Generating Test Cases from Role-Based Access Control Policies using Cause-Effect Graph

Yousef Khdairat, Khair Eddin Sabri*
Computer Science Department, King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan

* Corresponding author Email: k.sabri@ju.edu.jo
Manuscript submitted March 10, 2018; accepted May 8, 2018.
doi: 10.17706/jsw.13.9.497-505

Abstract: Role-based access control is one of the fundamental security models used to ensure the confidentiality and integrity of information by specifying policies and enforcing them through mechanisms. Usually, authorization constraints are defined on policies to enforce some regulations such as a user cannot be assigned to two conflicting roles. Once the RBAC mechanisms are implemented in a system, testing is performed to ensure the correctness of the implementation. Black-box testing is one approach for software testing where test cases are generated from the specification. The challenge of this approach is the huge number of test cases that can be generated.

This paper aims at reducing the number of test cases required to test the implementation of RBAC system. To achieve that, we use a cause-effect graph to specify policies, and then link authorization constraints to the cause-effect graph constraints. The specification of constraints within the cause-effect graph allows reducing the number of test cases by removing the useless cases due to authorization constraints. We illustrate our technique through an illustrative example with the aid of the BenderRBT tool. The results show that the number of test cases is significantly reduced.

Key words: Access control policy, Authorization constraints, Black box testing, cause-effect graph, Information security, role-based access control

1. Introduction

As a result of the emergence of recourse sharing in the computer system, the risks facing the protection of information systems are increased. These risks are related to the confidentiality, integrity, and availability of information. The confidentiality of information ensures that it is accessed by authorized users only. The integrity of information ensures that it can be changed by authorized users only. The availability of information ensures that the it is available to users when needed. Several researches are available in the literature related to information security such as the use of encryption to secure data [1], the analysis of information flow [2], and building a secure system [3].

Access control is one of the fundamental security concepts used to ensure the confidentiality and integrity of information. It determines the users that can access these resources. Access control consists of policies and mechanisms. Policies are guidelines for determining the authorized users to access information, whereas mechanisms are the methods used to implement these policies .

Role Based Access Control (RBAC) [4] is one of the most used models to represent access control policies. It has been analyzed and extended by many authors such as [5], [6] . RBAC assigns permissions to roles instead of users who are assigned to roles such that each user is authorized to access objects based on the permissions

assigned to his/her role. Usually, RBAC model states constraints on the policies defined in a system. These constraints can be seen as regulations imposed by the organizations to ensure the correctness and security of the system. For example, the Separation Of Duties (SOD) is a constraint that can be stated in an organization that prohibits users from playing two conflicting roles e.g., an employee cannot play the role of developer and reviewer as the employee developing the code should be different from the one reviewing it. Several researches are available in the literature to specify and enforce constraints such as [7]-[9]

To ensure that policies are correctly implemented, testing is usually performed on the code. Testing can be black-box based on the specification of the system or white-box based on the internal structure of the software. One of the black-box testing techniques is the cause-effect graph, which was generally used for hardware testing, but then was adapted to software testing by Elmendorf [10]. Cause-effect graph is a Boolean directed graph which represents a logical relationship between causes and effects such that if all causes are satisfied, then effects should be performed. Then, the graphs are transformed into decision tables that are used to generate test cases. In this paper, we employ cause-effect graph to represent and generate test cases from RBAC policies. The motivation of using cause-effect graph is due to its nature that makes it suitable to represent access control policies and generate test cases as both of them are based on true/false evaluation. The use of cause-effect graph enables us to specify role hierarchy. Also, it enables specifying several kinds of policies such as a user should be playing two roles in order to access a specific object. Furthermore, the main advantage of using cause-effect graph is that it allows specifying constraints on the causes part of the graph to reduce the number of generated test cases. These constraints are suitable to represent authorization constraints and therefore, reduce the number of generated test cases. Additionally, cause effect graph is a well-known technique in the literature used to specify and test several applications and many tools are available in the literature that can automatically generate test cases from the graph.

The structure of this paper is as follows. Section 2 summarizes literature review. Section 3 presents the proposed technique which is used to specify RBAC policies and constraints using cause-effect graph. Section 4 gives an illustrative example and the use of BenderRBT tool to specify policies and generate decision tables. Finally, we conclude in Section 5.

2. Literature Review

Our technique applies the cause-effect graph to generate test cases from the role based access control policies. Several researches are published on testing based on the cause-effect graph. For example, Paradkar, et al [11] developed a technique called CEG-BOR for specification-based testing. This technique consists of two phases. The first phase converts the informal software specifications to cause-effect graphs (CEG). The second phase applies the Boolean OperatoR (BOR) strategy to design and select test cases. Nursimulu and Probert [12] presented some drawbacks in the Myers' approach of the cause-effect graphing technique, and proposed solutions to resolve these limitations. Srivastava, et al., [13] presented an algorithm for creating decision table from CEG. Chung [14] developed a fault model for the cause-effect graph. Son, et al., [15] proposed a method to generate test cases based on model transformation from cause-effect graph and UML Diagram.

On other hand, several techniques are proposed in the literature for testing the implementation of access control policies and generating test cases, but none of these techniques uses cause-effect graph. For example, Pretschner, et al., [16] presented a model-based approach for testing access control requirements. A combinatorial testing was used to generate test cases. Martin [17] presented a test generation approach and its supporting tool called Targen. The author reduced the number of generated requests based on structural coverage information. They also presented an approach for redundant-rule detection based on change-impact analysis. Martin and Xie [18] presented a framework and its supporting tool called Cirg that generates tests based on change impact analysis. Li et al., [19] proposed an automatic approach to generate test cases for access control policies specified in XACML by using symbolic execution techniques. Masood et al., [20] presented a procedure to represent role-based access control policies using the finite state model, and then to generate test

cases. In [21], a finite state machine is used for testing RBAC mechanism. In [22], the authors proposed a framework to validate implemented policies against the intended rights. The framework relies on well-known software testing techniques such as mutation and combinatorial techniques. None of these methods reduces the number of test cases by taking the authorization constraints into consideration when generating test cases.

3. The Proposed Technique

In this section, we employ cause-effect graph to generate test cases. The idea is representing the specification of the system to be tested as a graph. Then, a decision table is constructed from the graph. Finally, the table is used to generate test cases. These steps are followed in our technique. However, we focus in this section on the way to represent RBAC policies using cause-effect graph, identify the authorization constraints, and then specify them within the graph.

The first step of the proposed technique is to represent RBAC policies as cause-effect graph. We show the representation of several kinds of RBAC policies, and the hierarchy of roles. The cause-effect graph consists of two parts: cause and effect. In our technique, we consider the roles and objects as causes and the access authorization as effect. To remove the complexity of specifying RBAC policies and to ensure the correctness of the specification, we represent the access to each object separately as one cause-effect graph. We add all the roles that can access a specific object to the same cause-effect graph. Therefore, other roles that are not included in the graph cannot access that object. Fig. 1 shows a typical RBAC policy that has two causes: the role R and the object O. Therefore, if a user is playing the Role R and requesting access to an object O, then the access is authorized. Otherwise, a user not playing the role R is not authorized.

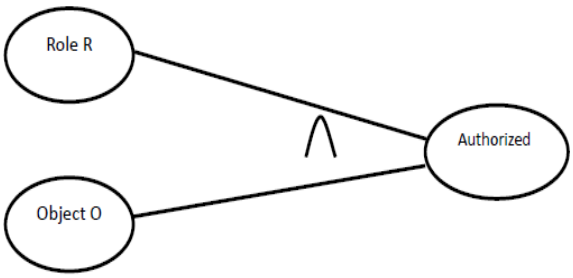


Fig. 1. Cause-effect graph for an RBAC policy.

By using the cause-effect graph, we can specify composite policies such as a user playing the role R1 or the role R2 is allowed to access an object O as shown in Fig. 2. Or a user should be playing two roles simultaneously to be able to access a specific object as shown in Fig. 3.

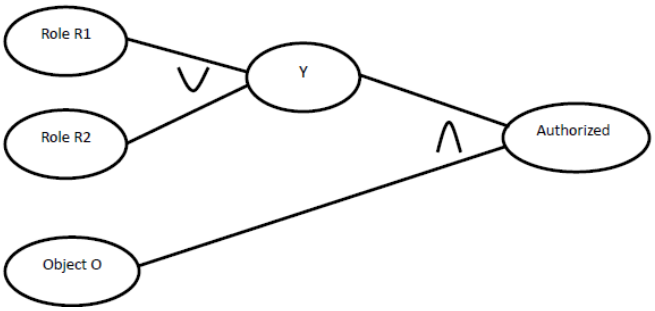


Fig. 2. Cause-effect graph for a composite RBAC policy.

Role hierarchy is an important feature of RBAC. It specifies the authority relation between the roles in an organization. We specify the role hierarchy by inserting explicitly the roles to the graph that contains their

descendents. For example, if we have R1>R2, then we insert the role R1 to each graph that contains R2 as shown in Fig. 4.

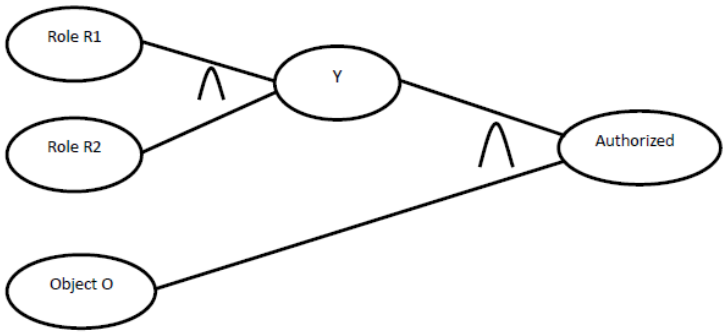


Fig. 3. Cause-effect graph for a composite RBAC policy.

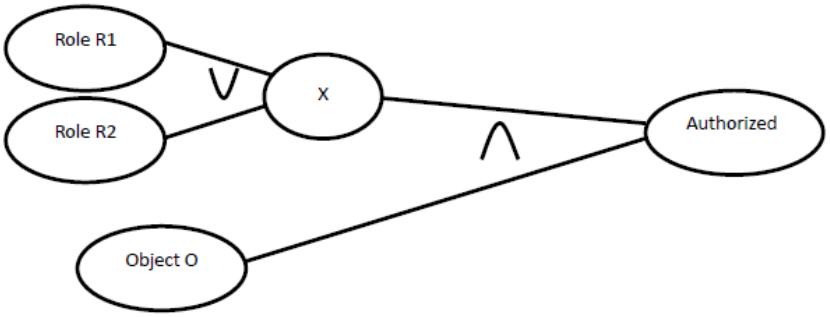


Fig. 4. Cause-effect graph for role hierarchical.

3.1. Constraints

In this section, we identify two authorization constraints that are widely used and described in the literature which are the separation of duties and the pre-request constraint. Then, we specify them using cause-effect graph constraints. Such specification reduces the number of test cases that can be generated from the specifications. It is of note that the satisfaction of these constraints should be validated separately as in [7]-[9]. In this paper, we assume that all constraints are correctly implemented and enforced.

Separation of duties constraint: This constraint states that a user cannot play two conflicting roles. For example, a user cannot play the role of teacher and student in a school system. This constraint can be represented by cause-effect graph using the exclusive constraint (E) between the conflicting roles. For example, Fig. 5 shows the SOD constraints on the roles R1 and R2.

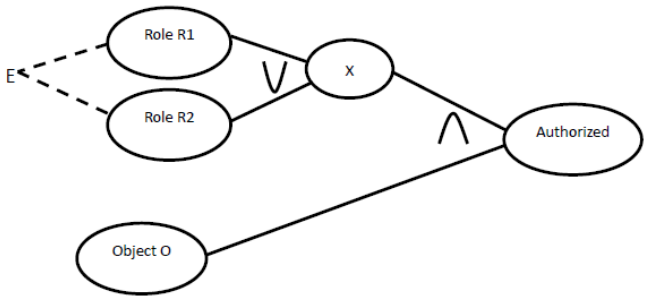


Fig. 5. Cause-effect graph representing separation of duties constraint.

A variation of the SOD constraint is the one and only one constraint which is different from the SOD as it requires one of the roles to be always true i.e, each user should be playing exactly one of the two roles.

Prerequisite constraint: This constraint states that for a user to play a role should be already playing another role. For example, a user playing the role of TA should be already playing the role of student. The constraint can be specified in cause-effect graph by using the constraint REQUIRED on roles. For example, Fig. 6 shows that playing the role R1 is a prerequisite for the role R3.

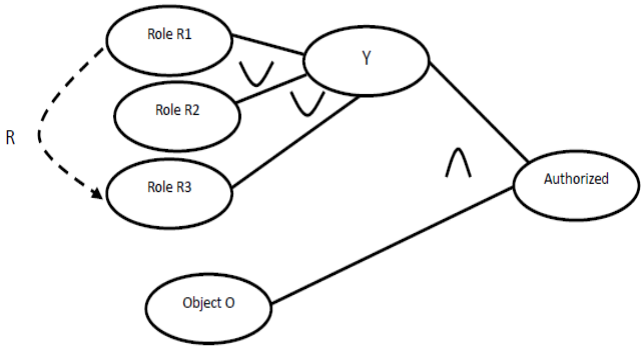


Fig. 6. Cause-effect graph to represent prerequisite constraint

The next step is building the decision table based on the cause effect graph taking into considering the specified constraints. The decision table contains the values of true and false or 1 and 0 which can be interrupted as a user is playing or not playing a role. For example, Table 1 shows the decision tree of the graph given in Fig. 4.

Table 1. An Example of a Decision Table

	#1	#2	#3	#4
R1	1	1	0	0
R2	1	0	1	0
O	1	1	1	1
Authorized	1	1	1	0

Table 1 states four test cases that can be generated. The first one states that a user playing the roles R1 and R2 is authorized to access the object O. The second case states that a user playing the role R1 only is authorized to access the object O. The third case states that a user is playing the role R2 only is authorized to access the object O. The fourth case states that a user is neither playing the role R1 nor the role R2 is not authorized to access the object O. The generation of the table from a graph is described in the literature such as in [13].

4. Illustrative Example

In this section, we present an example that includes many policies and shows the generated test cases. To automate the generation of test cases, we use a tool called BenderRBT [23]. This tool generates automatically the decision tables from the graphs. Also, the tool allows specifying constraints on the cause-effect graph. We use these constraints to represent constraints on access control policies which are used to reduce the number of generated test cases. It is of note that this tool produces even fewer test cases as it uses algorithms to eliminate some of the test cases that are most likely not giving a failure. The description of these algorithms are not within the scope of this paper.

Assume that the described system contains six roles denoted by R1, R2, R3, R4, R5, and R6 and assume that the role R6 has more authority than the role R4 i.e., R6 > R4. Also, assume that the system contains five objects denoted by A, B, C, D, and E. The authorizations of roles on objects are given through the following policies:

Policy 1: Users playing the role R2 can access objects A, B.

Policy 2: Users playing the role R3 can access object B.

Policy 3: Users playing the role R3 or the role R5 can access objects C, D.
Policy 4: Users playing the role R6 can access objects B, C, D.
Policy 5: Users playing both the role R5 and the role R6 can access object A.
Policy 6: Users playing the roles R1, R4 or R5 can access object E.

Assume that the system has some regulations on the specified policies stated as:

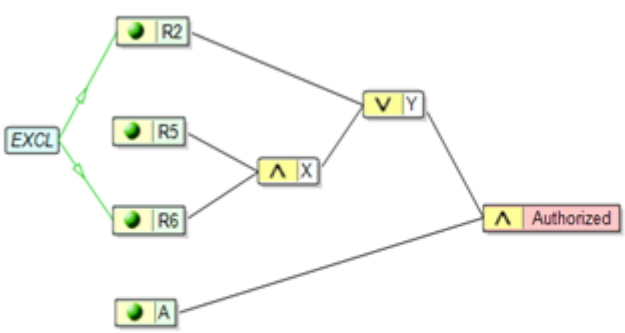
- Users cannot play role R2 and role R6 simultaneously.
- Users cannot play role R3 and role R6 simultaneously.
- Users should play one and only one of the roles R4 and R5.
- Playing the role R6 is a prerequisite to play the role R1

We use the cause-effect graph to describe the authorization access for each object and state the required constraints. We use BenderRBT tool to draw the graphs from which the tool generates automatically the decision trees used to obtain the test cases.

Fig. 7, shows that the object A can be accessed by a user playing the role R2 or by a user playing the roles R5 and R6 together. We add the constraint that R2 and R6 cannot be played simultaneously by a user. The tool generates automatically the decision tree that shows five test cases to be used to test the authorization for accessing object A. This is a reduction from taking all test cases (eight cases). The first test case represents an authorization request from a user playing the role R5 only. The result should be that permission is not granted (unauthorized). Similarly, the second test case shows the test case of a user playing only role R6. The third case shows that a user playing both the roles R5 and R6 can access the object A. The last test case which states that if an object is not A, it should be ignored when generating test cases. This is because it is covered by the other graphs i.e, if the object is not A, then it should be B, C, D, or E. The test case that a user is playing the roles R2 and R6 is removed from the table because of the stated constraints.

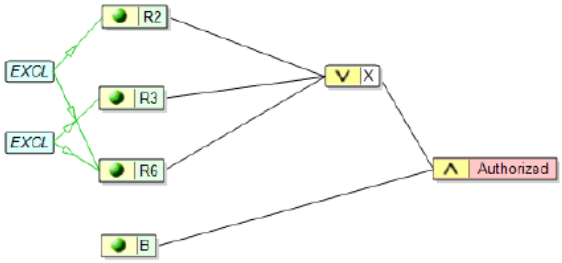
Fig. 8 shows the specification that represents the access to object B and its decision table. It shows that the roles R2, R3, and R6 can access the object B. Also, it shows the SOD constraints between R2 and R6, and between the roles R3 and R6. As observed in the table, no case shows a user playing roles R2 and R6 or a user playing roles R3 and R6.

Fig. 9 shows the graph that represents the access to object C. It contains one constraint which states that a user cannot play the roles R3 and R6. Therefore, none of the generated test cases gives true to R3 and R6 at the same time. Fig. 10 shows the graph that represents the access to object D. It contains one constraint which states that a user cannot play the roles R3 and R6 as well.



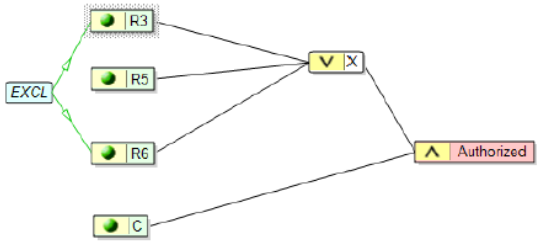
		UI Type	TEST#1	TEST#2	TEST#3	TEST#4	TEST#5
New/Old							
Causes:							
R5		Input	T	F	T	f	t
R6		Input	F	T	T	F	f
R2		Input	F	F	F	T	t
A		Input	T	T	T	T	F
Effects:							
X			F	F	T	F	f
Y			F	F	T	T	T
Authorized	(obs)	Action	F	F	T	T	F

Fig. 7. Cause-effect graph and decision table for object A.



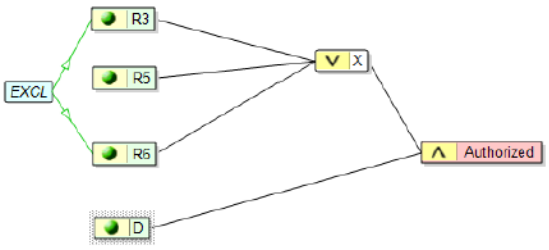
		UI Type	TEST#1	TEST#2	TEST#3	TEST#4	TEST#5
New/Old							
Causes:							
R3		Input	F	F	T	F	f
R6		Input	F	T	F	F	t
R2		Input	T	F	F	F	f
B		Input	T	T	T	T	F
Effects:							
X			T	T	T	F	T
Authorized	{obs}	Action	T	T	T	F	F

Fig. 8. Cause-effect graph and decision table for object B.



		UI Type	TEST#1	TEST#2	TEST#3	TEST#4	TEST#5
New/Old							
Causes:							
R5		Input	F	F	T	F	f
R3		Input	F	T	F	F	f
R6		Input	T	F	F	F	t
C		Input	T	T	T	T	F
Effects:							
X			T	T	T	F	T
Authorized	{obs}	Action	T	T	T	F	F

Fig. 9. Cause-effect graph and decision table for object C.



		UI Type	TEST#1	TEST#2	TEST#3	TEST#4	TEST#5
New/Old							
Causes:							
R3		Input	F	F	T	F	f
R6		Input	F	T	F	F	t
R5		Input	T	F	F	F	f
D		Input	T	T	T	T	F
Effects:							
X			T	T	T	F	T
Authorized	{obs}	Action	T	T	T	F	F

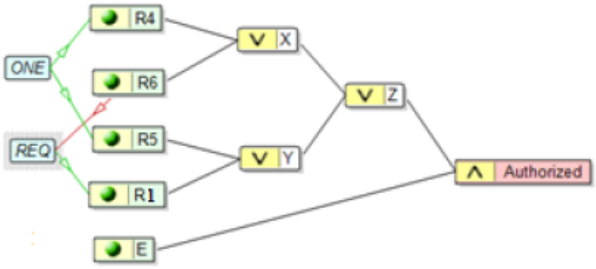
Fig. 10. Cause-effect graph and decision table for object D

Fig. 11 shows the graph that represents the access to object E. The Role 6 is added to the figure as R6>R4. It contains two constrains. The first one states that a user should play one of the roles R4 and R5 and the other constraint states that the role R6 is required for the role R1.

It is of note that the tool uses other algorithms to reduce further the number of test cases. This issue is out of the scope of this paper. Our aim is to show the availability of tools that can be used to facilitate the representation of graphs and the construction of decision tables.

The illustrative example shows that our technique enables generating test cases for all possible cases from the specification. It removes cases that cannot occur due to the regulations stated by the organization. Removing such cases reduces significantly the number of required test cases as seen in the example. This paper does not

cover the validation of enforcing the constraints as it is another issue being discussed widely by the literature. There are more than one approach that can be followed to generate test cases from the tables, either the system applies the test cases for all users playing the role in the system or just take one representative user for the testing. This depends on the complexity of the system and the available budget given for testing the system.



		UI Type	TEST#1	TEST#2	TEST#3
New/Old					
Causes:					
R4		Input	F	f	T
R1		Input	F	f	F
R5		Input	T	t	F
R6		Input	F	f	F
E		Input	T	F	T
Effects:					
X			F	f	T
Y			T	t	F
Z			T	T	T
Authorized	{obs}	Action	T	F	T

Fig. 11. Cause-effect graph and decision table for object E.

5. Conclusion

In this paper, we employ the cause-effect graph to represent RBAC policies and generate test cases. We are able to specify different kinds of policies and to represent role hierarchy. We integrate the authorization constraints such as SOD and prerequisite constraints within the cause-effect graph to reduce the number of test cases. This technique reduces the number of test cases as observed in the illustrative example. Also, as the number of constraints increases, the number of generated test cases is reduced. The use of a tool enables generating the test cases automatically within one second in the presented example.

One of the limitations of our technique is the complexity and time consumed in drawing the graphs. As the number of roles and objects increases the size of the graph becomes larger. However, we solved this problem partially by representing the graph of each object separately instead of having one graph for all objects. Another solution could be generating the graph automatically from the policies. As a future work, we intend to apply our technique on real applications and analyse the complexity of generating test cases.

References

[1] Sabri, K. E. (2014). Algebraic analysis of object-based key assignment schemes. *Journal of Software, 9(8), 2033-2042.*

[2] Sabri, K. E., Khedri, R., & Jaskolka, J. (2009). Verification of information flow in agent-based systems. *International Conference on E-Technologies*, 252-266.

[3] Mouratidis, H. (2011) Secure software systems engineering: The secure tropos approach. *Journal of Software, 6(3), 331-339.*

[4] Sandhu, R., Coyne, E., Feinstein, H., & Youman, C. (1996). Role-Based Access Control Models, *Computer 29(2), 38-47.*

[5] Sabri, K. E., & Obeid, N. (2016). A temporal defeasible logic for handling access control policies. *Applied Intelligence , 44(1), 30-42.*

[6] Mitra, B., Sural, S., Vaidya, J., & Atluri, V. (2017). Migrating from RBAC to temporal RBAC. *IET Information*

Security 11(5), 294 - 300.

[7] Alswae'r, N., & Sabri, K. E. Formal specification of constraints on role-based access control policies. *New Trends in Information Technology*, 28-33.

[8] Crampton, J. (2003) Specifying and enforcing constraints in role-based access control. *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*.

[9] Sabri, K. E., & Hiary, H. (2016) Algebraic model for handling access control policies. *Procedia Computer Science,* 653-657.

[10] Elmendorf, W. (1974) *Functional Analysis using Cause-Effect Graphs.*, Poughkeepsie, N. Y..

[11] Paradkar, A., Tai, K. C., & Vouk, M. A. (1997). Specification-based testing using cause-effect graphs. *Annals of Software Engineering*, *4(1-4),* 133-157.

[12] Nursimulu, K., & Probert, R. L. (1995) Cause-effect graphing analysis and validation of requirements. *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*.

[13] Srivastava, P. R., Patel, P., Chatrola, S. (2009) Cause effect graph to decision table generation. *ACM SIGSOFT Software Engineering Notes*, *34(2),* 1-4.

[14] Chung, I. (2014). Investigating effectiveness of software testing with cause-effect graphs. *International Journal of Software Engineering and Its Applications*, *8(7),* 41-54.

[15] Son, H., Kim, R., & Park, Y. (2014). Test case generation from cause-effect graph based on model transformation. *Proceedings of the International Conference on Information Science and Applications*.

[16] Pretschner, A., Mouelhi, T., & Traon, Y. L. (2008). Model-based tests for access control policies. *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation*.

[17] Martin, E. (2006). Automated test generation for access control policies. *Proceedings of the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications.*

[18] Martin, E., & Xie, T. (2007) Automated test generation for access control policies via change-impact analysis. *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*.

[19] Li, Y., Li, Y., Wang, L., & Chen, G. (2014) Automatic XACML requests generation for testing access control policies. *Twenty-Sixth International Conference on Software Engineering and Knowledge*, 217-222.

[20] Masood, A., Bhatti, R., Ghafoor, A., & Mathur, A. (2009). Scalable and effective test generation for role-based access control systems. *IEEE Transactions on Software Engineering*, *35(5),* 654-668.

[21] Damasceno, C. D. N., Masiero, P. C., & Simao, A. (2016). Evaluating test characteristics and effectiveness of FSM-based testing methods on RBAC systems. *Proceedings of the 30th Brazilian Symposium on Software Engineering*, 83-92.

[22] Bertolino, A., Daoudagh, S., Lonetti, F., & Marchetti, E. (2016) Testing access control policies against intended access rights. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 1641-1647.

[23] Bender RBT Inc (2006). *Cause-Effect Graphing User Guide.* http://www.benderrbt.com/BenderRBT-Cause-Effect%20Graphing%20User%20Guide.pdf

Yousef Khdairat has a master degree in computer science from The University of Jordan since 2016. His main research interest is software testing.

Khair Eddin Sabri is an associate professor in the computer science Department at The University of Jordan. He obtained his B.Sc. degree in Computer Science from the Applied Science University, Jordan in 2001. He received M.Sc. degree in Computer Science from The University of Jordan in 2004 and a Ph.D. degree in software engineering from McMaster University, Ontario Canada in 2010. His main research interest is the formal analysis of security properties