allocation
Optimization
Results output
Making a decision if the next itteration is needed using artificial intelligence
End of analysis

Fig. 1. Scheme of algorithm allocation

In the article we will review the first stage of the method of allocation of the algorithm from the binary code with the use of additional attractors - preparatory, which includes the task of allocating a set of attractors with simillar features and synthesis of information about the studied system.

1. Selection of attractors set with simillar features

A binary attractor of execution, obtained using an attractor simulator (hereinafter - just attractor), represents a sequence of steps. Each step contains the code for the executed instruction and the value of some of the main registers before it is executed.

Due to the full-system nature of the formation attractors contain actions regarding all programs active in the system during removal of attractors, including the kernel and other components of the operating system. It is assumed that the attractor steps according to their belonging to different processes and execution threads. In this case, the low-level components of the operating system, such as interrupt handlers, must be matched with separate processes and / or execution threads. The symbols associated with attractors, steps, instructions, and auxiliary data are summed-up in the table 1.

It should also be noted that sets *read* $[t^{(i)}]$ and *write* $[t^{(i)}]$ are associated with attractor's step, not the instruction, i.e. they may differ for different values of operands. Size attributes and attributes of belonging to a class of control instructions, on the contrary, are constant for a given instruction.

In addition to the set of attractors, which describes some of the scenarios of the behavior of the researched program, for analysis, the information on how these attractors correlate with each other is needed. In the framework of the proposed method, the relations are given using the following definitions:

Table 1: **Key notation for the attractor**

Notation	Description
$len[t]$; $t^{(i)}$	Number of steps in the attractor t; step to the attractor t with the number $i \in \{\overline{1, len[t]}\}$
$addr\left[t^{(i)}\right]$; $inst\left[t^{(i)}\right]$	Address that was executed on the step $t^{(i)}$ of instruction; instruction which was executed on the step $t^{(i)}$
$process\left[t^{(i)}\right]$ $thread\left[t^{(i)}\right]$	Process ID on step $t^{(i)}$; Run Flow Id on step $t^{(i)}$
$read\left[t^{(i)}\right]$; $write\left[t^{(i)}\right]$	A set of memory addresses that are read on step $t^{(i)}$; a set of memory addresses written on step $t^{(i)}$
$size[j]$; $branch[j]$	The size of the bytes of the instruction code j; Idnetifier of whenever instruction j is the control transmission.

Under the related attractors we mean the set of attractors obtained from the same initial state of the system (i.e., from the same image of the state of the simulator).

Such attractors will differ by scenarios they have been implemented in the analyzed system, which, in turn, is determined by the input data. In the case of interactive applications, the input data can be considered a sequence of actions in the graphical interface.

The basic initial data for the proposed algorithm restoration procedure, thus, will be a set of related attractors.

2. Synthesis of related attractors

The first step in the procedure for restoring the algorithm is to combine a set of related attractors into a general $G = (V, C)$ representation-oriented graph with loops, corresponding to the set of interprocedural graphs of the flow of control of individual execution streams with additional marks. In the general, when attractors include multiple threads of execution, this graph will be unconnected, and each flow of execution will match its component of connectivity.

In the graph G, as in the usual control flow graph, the vertices V correspond to the base blocks (the linear sections of the code of this execution thread), and the edges C are possible control transmissions between these parts. Since in the proposed approach the only source of knowledge about the control flow of the program is its attractors, the graph will include only those edges whose transitions were actually observed.

In addition to the usual vertices of the base blocks, for each flow of execution in the graph G are defined those which are guaranteed not to contain any instructions incoming and outgoing vertices, the first of which is dominant, and the second post-dominate over all other base blocks of this execution flow.

At each vertex are stored: initial address, sequence of instructions and generation number. Generation number is an integer that represents the state of the code of the program and allows you to correctly represent the code that is changed during execution. Within each separate flow of execution, the generation number will increase by one when rewriting the code of the given stream. The symbols to be used further are given in table 2.

Table 2. **Key notation**

Notation	Description
$entry[G]_T$; $exit[G]_T$	The input vertex of the flow of execution T in the graph G; output vertex of the flow of execution T in the graph G
$start[B]$; $end[B]$; $insn[B]$; $gen[B]$	Primary address of the base unit B ; the address of the end of the base unit B , not enabled; sequence of instructions in the base unit B ; generation number of the base unit B
$succ[B]$; $pred[B]$	The set of base blocks in which the edges of B ; the set of base blocks from which the edges B start
$from[e]$; $to[e]$	The basic block from which the edge e starts; base unit, to which an edge e leads

he address of the end of the base unit $end[B]$ is calculated on the basis of $start[B]$ and $insn[B]$:

$$end[B] = start[B] + \sum_{j \in insn[B]} size[j].$$

In addition to the symbols from the table 2, in the pseudocode of the following algorithms the following functions will be considered available, the implementation of which depends on the selected method of storing the graph.

1. F1 function creates and returns a new empty graph.
2. The function F2 (G, T, n, a) creates a new base unit in the graph G belonging to a generation with the number n of the runtime T, and assigns it the initial address a. The list of instructions for the newly created base unit is initially set to blank. The address a may have a special meaning, that corresponds not to the addresses of the start and end vertices.
3. Function F3 (G, B) removes the base unit B from graph G along with all edges adjacent to it.
4. The function F4 (G, T, n) returns the ordered list of base generators with the number n of the flow of T in the graph G.
5. The function F5 (G, T, n, a) finds the base unit B in the graph G, which belongs to a generation with the number n of the runtime T, such that it belongs to the address a: . If no such block is found, a special value is returned.
6. The function F6 (G, B, a) produces the division of the base unit B into two so that part of the instructions B, whose addresses are smaller than the address a, fall into the first block, and the rest - in the second. The address a must belong to the base block B. The function returns a pair of received blocks, where the addresse b is less than a, and b - more or equal.
7. Function F7 (G, B, B ') connects the base blocks B and B' in the graph G by the edge. If such an edge already exists, then the new one is not added. The function F8 (G, T, n) returns an unregulated set of edges connecting base generating units with the number n of the flow of T in graph GIf only one of the incident edges of the base blocks belongs to a given generation, and the second one is not, then such an edge is not included in the set of returned.

If this set is empty, then a graph is empy as well. The graph has the following properties.

1. Each component of the connectivity describes the flow of control of one flow of execution.
2. Each edge connects either one-generation base units or from a base unit with a lower generation number to a base unit with a larger.
3. Within a single generation, one flow of execution, the base units do not intersect at the addresses. When a control is found in a given generation, there is no overwrite of executable code. In aggregate, this means that within a single generation of one stream of execution, the method of static analysis is applied without change
4. The transformation of graph G, squeezing within each component of the connection all the vertices with the same generation number into one vertex, allows us to obtain an acyclic graph describing the code modification episodes in each flow of execution. Such a graph will be called an evolution graph. The type of evolution graph allows you to get an idea of the nature of the code transformations carried out in the system of the language being studied. The number of vertices and / or edges in it can be considered one of the metrics of complexity of the system.

Further, we describe the algorithms that implement the initial construction of the representation G on the first iteration of the procedure for the restoration of the algorithm and its replenishment in subsequent iterations.

Firstly, let's review the simplified situation, when in the processed attractor there is no modification of the code in the process of execution. This situation is possible in practice, when the attractor presents the work of the main part, mostly unprotected by the mechanisms of self-modification of the program: loading the program image and dynamic libraries has already been carried out and carried out the binding of all the functions used.

The pseudo code for the "static graph restoration" algorithm is shown on fig. 2. The algorithm receives at the input of the attractor t and issues the graph G for it at the output. For each execution stream, the method called "continue the static reproduction of the graph" is called (fig. 3).

```
function Static graph construction ( $t$ )
    $G \leftarrow$ F1
    for $T \in \bigcup_{i=1}^{len[t]} \{thread[t^{(i)}]\}$ do
        $entry[G]_T \leftarrow$ F2 ( $G$, $T$ , 1, 0)
        $exit[G]_T \leftarrow$ F2 ( $G$, $T$ , 1, 0)
        $E \leftarrow$ Static graph construction funiculus ( $t$ )
        $G \leftarrow$ F1
        ( $G$ , $entry[G]_T$ , $t$ , 1, $len[t]$ , $T$ , 1)
        F7 ( $G, E, exit[G]_T$ )
    end for
    return $G$
end function
```

Fig. 2. Pseudo-code for the algorithm of "static graph creation"

```
function Static graph construction funiculus ( $G$ , $S$ , $t$ , $a$ , $b$ , $T$ , $n$ )
    $E \leftarrow S$ ; $m \leftarrow 0$
    for $i = a, a+1, ..., b$ do
        if $thread[t^{(i)}] = T$ then
            if $m = 0$ then
                $B \leftarrow$ F5 $(G, T, n, addr[t^{(i)}])$
                if $B = 0$ then
                    $B \leftarrow$ F2 $(G, T, n, addr[t^{(i)}])$
                    $i \leftarrow i - 1$
                else if $start[B] \neq addr[t^{(i)}]$ then
                    $\langle B', B \rangle \leftarrow$ F6 ( $G$ , $B$ , $addr[t^{(i)}]$)
                end if F7 ( $G$ , $T$ $E$ , $B$ )
                $E \leftarrow B$; $m \leftarrow |insn[B]| - 1$
            else if $m > 0$ then
                $m \leftarrow m - 1$
            else if $m < 0$ then
                $B \leftarrow$ F5 ( $G$ , $T$ , $n$ , $addr[t^{(i)}]$)
                if $B = 0$ then
                    $insn[E] \leftarrow insn[E] \cup insn[t^{(i)}]$
                    if $branch[insn[t^{(i)}]]$ then
                        $m \leftarrow 0$
                    end if
                else
                    F7 ( $G$ , $E$ , $B$ )
                    $E \leftarrow B$; $m \leftarrow |insn[B]| - 1$
                end if
            end if
        end if
    end for
    if $(E \neq S) \wedge (addr[t^{(b)}] + size[insn[t^{(b)}]] \neq end[E])$ then
        $\langle E, E' \rangle \leftarrow$ F6 ( $G$ , $E$ , $addr[t^{(b)}] + size[insn[t^{(b)}]]$)
    end if
    return $E$
end function
```

Fig. 3. Pseudo-code for the algorithm "Continuation of static graph restoration"

Its parameters are as follows: G is the graph to be built, S is the start vertex, t is the attractor, [a, b] is the range of attractor steps considered, T is the execution flow identifier, n is the generation number assigned to the created base blocks.

The value of most of these parameters during the call on the "static graph" algorithm is fixed, but they will start to change with the addition of dynamic code support in the algorithm. The algorithm returns the base unit in which check was performed last.

The algorithm implements a successive passage through the steps of the attractor belonging to the flow of execution T. At the same time it tracks, in which base block the execution is performed. The variable m describes the state of the algorithm: for m = 0, in the previous step under consideration, a control transfer took place, or this step was first revised; When m > 0 the control is located inside a known base unit, it remains to see m sequential instructions before it is finished; When m < 0 the control is located inside the base unit, it has never been seen before.

When considering the transfer of control from the current block E, the following three situations are possible:

1. The control is transmitted to the address of the beginning of a known base unit B. If it does not already exists, the edge corresponds to this transition.

2. The control is transmitted to the address in a known base unit B, but not at the beginning. A division of the base unit B is carried out for this address, after which an edge E is added in the second set of the base units received.

3. The control is transmitted to an address that does not belong to any known base unit. Then a new base unit B with this address is created as the start address, an edge is added from E to B, and the algorithm switches to the new block view mode $(m < 0)$.

When you look at the instructions of the newly created base unit in situation 3, each subsequent instruction is added to its insn list. This procedure continues until one of the following conditions for the completion of the base unit is completed.

1. The last instruction of the flow of T in the range of steps is revised $[a, b]$.

2. Revised management transfer instruction. By definition of the base unit, this is its last instruction. The algorithm returns to the control transfer control mode $(m = 0)$.

The address of the next instruction corresponds to the beginning of the known base unit B. An appropriate edge is added, and the algorithm goes into the skip mode of the known base unit .

Finally, when all the instructions for the flow of T in the range of steps are revised $[a, b]$, if necessary, separation of the last considered base unit E is carried out. This need arises if the instruction of the last of the revised steps to the attractor is not the last instruction in *insn*[*E*]. Separation is carried out at the end of this instruction.

Then the first of the blocks obtained after the separation of the base units can be marked as finite, since the flow of control at reaching its end may stop. In addition, this property is used in the next section when adding support for the dynamic code.

3. Dynamic code

The presence of dynamically changing code can be caused by various reasons. Here are a few options, from the most common and unrelated to the intentional counteraction to the analysis, to the purposeful (creating difficulties), especially for static analysis.

Usage of dynamic editor for linking uploads and outloads in dynamic libraries. The address range of the newly downloaded library can cross the address range that was already uploaded. Thus at different periods of time there will be a different code in the same memory addresses.

Use of "springboards" and delayed bindings. When you first go to the address of dynamically linked function control can be transferred to a subroutine that performs deferred binding. Once the binding is performed, the routine will correct the "springboard" in such a way that during the subsequent calls the function is called directly and will give control to it for the first time.

The program contains mechanisms of decryption, decompression, dynamic code generation, which overwrite or do not require and reject fragments of the program; or if such mechanisms work with the program in parts, then use one rewritable buffer for the next decrypted, unpacked or generated part of the program.

Polymorphic nature of the program or its parts. This case differs from the previous one, because most often the next version of the program code is based on the previous, which further complicates the analysis. Most often, such mechanism is embedded in malicious code, especially in viruses, in order to prevent signature analysis in antivirus software.

The method proposed in this article does not distinguish the causes of the dynamic change of the code. All possibilities are treated in the same way, which on the one hand has the advantages of the universality of the method, but on the other hand ignores additional "hints" about the behavior of the program contained in these reasons. However, a certain idea of the nature of the dynamic code in this program gives the evolutionary graph described above.

Regardless of the content that falls into an episode of dynamic program code changes, it passes through one of the following two scenarios.

The program executed on this system records the values in memory. This memory may belong to this program, as well as any other (for this operating system should support the ability to connect the flow of one process to the address space of another). Recorded values either change the code which is already executed, or generate code that will be executed in the future. One way or another, the entry is made to addresses that also appear in the attractor as being executed.

One or more pages of physical memory change as a result of executing a DMA transaction by any computer device (often a hard disk controller). In this case, the attractor does not observe the fact of direct recording, since such transactions are initiated by the device itself, and they do not meet any instructions. Recorded addresses, as in the first scenario, have either been used previously or will be used later as executable.

Summarizing all of the foregoing, one can formulate the criterion for the presence of a dynamic code in the part of the attractor. For example, attractor t has the range of step numbers and any process is recorded P, and in the given segment there are steps, belonging to P. Let's mark the set of these steps through . We construct the following two sets:

$$\Re(R_P) = \bigcup_{r \in R_P} read[t^r]; \tag{1}$$

$$W(R_P) = \bigcup_{r \in R_P} write[t^r]; \tag{2}$$

$$X(R_P) = \bigcup_{r \in R_P} [addr[t^r], addr[t^r] + size[insn[t^r]] - 1]. \tag{3}$$

As we can see the set contains all the addresses by which the instruction was recorded in the steps , while set - All addresses from which you selected in steps instructions for execution. Let's assume that a known set of virtual memory addresses is overwritten in a process as a result of DMA transactions in the range R. We note this set through.

In the following notation, the set will not contain the dynamic of the code if and only if completed

$$(\Re(R_P) \cup W(R_P) \cup D_P(R)) \cap X(R_P) = 0 . \tag{4}$$

In this case, the code relating to the process in the range of steps is static and can be analyzed by methods of static analysis. We also apply the algorithm of "static graph creation" to the steps. The above reaserch allow us to construct the graph "CTG" for for an arbitrary attractor, without limitations of the static code. The full algorithm "CTG-Full" will consist of the following two steps.

1. Split the attractor into segments of static code in each process. At this stage, we will use a greedy algorithm: we will build segments from smaller step numbers in the attractor to larger ones, and each increment will be expanded until 3 are executed.
2. Application of the algorithm "Continuation of static graph restoration" in each received interval of static. At the same time, with each regular segment, the next generation number will be confirmed within this flow of execution.

The pseudo-code for the "CTG-Full" algorithm is shown on fig. 4, and the algorithm for splitting the attractor into segments of the static code "CTG-Full-Partition" on fig. 5. The CTG-Full algorithm receives attractor t on input and outputs a graph G built for it The "CTG-Full-Partition" algorithm returns an ordered sequence of static segments S long the attractor t and identifier of the process P. The segments in the sequence do not intersect, and their association corresponds to the entire range of steps in the attractor t, which means S sets the breaker to the attractor t.

In some doubtful situations, the algorithm built may require additional refinements.

Such situations are possible in programs provided with certain types of hinged protection, as well as in the operating system code. Therefore, in the future there is a need for the practical adaptation of the developed algorithms to possible casuist deviations in the programs.

```
1: function CTG-Full ($t$)
2:  $G \leftarrow$ F7
3: for $T \in \bigcup_{i=1}^{len[t]} \{thread[t^i]\}$ do
4:  $entry[G]_T \leftarrow$ F2 ($G, T, 1, 0$)
5:  $n_T \leftarrow 1$; $E_T \leftarrow entry[G]_T$;
6: end for
7: for $P \in \bigcup_{i=1}^{len[t]} \{process[t^i]\}$ do
8: for $\langle a,b \rangle \in$ CTG-Full-Розбиття $(t,P)$ do
9: for $T \in \bigcup_{i=a}^{b} \{thread[t^i]\}$ do
10: $E_T \leftarrow$ Static graph construction funiculus ($G, E_T\ t, a, b, T, n_T$)
11: $n_T \leftarrow n_T + 1$
12: end for
13: end for
14: end for
15: for $T \in \bigcup_{i=1}^{len[t]} \{thread[t^i]\}$
16: $exit[G]_T \leftarrow$ F2 ($G, T, n_T, 0$)
17: F7 ($G, E_T, exit[G]_T$)
18: end for
19: return $G$
20: end function
```

Fig. 4. "CTG-Full" algorithm

Conclusion

Thus, a set of algorithms for allocating a set of attractors with related features and synthesizing information about the investigated system was developed, which is the first stage of the method of allocation of the algorithm from the binary code with the use of additional attractors.

A distinctive feature of development of this phase is the possibility of constructing a graph for an arbitrary attractor, without limiting the static nature of the code. This will allow a significant expansion of the spectrum of the program code under investigation, including codes with signs of a dynamic change.

```
1: function «CTG-Full-Розбиття» ($t,P$)
2: $W, X \leftarrow 0$; $S \leftarrow \{\ \}$; $a \leftarrow 1$
3: for $i \in 1,2,\ldots,len[t]$ do
4: if $process[t^{(i)}] = P$ then
5:  $\omega \leftarrow read[t^{(i)}] \cup \Re_P(\{i\})$
6:  $\omega 1 \leftarrow write[\omega] \cup D_P(\{i\})$
7:  $x \leftarrow [addr[t^i], addr[t^i] + size[insn[t^i]] - 1]$
8: if $(W \cap x = 0) \wedge (X \cap \omega = 0)$ then
9:  $W \leftarrow W \cup \omega$
10: else
11: $W \leftarrow \omega 1$;
12: $S \leftarrow S \cup \langle a, i-1 \rangle; a \leftarrow i$
13: end if
14: end if
15: end for
16: return $S \cup \langle a, len[t] \rangle$
17: end function
```

Fig. 5. Algorithm for splitting the attractor into segments of the static code "CTG-Full-Partition"

The further development of this research is to study the whole scheme and develop an appropriate method for allocating a binary code algorithm for software security analysis.

СПИСОК ЛІТЕРАТУРИ

1. Абушинов О. Особенности тестирования безопасности ПО [Электронный ресурс] / О. Абушинов. – Режим доступа: https://testitquickly.com/2010/11/20/22.
2. Дорофеев А. Тестирование на проникновение: демонстрация одной уязвимости или объективная оценка защищенности? [Электронный ресурс] / А. Дорофеев. – Режим доступа: https://www.npo-echelon.ru/doc/inside-dorofeev.pdf.
3. Жаркова А. В. Об аттракторах в конечных динамических системах двоичных векторов, ассоциированных с ориентациями пальм / А. В. Жаркова // Прикладная дискретная математика. – Томск, 2014. – №7. – С. 58-67.
4. Ильюк Д. Тестирование безопасности – выбираем нужное [Электронный ресурс] / Д. Ильюк. – Режим доступа: http://software-testing.ru/library/testing/security/1986-security-testing.
5. Кузнецов О. О. Протоколи захисту інформації у комп’ютерних системах та мережах / О. О. Кузнецов, С. Г. Семенов. – Х.: ХНУРЕ, 2009. – 184 с.
6. Семенов С. Г. Комплекс математичних моделей процессу розробки програмного забезпечення / Інформаційні технології та комп’ютерна інженерія / С. Г. Семенов, Кассем Халіфе. – Вінниця : ВНТУ, 2017. – Вип. 3(40). – С. 61-68.
7. AMD SimNow Simulator [Электронный ресурс]. – Режим доступа: http://developer.amd.com/cpu/simnow/Pages/default.aspx.
8. IDA Pro – at the cornerstone of IT security [Электронный ресурс]. – Режим доступа: https://www.hex-rays.com/products/ida/ida-executive.pdf.
9. Ivancevic V. G. High-Dimensional Chaotic and Attractor Systems: A Comprehensive Introduction / V. G. Ivancevic, T. T. Ivancevic. – Springer Science & Business Media, 2007. – 697 p.
10. Magnusson P. S. A Full System Simulation Platform / P. S. Magnusson // IEEE Computer, 35(2), Feb. 2002, P. 50-58, available at : https://doi.org/10.1109/2.982916.
11. Robert C. Seacord Secure Coding in C and C++ / C. Robert. – The SEI Series in Software Engineering, 2013. – 569 p.

REFERENCES

1. Abushinov, O. (2017), *Features of software security testing*, available at: https://testitquickly.com/2010/11/20/22 (last accessed March 05, 2018).
2. Dorofeev A. (2017), *Penetration testing: demonstration of one vulnerability or an objective evaluation of security*, available at: https://www.npo-echelon.ru/doc/inside-dorofeev.pdf (last accessed March 05, 2018).
3. Zharkova, A.V. (2014), "On attractors in finite dynamical systems of binary vectors associated with palm orientations", *Applied Discrete Mathematics*, Tomsk, No. 7, pp. 58-67.
4. Iljuk, D. (2017), Safety testing - choose the right one, available at: http://software-testing.ru/library/testing/security/1986-security-testing (last accessed March 05, 2018).
5. Kuznetsov, O.O. and Semenov, S.G. (2009), *Protocols of the information zahistu at computer systems on that level*, KhNURE, Kharkiv, 184 p.
6. Semenov, S.G. and Kassem, Khalifa (2017), "A complex of mathematical models for the process of disassembling software software", *Information technology and computer engineering*, VNTU, Vinnitsa, No. 3 (40), pp. 61-68.
7. *AMD SimNow Simulator* (2016), available at: http://developer.amd.com/cpu/simnow/Pages/default.aspx (last accessed March 05, 2018).
8. *IDA Pro – at the cornerstone of IT security* (2016), available at: https://www.hex-rays.com/products/ida/ida-executive.pdf (last accessed March 05, 2018).
9. Ivancevic, Vladimir G., Ivancevic, Tijana T. (2007), *High-Dimensional Chaotic and Attractor Systems: A Comprehensive Introduction*, Springer Science & Business Media, 697 p.
10. Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A. and Werner, B. (2002), "A Full System Simulation Platform", *IEEE Computer*, No 35 (2), pp. 50–58, available at : https://doi.org/10.1109/2.982916.
11. Robert, C. (2013), *Seacord Secure Coding in C and C++*, The SEI Series in Software Engineering, 569 p.

Надійшла (received) 01.04.2018
Прийнята до друку (accepted for publication) 16.04.2018

Модель підготовки даних виділення алгоритму з двійкового коду для аналізу безпеки програмного забезпечення

І. Мамузіч, Д. О. Лисиця, А. О. Лисиця

Предмет дослідження – використання технології відновлення в машинно-незалежному вигляді алгоритму по набору двійкових атракторів для аналізу безпеки програмного забезпечення. **Мета статті** – розглядання першого етапу методу виділення алгоритму з двійкового коду з використанням додаткових атракторів – підготовчого, що включає в себе завдання виділення множини атракторів с загальними ознаками та синтез інформації про досліджувану систему. Отримані такі **результати.** Проведено аналіз спеціалізованих симуляторів, що дозволяють вирішувати питання виділення (вилучення) деяких алгоритмів з двійкового коду. Визначено необхідність дослідження додаткових атракторів двійкового коду програми для підвищення точності тестування безпеки програмного забезпечення. Схематично запропоновано загальну структуру виділення алгоритму з двійкового коду. **Висновки.** Розроблено комплекс алгоритмів, що в цілому складають модель першого етапу підготовки даних виділення алгоритму з двійкового коду для аналізу безпеки програмного забезпечення. Особливістю розробок цього етапу є можливість побудування графу для довільного атрактору, без обмеження на статичність коду. Це надасть можливість суттєвого розширення спектру досліджуваних програмних кодів, у тому числі кодів, що мають ознаки динамічної зміни. Подальший розвиток роботи полягає у дослідженні всієї схеми та розробки відповідного методу виділення алгоритму з двійкового коду для аналізу безпеки програмного забезпечення.

Ключові слова: тестування безпеки програмного забезпечення; двійковий атрактор; етичний хакінг.

Модель подготовки данных выделения алгоритма из двоичного кода для анализа безопасности программного обеспечения

И. Мамузич, Д. А. Лисица, А.А. Лисица

Предмет исследования – использование технологии восстановления в машинно-независимом виде алгоритма по набору двоичных аттракторов для анализа безопасности программного обеспечения. **Цель статьи** – рассмотрение первого этапа метода выделения алгоритма из двоичного кода с использованием дополнительных аттракторов – подготовительного, который включает в себя задачу выделения множества аттракторов с общими признаками и синтез информации об исследуемой системе. **Получены следующие результаты.** Проведен анализ специализированных симуляторов, позволяющих решать вопросы выделения (изъятия) некоторых алгоритмов из двоичного кода. Определена необходимость изыскания дополнительных аттракторов двоичного кода программы для повышения точности тестирования безопасности программного обеспечения. Схематически предложена общая структура выделения алгоритма из двоичного кода. **Выводы.** Разработан комплекс алгоритмов, которые в целом составляют модель первого этапа подготовки данных выделения алгоритма из двоичного кода для анализа безопасности программного обеспечения. Особенностью разработок этого этапа является возможность построения графа для произвольного аттракторов, без ограничения на статичность кода. Это позволит существенно расширить спектр изучаемых программных кодов, в том числе кодов, имеющих признаки динамического изменения. Дальнейшее развитие работы заключается в исследовании всей схемы и разработке соответствующего метода выделения алгоритма из двоичного кода для анализа безопасности программного обеспечения.

Ключевые слова: тестирование безопасности программного обеспечения; двоичный аттрактор; этический хакинг.