Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 190 (2021) 408–413

Procedia Computer Science

www.elsevier.com/locate/procedia

2020 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: Eleventh Annual Meeting of the BICA Society

Algorithm for Optimal and Complete Testing of Software and Hardware Data Security Tools

Tatiana M. Kanner[a], Andrey M. Kanner[a,*], Anna V. Epishkina[a]

[a]*National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Kashirskoe highway 31, Moscow, 115409, Russian Federation*

Abstract

The paper considers the disadvantages of existing approaches to testing software and hardware data security tools in order to confirm compliance of the implemented functionality with the declared characteristics. It demonstrates the necessity of ensuring completeness and optimality of testing. The paper describes some testing approaches based on the development of mathematical models using the automata theory and ensuring completeness of testing, but leaving the question of its optimality open. It describes the approach proposed earlier by the authors, which ensures both completeness and optimality of testing using the graph theory. In accordance with this approach, the software and hardware data security tool is represented as a directed graph without loops or multiple edges. The graph vertices correspond to the states of the software or hardware component, and the directed edges correspond to the transitions of the data security tool from one state to another when performing non-target functions or security functions. On the basis of this approach, the authors propose an algorithm for solving the problem of testing software and hardware data security tools using a number of well-known algorithms on graphs. In the paper, it is substantiated that a solution to the problem of ensuring completeness and optimality of testing exists if and only if any vertex of the derived graph obtained by removing all unused vertices and edges either belongs to a directed chain or lies in a strongly connected component. Application of the proposed algorithm for solving the problem of testing one of the software and hardware security tools is considered, and the possibility of its application in practice is confirmed.

Keywords: Software and hardware data security tools; completeness and optimality of testing; directed graphs; directed chain; strongly connected components; route inspection problem
2010 MSC: 05C20; 05C38; 05C45; 05C85; 68M07; 68R10; 93A30

* Corresponding author.
E-mail address: kanner@mail.ru

1877-0509
10.1016/j.procs.2021.06.049

1. Introduction

Any developed software and hardware data security tools (DST), like any other software and/or hardware, must be tested in order to confirm compliance of the implemented functionality with the declared characteristics. During testing, however, it is often problematic to justify that the DST has been tested in all possible states with all possible combinations of parameters and settings, that is, to justify fullness, and as a consequence, completeness of testing. In addition, even if you ensure completeness of testing, in most cases it will not be conducted in an optimal way and only in some cases – in an optimal way with implementation of the minimum number of steps and iterations required for it. As a rule, however, completeness and optimality of testing are not ensured in practice. This entails double costs – some functionality of the data security tool remains untested under certain conditions, although it has taken a rather long time to conduct such a defective testing.

In [1], it is demonstrated that testing software and hardware data security tools, in contrast to software data security tools, has some specific features associated with presence of a hardware component and its use in the implementation of some security functions. In this paper, let us assume that all such features have already been taken into account and all security functions are feasible in certain states of the software or hardware component of the data security tool.

Works of various authors [1, 2, 3, 4, 5] propose testing approaches based on the development and use of mathematical models (the so-called Model-based testing), most often using the automata theory. Such approaches may be used to ensure completeness of testing, with the question of its optimality remaining open. In [6], the authors formulated the testing problem taking into account its completeness and optimality, and proposed an approach to verifying feasibility of the problem of testing software and hardware data security tools using mainly provisions of the graph theory. On the basis of this approach, in this paper the authors propose an algorithm for testing security functions of software and hardware data security tools, which allows them to be tested and ensures its optimality and completeness. Moreover, the paper describes application of this algorithm for the selected software and hardware data security tool in order to test it in practice.

2. Materials and methods

By analogy with [6], let us denote a directed graph without loops and multiple edges (simple digraph), corresponding to the software and hardware DST, as $G_m = (V, E)$, where:

- V is a set of graph vertices corresponding to the states of the software or hardware components of the data security tool;
- $E \subseteq V \times V$ is a set of directed edges (arcs) of the graph – transitions of the DST from one state to another when performing non-target functions or security functions.

The testing problem consists in traversing from some initial vertex $v_0 \in V$ of all the graph edges directed to the vertices, where any security functions of the DST can be implemented – $V_{sf} \subseteq V$. In such a case ensuring completeness and optimality of testing is based on finding a path going through all the edges at least once in the minimum number of transitions. Thus, in accordance with [6, 7, 8, 9], the testing problem is reduced to the Chinese postman problem also known as the Route Inspection Problem, or in some cases, to the problem of finding the Eulerian path.

Therefore, on the basis of [6], we may propose the following algorithm for solving the problem of testing security functions of the software and hardware DST, which consists in performing the following steps:

1. To construct G'_m corresponding graph on the basis of G_m initial graph of the software and hardware DST by removing some vertices and edges not used in solving the testing problem, using the rules of preserving connectivity of the remaining vertices given in [6].
2. If there are isolated vertices in G'_m (not removed when constructing G'_m graph), the testing problem cannot be solved, since it will be impossible to meet the testing completeness condition specified in [6].
3. To check that any $v \in V$ vertex in G'_m either belongs to a directed chain or lies in a strongly connected component (SCC), that is has the form shown in Fig. 1. This can be done using the Kosaraju-Sharir Algorithm in two depth-first searches of the graph [7, 10, 11, 12]: to find all connected components and check connectivity between

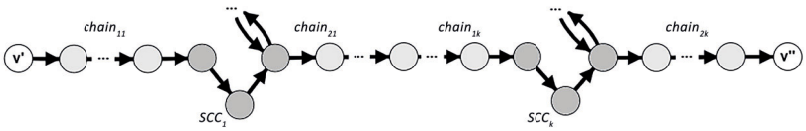

Fig. 1. General form of the graph for the purpose of solving the testing problem – individual vertices, chains or connected components can be missing, $k \in \mathbb{N}_0$.

the initial v_0 vertex and all other vertices. Otherwise, the testing problem cannot be solved, since it will also be impossible to meet the completeness condition given in [6].

4. For all the vertices of G'_m graph, it is necessary to calculate the difference between out-degrees and in-degrees. If the graph includes only balanced vertices (the difference is "0"), then there is the Eulerian cycle in the graph. If the graph includes only two unbalanced vertices with a difference of "1" and "−1", and the other vertices are balanced, then there is the Eulerian path in the graph. In these cases, it is necessary to continue the algorithm from paragraph 9.
5. Otherwise, it is needed to consider unbalanced vertices separately in order to find the paths that need to be traversed once again while fulfilling the requirement of minimizing edge traversals stipulated by the testing optimality condition given in [6]. Unbalanced vertices can be represented as a bigraph.
6. Using the algorithm for finding the shortest paths for all the vertices in the resulting bigraph – the Floyd-Warshall algorithm [7, 10], to calculate the length of the shortest paths from vertices with a negative difference between out-degrees and in-degrees to vertices with a positive difference.
7. Using the Hungarian algorithm, the Kuhn-Munkres algorithm or the Ford-Fulkerson algorithm [7, 10, 11], to select from all the combinations of possible shortest paths those paths that, when reused, allow all the vertices to become balanced, with the total length of such paths remaining minimal. When adding a path from a vertex with a negative difference between out-degrees and in-degrees to a vertex with a positive difference, the difference in both vertices will change exactly by "1" (will increase by "1" for the first one and decrease by "1" for the second one). At the same time, the difference will not change for all other vertices, since only the incoming and outgoing edges can be added (the total difference will remain "0").
8. To add edges for the additional paths selected at the previous step to G'_m graph. Since all the vertices are balanced now, there is the Eulerian cycle.
9. Using a cycle-based algorithm known as the Hierholzer's algorithm [9], to construct the Eulerian cycle in the resulting graph. Traversal of the edges constructed at the previous step is equivalent to a repeated traversal of the corresponding edges of G'_m graph. Some iterations of the Eulerian cycle can be interchanged, since the optimal solution may be the not only one possible.

The block diagram of the proposed testing algorithm is shown in Fig. 2.

3. Results and conclusion

Let us demonstrate the possibility of using the proposed algorithm for the selected software and hardware DST – PCDST SHIPKA [1]. This software and hardware DST can be found in 23 different states for its software and/or hardware components; the initial state v_0 is the state when the hardware component is off (not connected to the USB port) and is not initialized. To switch to other states, non-target functions (initialization, formatting, connection and disconnection of the hardware component) or security functions (identification and authentication, generation of cryptographic keys, performing cryptographic operations, etc.) are performed. The state graph for this DST is shown in Fig. 3.

To find a solution to the testing problem, it is necessary that any $v \in V$ vertex of G'_m graph derived from G_m graph either belongs to a directed chain or lies in a strongly connected component. If a similar condition is satisfied for the initial G_m graph, the testing problem also has a solution. Applying the Kosaraju-Sharir Algorithm given in the testing algorithm, let us perform two depth-first searches of the graph shown in Fig. 4 and 5.

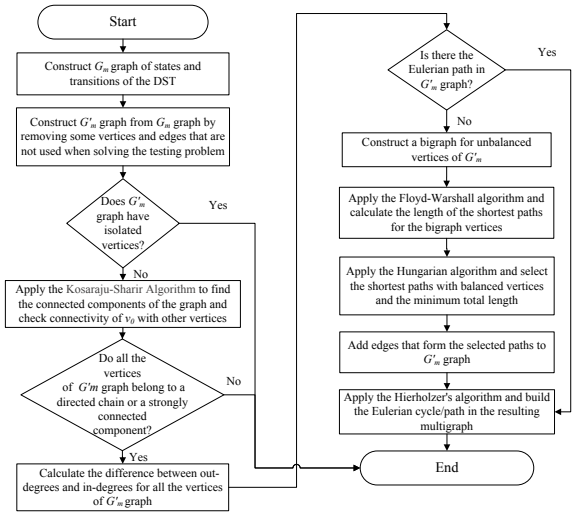

Fig. 2. Block diagram of the algorithm for solving the problem of testing security functions of software and hardware DST.

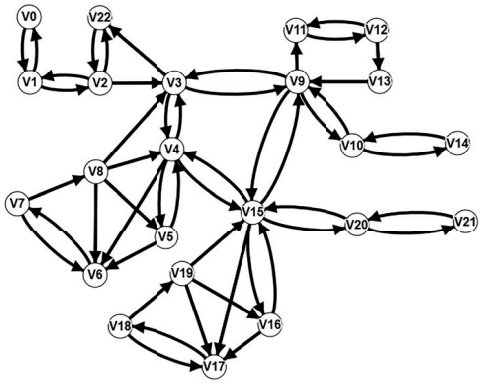

Fig. 3. G_m graph for the DST under research.

Thus, the initial G_m graph is strongly connected, meaning that the proposed algorithm for solving the testing problem using graph theory can be applied, and the problem of testing its security functions can be solved. As a result of applying the remaining steps of the proposed algorithm we obtain a path shown in Fig. 6, which is one of the optimal traversals of the graph ensuring completeness of testing.

It should also be noted that in this case the testing problem has a solution even for the initial G_m graph, and not only for its derived G'_m graph. Therefore, a solution can be obtained not only to the functional, but also to the negative testing of this data security tool.

It should be noted that the proposed algorithm involves traversing the graph vertices using one sequence of transitions. If the graph traversal cannot be conducted using one sequence of transitions according to the above algorithm, this means that the graph has several unconnected branches. This means that there is an error in the data security tool, since it is impossible to ensure that all security functions are performed correctly without disrupting other security functions. Therefore, the authors do not consider the option with several sequential traversals of the DST graph.

The paper demonstrates that a solution to the problem of ensuring completeness and optimality of testing exists if and only if any $v \in V$ vertex of the corresponding G'_m graph either belongs to a directed chain or lies in a strongly connected component, as in Fig. 1. Since the initial G_m graph may violate these conditions, it is sufficient that such conditions are fulfilled at least for the derived G'_m graph. However, if the initial graph also satisfies the above condi-

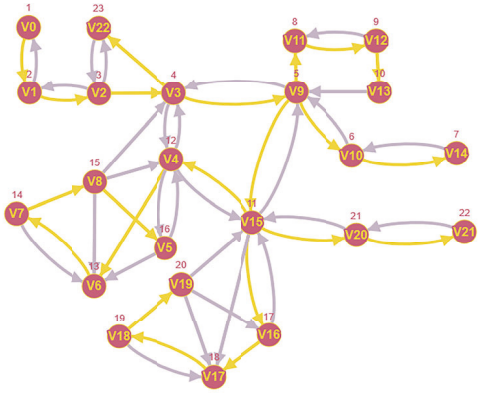

Fig. 4. Results of the depth-first search of G_m graph from the initial v_0 vertex; all the vertices are accessible from v_0 – G_m graph is at least weakly connected.

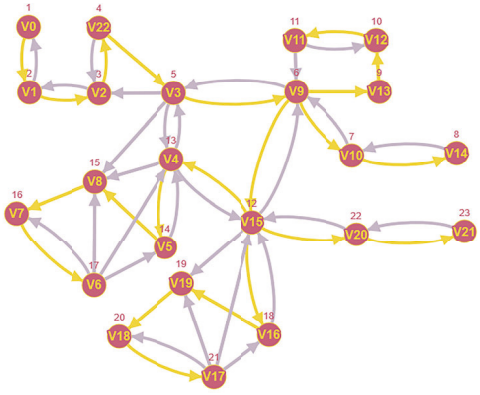

Fig. 5. Results of the depth-first search of the reverse G_m graph from the initial v_0 vertex; all the vertices are accessible from v_0 – G_m graph is strongly connected.

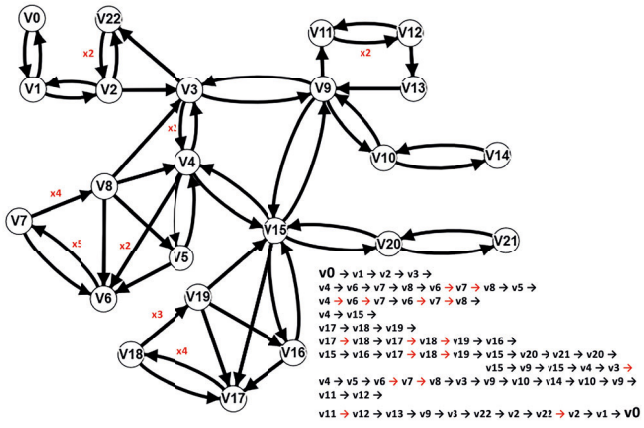

Fig. 6. One of the optimal traversals of G_m graph, the marked edges should be traversed several times

tions, a solution to the problem of ensuring completeness and optimality exists as well. Otherwise, the testing problem has no solution.

The complexity of the proposed algorithm for solving the testing problem is equal to the maximum complexity of the above well-known algorithms as used on graphs, since all the algorithm steps are performed sequentially and there is no nesting. Accordingly, this algorithm has polynomial complexity $O(|V|^3)$.

References

[1] Kanner, Tatiana (2016) "Applicability of software testing methods to software and hardware data security tools." *Glob. J. Pure Appl. Math.* **12** (1): 167–190.

[2] Beizer, Boris (2003) "Software testing techniques, 2nd ed." *Dreamtech*. doi:10.1002/stvr.4370020406

[3] Broy, Manfred, Jonsson, Bengt, Katoen, Joost-Pieter , Leucker, Martin, and Pretschner, Alexander. (2005) "Model based testing of reactive systems." *LNCS 3472, Springer Berlin Heidelberg*. doi:10.1007/b137241

[4] Burdonov, Igor, Kossatchev, Alexander, and Kulyamin, Victor. (2000) "Application of finite automatons for program testing." *Programming and Computer Software. Springer* **26** (2): 61–73. doi:10.1007/BF02759192

[5] Kulyamin, Victor "Model-based testing. Lecture course in VMiK Moscow State University." `https://sites.google.com/site/swtestcourse/`, last accessed 2020/11/06 (in Russ.).

[6] Kanner, Andrey, and Kanner, Tatiana. (2020) "Testing Software and Hardware Data Security Tools Using the Automata Theory and the Graph Theory." *Proceedings of Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology, IEEE*: 615–618. doi:10.1109/usbereit48449.2020.9117757

[7] Sedgewick, Robert (2001) "Algorithms in C. Part 5: Graph algorithms, 3rd ed." *Addison-Wesley Professional*.

[8] Edmonds, Jack, and Johnson, Ellis L. (1973) "Matching Euler tours and the Chinese postman." *Mathematical programming* **5** (1): 88–124. doi:10.1007/BF01580113

[9] Skiena, Steven (2010) "The Algorithm design manual., 2nd ed." *Springer*. doi:10.1007/978-1-84800-070-4

[10] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., and Stein, Clifford. (2009) "Introduction to algorithms, 3rd ed." *MIT press*.

[11] Sedgewick, Robert, and Wayne, Kevin. (2011) "Algorithms, 4th ed." *Addison-Wesley Professional*.

[12] Sharir, M. (1981) "A strong connectivity algorithm and its applications to data flow analysis." *Computers and Mathematics with Applications* **7** (1): 67–72. doi:10.1016/0898-1221(81)90008-0