Testing Software and Hardware Data Security Tools Using the Automata Theory and the Graph Theory

1st Tatiana M. Kanner
National Research Nuclear University MEPhI
Moscow, Russia
sheikot@mail.ru

2nd Andrey M. Kanner
National Research Nuclear University MEPhI
Moscow, Russia
kanner@mail.ru

***Abstract*—The article focuses on the application of existing provisions of the automata and graph theories to solving the problem of testing software and hardware data security tools (DST). The software and hardware DST, unlike software ones, include hardware components that implement key security functions, while preventing from using a number of testing methods and tools. In addition to the possibility of applying a particular known testing method or tool to software and hardware DST, what remains acute is the problem of ensuring completeness and optimality of such testing. The developers of various DST do not often have a clear understanding of when they can stop testing and whether the test results allow them to talk about its completeness. Accordingly, testing of DST is often spontaneous, and the developer does not understand whether all the security functions have been tested, whether all the states and all possible sets of parameters have been tested, and whether testing is being carried out in the optimal way. To eliminate these shortcomings, the authors of the article propose to use a mathematical approach based on the theories of automata and graphs to solve the problem of testing software and hardware DST, which can be also used for other software and hardware, as well as software tools and systems. Applying this approach in practice, it is possible to confirm or reject the possibility of ensuring completeness of testing a specific data security tool, as well as identifying specific measures to ensure completeness and optimality of testing.**

***Index Terms*—software and hardware data security tools, access control subsystem, completeness and optimality of testing, finite deterministic automaton, directed graphs, Eulerian path, Chinese Postman Problem**

I. Introduction

Some works [1]–[5] have already considered testing of various software or software-hardware tools, including data security tools, from the point of view of using some mathematical apparatus. In such works, the automata theory is mainly used to describe a particular software or software-hardware complex, while the behavior of the complex is modeled by automata transitions and obtaining output values. So, for example, in the previous work [5], the author developed a descriptive and then a formal model of an arbitrary software and hardware data security tool (DST), and then formed the relevant necessary and sufficient conditions for the testing possibility in principle. However, the results of this work did not provide for a clear method for further testing of software and hardware DST, but only justified the possibility or impossibility of such testing.

The real problem of testing the software and hardware DST is to ensure that all possible security functions are tested in the required set of automaton states corresponding to this security tool, and, if possible, while ensuring the minimum number of actions to be performed. This means that in addition to the possibility of testing, it is necessary to ensure its completeness and optimality. To solve this problem, let us introduce the necessary denotations and definitions.

Let us introduce an arbitrary $m \in M$, where M is a finite set of all software and hardware DST. We will further consider m as a general case of a software and hardware DST, for which we will formulate the corresponding definitions and assertions.

Let us denote $V = \{v_0, v_1, \dots, v_n\}$, $n \in \mathbb{N}$ as a finite set of states in which $m \in M$ can exist (various combinations of states of software and hardware components). In this case $v_0 \in V$ is the initial state, for example, when the software component is not installed and/or a hardware component is not initialized.

Let us denote $I = I_{sf} \cup I_{nf}$ as a set of functions of $m \in M$, which can be performed in states $v_j \in V$, $j = 0, \dots, n$ and are either security functions with formal parameters (I_{sf}) or non-target functions of the software and hardware DST (I_{nf}). $I_{sf} \cap I_{nf} = \emptyset$.

In the general case, for any $m \in M$, the performance of the security function $i \in I_{sf}$ should be verified in one or several states from V. If several parameters are tested for a specific security function, the V set should include the states corresponding to all possible combinations of parameters for the given security function. Thus, testing will consist in traversing the states of the software and hardware DST without the need to revisit the same state to test various parameters of any security function.

In addition, it should be borne in mind that some security functions of the software and hardware DST may be performed only in certain states of the software and/or hardware components of the security tool. Let us denote $v_j \in V$, $j = 0, \dots, n$ as a state with potentially computable security functions, if $\exists i_{sf_1}, \dots, i_{sf_k} \in I_{sf}$, $k \in \mathbb{N}$, which should be tested in state v_j in accordance with V arrangement. Let us denote $V_{sf} \subseteq V$ as a set of all states with potentially computable security functions.

Let us define a fixed abstract software and hardware DST representing it in the form of a finite determinate automaton

$m = (V, v_0, V_{sf}, I, O, T)$, where:

- V, v_0, V_{sf} and I correspond to the previously introduced designations and definitions;
- V is a set of the automaton's states, v_0 is the initial state of the automaton, V_{sf} is a set of all the states with potentially computable security functions;
- I is a set of stimuli (inputs) of the automaton;
- O is a set of reactions (outputs), that is, the results of applying the stimuli in the automaton's states;
- $T \subseteq V \times I \times O \times V$ is a set of the automaton's transitions; at each transition (v, i, o, v'), where $v \neq v'$, from state v in case of applying stimulus i with reaction o, the automaton transits to state v'.

The problem of testing the software and hardware DST is to traverse the automaton's $(V, v_0, V_{sf}, I, O, T)$ states by applying stimuli, where:

- To ensure completeness of testing, it is necessary to transit to each state $v_j \in V_{sf}$, $j \leq n$ using the maximum possible number of stimuli $i \in I$ (with various sets of parameters and input conditions, that is, from various states of the software and hardware DST);
- To ensure optimality of testing, it is necessary to implement the minimum number of the automaton's transitions (v, i, o, v').

We will say that the testing problem has been solved for the software and hardware DST $m = (V, v_0, V_{sf}, I, O, T) \in M$, when completeness and optimality conditions are met simultaneously (hereinafter $T' \subseteq T$ will mean a set of all the transitions made during testing):

$$\forall v' \in V_{sf} : \exists (v, i, o, v') \in T' \tag{1}$$

$$\sum_{\substack{v \in V \\ v' \in V_{sf}}} (v, i, o, v') \rightarrow max, (v, i, o, v') \in T' \tag{2}$$

$$|T'| \rightarrow min \tag{3}$$

II. Materials and Methods

To solve the testing problem and verify (1)–(3), we can use the existing provisions of the graph theory [6]–[9] by presenting an automaton corresponding to a fixed software and hardware DST in the form of a graph [4].

Let us denote $G_m = (V, E)$ as a directed loop-free graph without multiple edges (a simple digraph) corresponding to the software and hardware DST $m \in M$ represented in the form of a finite deterministic automaton $m = (V, v_0, V_{sf}, I, O, T)$, where:

- V is a set of the graph nodes corresponding to the automaton's states;
- $E \subseteq V \times V$ is a set of the graph's directed edges, that is the transitions $(v, i, o, v') \in T$ not taking into account the stimuli and reactions of the automaton.

Such representation of $m \in M$ in the form of a graph does not allow to include some details of the automaton's $m = (V, v_0, V_{sf}, I, O, T)$ operation, namely stimuli and reactions. Let us further assume that for the above-defined graph $G_m = (V, E)$ the necessary and sufficient conditions for testing are fulfilled [5], meaning that the set of edges E contains only those transitions that are computable.

Equations (1) and (2) allow us to conclude that the traversal of graph $G_m = (V, E)$ should be performed mainly by the edges from set $V_{sf} \subseteq V$, and not generally by the whole set of edges V. That is why instead of graph $G_m = (V, E)$ we will consider a derived graph G'_m, which will be constructed by removing some nodes and edges that are not used in solving the testing problem.

Let us denote $G'_m = (V', E')$ as a graph derived from G_m by deleting nodes $v \notin V_{sf} \cup \{v_0\} \subseteq V$ and some edges from E in accordance with the following rules:

- $\forall v', v'' \in V \setminus \{v\}$: $(v, v'') \in E$ and $(v', v) \in E$ – it is necessary to add new edges $(v', v'') : v' \neq v''$ to E and at the end – to delete v from V and all edges (v, v'') and (v', v) from E for which new edges were constructed;
- $\forall v' \in V \setminus \{v\}$: $(v', v) \in E$ and $\nexists v'' \in V \setminus \{v\}$: $(v, v'') \in E$ – it is necessary to delete all the edges (v', v) from E and at the end – to delete v from V;
- If $\nexists v' \in V \setminus \{v\}$: $(v, v') \in E$ or $(v', v) \in E$ – it is necessary to delete v from V;
- $\forall v' \in V \setminus V_{sf}$: $(v, v') \in E$ and $\nexists v'' \in V \setminus \{v\}$: $(v'', v) \in E$ – it is necessary to delete all the edges (v, v') from E, if at the end $\nexists v''' \in V_{sf}$: $(v, v''') \in E$ – delete v from V.

Let us justify that graph $G'_m = (V', E')$ is equivalent to graph $G_m = (V, E)$ in terms of solving the testing problem under (1)–(3). According to our denotation of G'_m, the following nodes and edges are not deleted from the initial graph G_m:

- v_0;
- nodes with potentially computable security features;
- edges to nodes with potentially computable security functions (in the first rule, such edges can be replaced by similar ones without participation of an intermediate deleted node).

In this connection, if (1)–(2) are fulfilled for graph G_m, they will be also fulfilled for graph G'_m (and vice versa), since edges to nodes with potentially computable security functions are not deleted from G_m, and respectively, their sum will not change. Fulfillment of (3) is not prevented, since if the set of transitions for G_m is minimal, it will be minimal for G'_m and vice versa. Thus, the solution to the testing problem in terms of fulfilling (1)–(3) for graph G'_m will be a solution to the testing problem for G_m as well.

Let us consider the testing problem from the point of view of the known provisions of the graph theory. Equations (1)–(3) consist in finding a path passing through all the edges of graph G'_m at least once for a minimum number of transitions. Thus, in accordance with [7], [10], [11], the testing problem is reduced either to the problem of finding the Eulerian path, or to the Chinese Postman Problem, also known as the Route Inspection Problem.

The Eulerian path is the optimal solution to this problem, since each edge is traversed only once. The Eulerian path in graph G'_m exists if and only if [7], [10]:

1) G'_m is strongly connected [6]–[9];
2) there are two nodes $v', v'' \in V'$: their in-degrees and out-degrees [10] satisfy the following conditions:
   - $indeg(v') = outdeg(v') + 1$;
   - $indeg(v'') = outdeg(v'') - 1$.
3) The following is fulfilled for all other nodes: $\forall v \in V' \setminus \{v', v''\}$: $indeg(v) = outdeg(v)$.

To search for the Eulerian path, one can use, for example, a cycle-based algorithm (Hierholzer's algorithm) [10], whose complexity is $O(|E'|)$. Verification of the conditions for the existence of the Eulerian path initially consists in verifying the graph's strong connectivity, for example, using the adapted Kosaraju-Sharir's algorithm, whose complexity for two depth-first search traversals is $O(|V'|+|E'|)$ as in [6]–[9], [12]. The in-degrees and out-degrees for each node can be also found in $O(|V'| + |E'|)$. That is, the total complexity of checking the existence and finding the Euleraian path is $O(|V'| + |E'|)$.

However, as a rule, the graph of some software and hardware DST will not contain the Eulerain path, therefore, in most cases, it is not possible to use such an efficient algorithm in practice. If there is no Eulerian path in the graph, the problem is reduced to the Chinese Postman Problem, in which the number of the graph's edges with repeated traversals is minimized. This problem may be also solved only if the graph is strongly connected. The difficulty of finding the optimal path is $O(|V'|^2 \times |E'|)$ for a polynomial time [10], [11].

Thus, the solution to the testing problem, which consists in finding a complete and optimal path through the nodes of the directed graph, will exist if and only if G'_m is a strongly connected graph. Moreover, for a strongly connected graph, this problem can be solved for a polynomial time, that is, it belongs to the class of problems P, and not NP [6], [8].

III. Results and Discussion

The article proposes an approach to verifying fulfillment of the testing problem for software and hardware DST using the provisions of the automata and graph theories. In accordance with this approach, it is impossible to solve this problem for some software and hardware data security tools (the graph is not connected); for the remaining DST the problem may be solved in the worst case for a polynomial time using well-known graph algorithms.

It should be also noted that the testing problem can be changed and include negative testing, that is, verifying "non-fulfillment" of security functions in states $V \setminus V_{sf}$. The approaches and algorithms used to solve such a testing problem are completely similar, except for the necessity to apply them to the whole graph G_m, and not to the derived graph G'_m obtained after deleting some nodes and edges that are unnecessary for the initial problem.

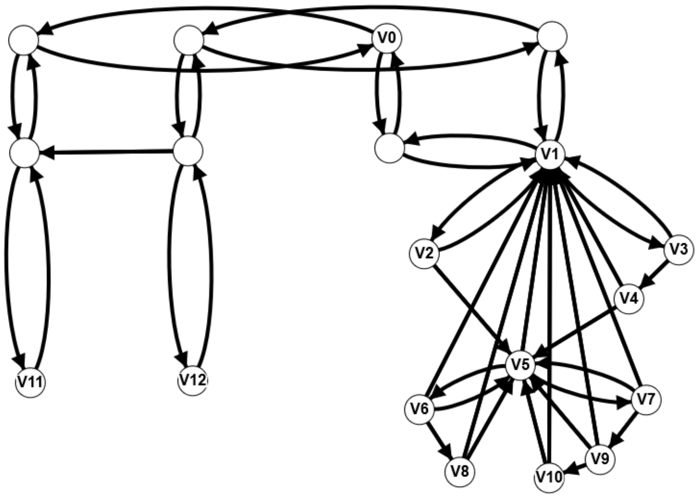


Fig. 1. Graph G_{acx}, where $v_0, \dots v_{12}$ are the states with potentially computable security functions, unlabeled states are unnecessary for the testing problem.

Applying the proposed approach in practice allowed the authors to eliminate the shortcomings of the process and of the test results of the developed access control subsystem "Accord-X", which is built on the basis of Accord-TSHM trusted start-up hardware module [13]. Using this approach, transition graphs corresponding to the considered DST were obtained, which are shown in Fig. 1–3.

One of the optimal traversal for the resulting graph is shown in Fig. 4.

The obtained graph and its optimal traversal allowed to ensure the completeness of testing of this data security tool,

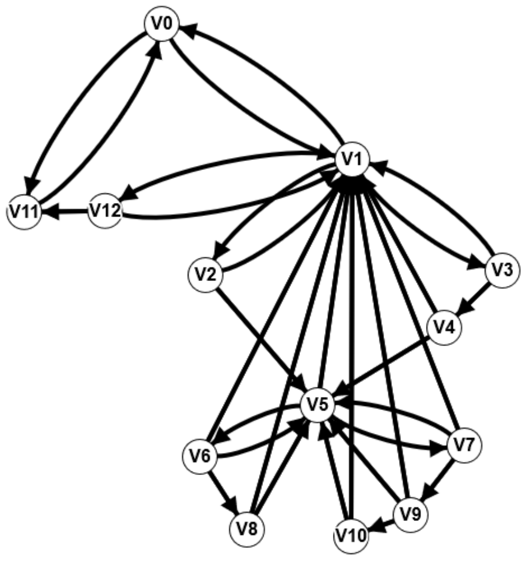


Fig. 2. Graph G'_{acx} – not Eulerian graph (for v_1 there are 7 more incoming edges than outgoing), but strongly connected.

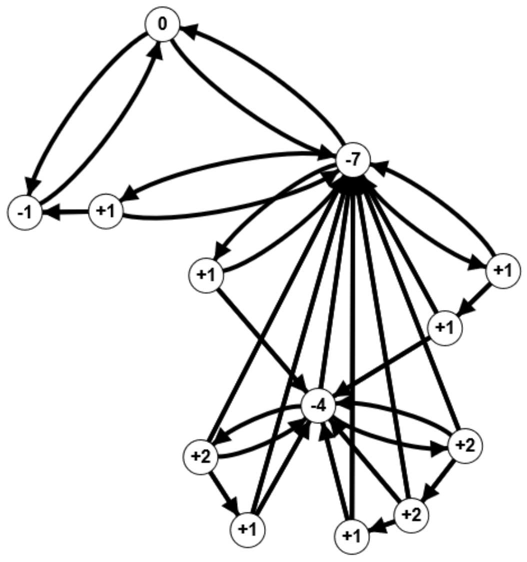
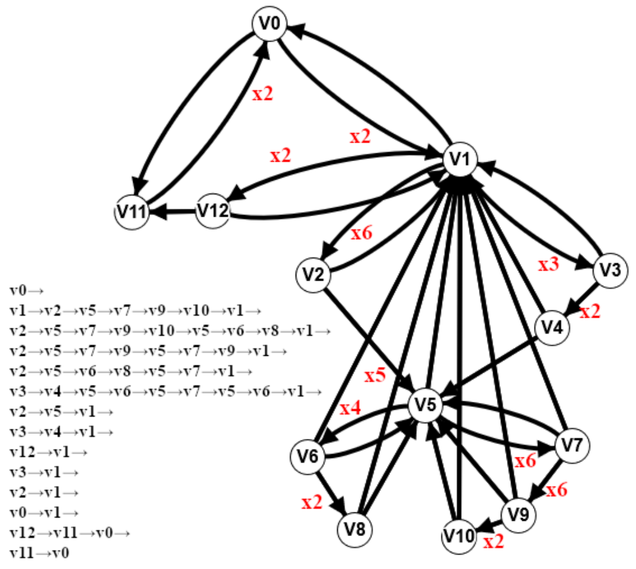


Fig. 3. Graph G'_{acx}, there are corresponding values of $outdeg(v) - indeg(v)$ inside the nodes, optimal duplicate-edges must be added for all imbalanced nodes with nonzero value in order to reduce the graph to Eulerian.



v0→
v1→v2→v5→v7→v9→v10→v1→
v2→v5→v7→v9→v10→v5→v6→v8→v1→
v2→v5→v7→v9→v5→v7→v9→v1→
v2→v5→v6→v8→v5→v7→v1→
v3→v4→v5→v6→v5→v7→v5→v6→v1→
v2→v5→v1→
v3→v4→v1→
v12→v1→
v3→v1→
v2→v1→
v0→v1→
v12→v11→v0→
v11→v0

Fig. 4. One of the optimal traversal for G'_{acx}, for edges labeled with the number of repeated passes virtual duplicate-edges were constructed to process the algorithm.

which could not be provided earlier due to some transition's omission.

References

[1] B. Beizer, Software testing techniques, 2nd ed., Dreamtech, 2003.

[2] M. Broy, B. Jonsson, J. P. Katoen, M. Leucker, A. Pretschner, Model based testing of reactive systems, LNCS 3472, Springer Berlin Heidelberg, 2005.

[3] V. V. Kulyamin, "Model-based testing. Lecture course in VMiK Moscow State University", http://panda.ispras.ru/ kuliamin/mbt-course.html (in Russian).

[4] I. B. Burdonov, A. S. Kossatchev, and V. V. Kulyamin, "Application of finite automatons for program testing", Programming and Computer Software, Springer, vol. 26 (2), 2000, pp. 61–73.

[5] T. M. Kanner, "Applicability of software testing methods to software and hardware data security tools", in Glob. J. Pure Appl. Math., vol. 12(1), 2016, pp. 167–190.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, & C. Stein, Introduction to algorithms, 3rd ed., MIT press, 2009.

[7] R. Sedgewick, Algorithms in C. Part 5: Graph algorithms, 3rd ed., Addison-Wesley Professional, 2001.

[8] R. Sedgewick, K. Wayne, Algorithms, 4th ed., Addison-Wesley Professional, 2011.

[9] T. Roughgarden, Algorithms illuminated. Part 2: Graph algorithms and data structures, Soundlikeyourself Publishing, 2018.

[10] S. S. Skiena, The Algorithm design manual, 2nd ed., Springer, 2010.

[11] J. Edmonds, E. L. Johnson, "Matching Euler tours and the chinese postman", Mathematical programming, vol. 5(1), 1973, pp. 88–124.

[12] M. Sharir, "A strong connectivity algorithm and its applications to data flow analysis", Computers and Mathematics with Applications, vol. 7(1), 1981, pp. 67–72.

[13] A. M. Kanner, L. M. Ukhlinov, "Access control in GNU/Linux", Inf. Secur. Quest., vol. 3, 2012, pp. 35–38 (in Russian).