UDC 004.9 **doi: 10.20998/2522-9052.2021.1.17**

Galina Cherneva[1,2], Pavlo Khalimov[3]

[1] South West University, Blagoevgrad, Bulgaria
[2] "Todor Kableshkov" University of Transport, Sofia, Bulgaria
[3] National Technical University "Kharkov Polytechnic Institute", Kharkiv, Ukraine

MUTATION TESTING OF ACCESS CONTROL POLICIES

Abstract. One of the most important and integral components of modern computer security are access control systems. The objective of an access control system (ACS) is often described in terms of protecting system resources against inappropriate or unwanted user access. However, a large degree of sharing can interfere with the protection of resources, so a sufficiently detailed AC policy should allow selective exchange of information when, in its absence, sharing can be considered too risky in general. Erroneous configurations, faulty policies, as well as flaws in the implementation of software can lead to global insecurity. Identifying the differences between policy specifications and their intended functions is crucial because the correct implementation and enforcement of the policies of a particular application is based on the premise that the specifications of this policy are correct. As a result of the policy, the specifications presented by the models must undergo rigorous validation and legalization through systematic checks and tests to ensure that the specifications of the policies really correspond to the wishes of the creators. Verifying that access control policies and models are consistent is not a trivial and critical task. And one of the important aspects of such a check is a formal check for inconsistency and incompleteness of the model, and the security requirements of the policy, because the access control model and its implementation do not necessarily express policies that can also be hidden, embedded by mixing with direct access restrictions or another access control model.

Keywords: access control; access control system; mutation testing; access control testing; policy.

Introduction

The main purpose of an access control system is often described as: protecting system resources against inappropriate or unwanted user access. From a business point of view, the goal can also be described by the optimal exchange of information for users and the program. However, much of the sharing can interfere with resource protection. Thus, detailed access control policy should allow for the selective exchange of information when, in its absence, sharing may be considered excessively dangerous.

Proper implementation and usage of access control policies are based on the premise that policies and specifications are true without hidden or conflicting rules that cause leakage or block access to information objects. Access control policy specifications should be thoroughly reviewed and legalized through continuous testing to ensure that policy specifications do reflect policy maker trends. Verification and testing of access control policies are similar to general testing of application software in many respects, but there are also differences. to nominally and accurately capture the security requirements that an access control system must adhere to, in this case models should be created in order to bridge a large gap in the abstraction between policy and mechanism.

Thus, information for development and implementation, as well as unambiguous and exact expression are provided by the access control model.

1. Access Control Policies

Access Control Policies — are high-level requirements that define how and by whom access is managed, under what circumstances they can access specific information. While access control policies may be program-oriented and thus taken to attention by the program provider, policies may relate to user actions within an organizational unit or across organizational boundaries. For example, policies may relate to the use of resources within or between organizational units or may be based on factors requiring knowledge, competence, authority, commitment or conflict of interest. Such policies may cover several computing platforms and programs. It is impractical to create a list of common access management policies, as business objectives, risk tolerance, corporate culture and regulatory responsibilities that affect policy vary from enterprise to enterprise and even from one organizational unit to another.

There are several well-known access control policies that can be categorized: discretionary or non-discretionary. Typically, discretionary access control policies are associated with identifier-based access control, and non-discretionary access control policies are associated with rule-based controls (e.g., a Mandatory access control).[1].

2. Discretionary access control

Discretionary access control delegates a certain part of access control at the discretion of the owner of the object, or any other person authorized to control access to the object. For example, it is typically used to restrict another user's access to a file (it is the file owner that controls other users' access to this file). Only those users specified by the owner can have some combination of read, write, execute, and other file permissions. Policy of discretionary access control, usually very flexible and widely used in the commercial and public sectors. However, discretionary access control is known to be considered vulnerable for two reasons: First, granting read access is transient. For example, when user "A" gives user "B" read access to a file, nothing stops user "B" from copying the contents of user "A's" file. Copying performs to the object controlled by user "B". User "B" can now grant any

other user access to a copy of user file "A" behind the back of user "A". Also, discretionary access control policies are vulnerable to various Trojan attacks. Because programs inherit the identity of the user who calls them, for example user "B" can write a program for user "A" which at first look performs some useful functions and at the same time can destroy the contents of files which belongs to user "A". During investigation of a problem, the audit files will indicate that user "A" has destroyed his own files [2]. Thus, formally, the disadvantages of discretionary access control are as follows:

- Information can be copied from one object to another; so there is no real guarantee in the flow of information in the system.
- There are no restrictions on the use of information when the user receives it.
- Access rights to objects are determined by the owner of the object, not through a system-wide policy that reflects the security requirements of the organization.

3. Testing of discretionary access control

For now, there are several basic methods of testing for accessibility to architecture: black box testing, white box testing, and gray box testing. Black box testing — this testing evaluates the functionality of the model without knowledge of its internal work. Testing by the white box method, is opposite of the black box testing and tests the internal functionality [3]. Gray box testing is a test that is performed with general knowledge of the model, usually performed by the user or the entity that simulates the user. Black box testing methods (or behavioral testing) may not be sufficient to protect against some spontaneous actions that are built into the access control model.

The methods of white box testing include the following testing tools:

- Error guessing;
- Error seeding;
- Exhaustive testing.

Error guessing - is a testing method where tests are developed based on the experience of previous testing or on the knowledge and experience of the tester, which defects are typical for certain components of the model or for functional areas. This method can also be applied to black box testing. Often the method is used in conjunction with other testing tools. The main advantage of this method is that this method reveals the cons of the system or model that cannot be detected by formal testing tools, but the effectiveness of the method is directly proportional to the experience of the tester or the experience of past testing, and does not guarantee high coverage.

Error seeding - is the process of deliberately entering errors into a program to test whether test cases are able to record added errors. This technique aims to detect errors in order to determine the relationship between actual and artificial errors. Artificial errors are unknown errors, and actual errors are injection errors, so test cases are used to check for such faults. This is basically an evaluation technique that helps to determine the presence of real errors based on the number of sown errors found. The main disadvantage of this method is the need to remove errors from the model after testing, which in turn complicates the automation of model testing.

Exhaustive testing, also known as complete testing, it is the complete testing of all system components with all possible combinations of input and output data. This method has a high code coverage ratio, but requires significant computing resources and time. The more complex the system, the more resources are required. In practice, it is used to test simple components of the model [4]. To solve the problem of automation, resources, and high coverage, consider mutation testing of the white box method, which generates additional test inputs to cover policy-related objects. The following example shows a mutation check for safety requirements. This policy formalizes the university's access control model for assigning and obtaining grades. It has two subjects: faculty and student, as well as two resources: credit scores and exam grades, and three actions: assign, review, and obtain. For this example, we expect the following requirements:

S1 — There are no students who have the rights to assign exam grades.

S2 — All faculty members have rights to assign credit and exam grades.

S3 — There is no such combination of subjects that the user with these subjects has the right to obtain and assign a resource examination marks.

The first security requirement S1 is intuitive, as we certainly don't want students to assign grades. The second safety requirement S2 is to ensure that teachers can actually assign grades. S3 is an example of segregation of duties, as we do not want anyone to assign their grades. There is an obvious conflict of interest. The following is an example of an access control policy model. To make the example readable and concise, the Model is written as simple statements. Are there requirements that do not immediately meet these three requirements. Requirements are, in essence, a constraint on the division of responsibilities that controls any request from both teaching and student subjects. All three safety requirements are met in the model. The first step in validating a mutation is to create mutation models using a mutation operator that simply inverts the effect of each rule by changing the permission to prohibit or the permission to allow. The number of mutant models created by this operator is equal to the number of rules in the model. The example model has only two rules [5] (Fig. 1 – Fig. 3).

The second step of verification is to determine what safety requirements are contained in the original model and each model of the mutant. The mutation model is excluded at the safety requirement if the safety requirement is maintained for the original model, but not for the mutant model. In other words, the safety requirement reveals a defect inherent in the mutation model. The greater the number of mutant models excluded, the more the original model meets safety requirements.

```
1. if object = faculty
2. and resource = (credits or exam grades)
3. and action = (view або assign)
4. then
5. allow
6. if object = student
7. and resource = or exam grades
8. and action = obtain
9. then
10. allow
```

Fig. 1. Requirements of access control model

```
1. if object = faculty
2. and resource = (credits or exam grades)
3. and action = (view або assign)
4. then
5. prohibit
6. if object = student
7. and resource = or exam grades
8. and action = obtain
9. then
10. allow
```

Fig. 2. First mutant of model

```
1. if object = faculty
2. and resource = (credits or exam grades)
3. and action = (view або assign)
4. then
5. allow
6. if object = student
7. and resource = or exam grades
8. and action = obtain
9. then
10. prohibit
```

Fig. 3. Second mutant of model

The first mutation model in Figure 2 does not meet the requirement of S2, and thus the first model is excluded. Requirement S2 seeks to provide a policy under which all teachers can assign grades. Because the error in the model is the rule that provides such access, the security requirement is violated. The second model of the mutant in Figure 3 is not excluded by any of the three safety requirements, showing that the original model is not comprehensive and does not fully meet the safety requirements. Mutation testing serves a safety requirement for two purposes: to determine whether the model fully covers the safety requirements to facilitate changes to the model so that it covers all safety requirements or vice versa. In some cases, instead of correcting the model, mutation testing serves to correct the security requirement by correcting the policy:

S4 — All students can obtain exam grades.

4. Discussion of results

Summarizing, Fig. 4 illustrates the required inputs and outputs of the mutation test. The input data is a tested model and, in this case, one mutation operator. The mutator then generates a set of mutant models, each of which has one error. The mutation operator generates a mutant for each rule, denying the decision of this rule. Although black box testing is relatively quick for mutants, large models can use thousands of mutant models to easily generate them. An equivalent mutant is a mutant that is syntactically different from the original model, being semantically equivalent.

In other words, an equivalent mutant will give the same result as the original model for all input data and thus does not give any benefit and result in artificially reducing the killing rate of mutants, which gives low quality and inaccurate measurement quality. It is also possible to determine which safety requirements are met and which do not correspond to both the original model and each model of the mutant [6].

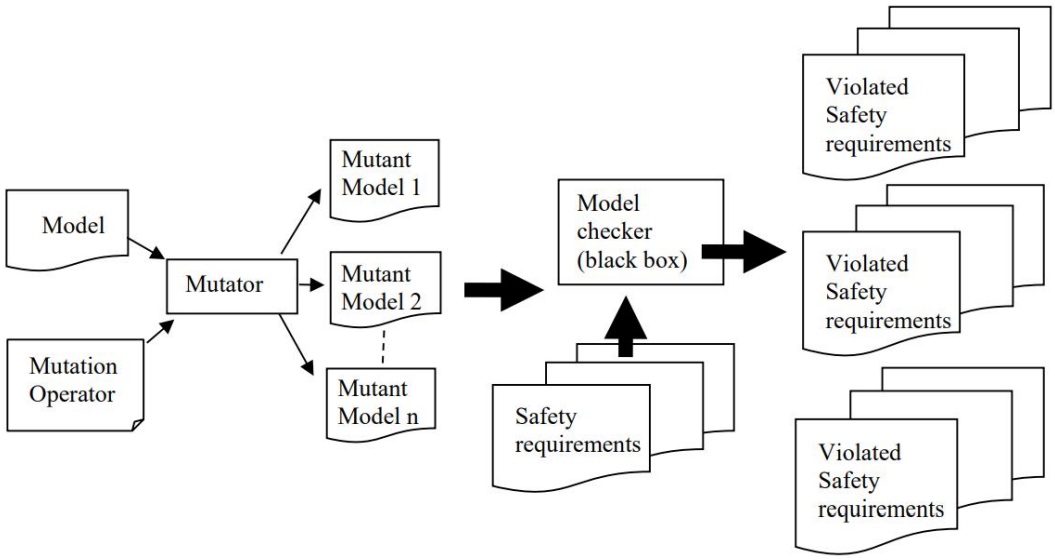

Fig. 4. Mutant models generation

An important step in mutation testing is to calculate the exclusion factor for mutation models.

The model exclusion factor is the ratio of the number of excluded models to the total number of

mutation models. This ratio serves as a metric to quantify the coverage of the safety requirements of the model. The high exclusion factor indicates that the original model covers a large number of safety requirements. For example, the coverage of the models (i.e. the model exclusion factor) for the security requirements S1, S2 and S3 in the above example is 50%, because only one of the two models is excluded. If you add S4, then the exclusion factor will increase to 100%.

There are trace files, which were previously generated by security requirements verification, are analyzed in order to divide the security requirement into four subsets for each mutant. The Venn diagram is illustrated in Fig. 5, which describes the relationship of these four sets for one mutant model.

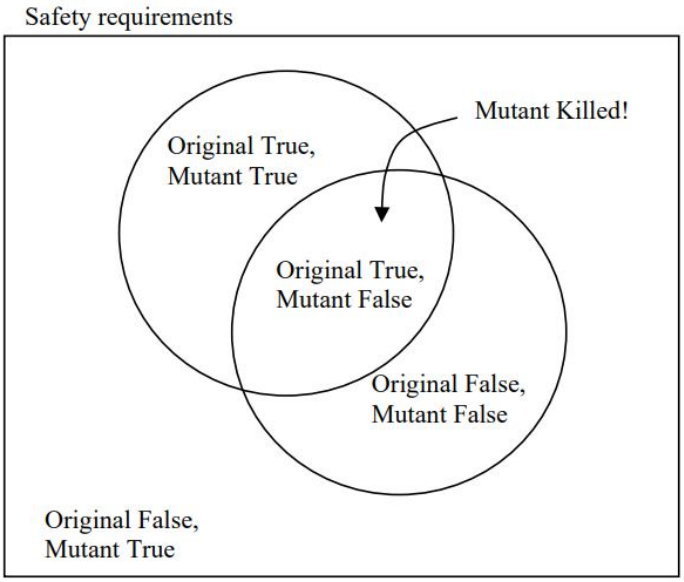

Fig. 5. Venn diagram illustrating the four safety requirement states

The area inside the box presents a set of all safety requirements. The area inside the left circle is a set of safety requirements that are true for the original model.

Thus, the area outside the smallest left circle and inside the field is a set of security requirements that do not correspond to the reality for the original model (i.e. these security requirements do not meet the requirements of the original model).

The area inside and the right circle represent a mutant model that is considered erroneous for safety requirements. Therefore, the area outside the right side of the circle and inside the box is a set of safety requirements that are true for the mutant model. An interesting area is the intersection between the two circles. If at least one safety requirement adheres to the original model, but it is not met fairly for the mutant model, then the mutant is killed. If two circles do not intersect, the mutant is not killed [7].

A safety requirement that is true for both the original model and the mutant model is irrelevant to the detection of a fault in the mutant model because the safety requirement does not apply to the part of the model that contains the defect. The security requirement that is considered erroneous for the original model does not matter, as it is unclear whether this erroneous security requirement is caused by an error in the model or the security requirement itself.

More specifically, before performing a mutation testing, these safety requirements must be checked manually to determine whether they are erroneous due to a model error, a safety requirement error, or an error in environmental constraints.

Conclusion

Therefore, mutation testing, in contrast to exhaustive testing, can be implemented for any complexity with a sufficiently high code coverage ratio.

Since additional mutant models are created based on the base model, and all tests are performed on the mutants, this guarantees the integrity of the base model, and also provides an opportunity to automate testing.

Thus, the mutation testing method can be used to test access control policies.

References

1. Hu, V.C., Ferraiolo, D.F. and Kuhn, D.R. (2006), *Assessment of Access Control Systems, NIST Interagency Report 7316*, National Institute of Standards and Technology, Gaithersburg, Maryland, DOI: https://doi.org/10.6028/NIST.IR.7316.
2. Muhammad, Aqib and Riaz Ahmed, Shaikh (2015), "Analysis and Comparison of Access Control Policies Validation Mechanisms", *International Journal of Computer Network and Information Security (IJCNIS)*, Vol. 1, pp. 54-69. DOI: https://doi.org/10.5815/ijcnis.2015.01.08.
3. Hu, V.C., Kuhn, D.R. and Xie, T. (2008), "Property Verification for Generic Access Control Models", *Proceeding of The 2008 IEEE/IFIP International Symposium on Trust, Security and Privacy for Pervasive Application (TSP2008)*, Shanghai, China, December 17-20, DOI: https://doi.org/10.1109/EUC.2008.22.

4. Brian, L.K., Labish, Yu. and Shusha, M. (2005), “Stress Testing of Real Time Systems with Genetic Algorithms,”, *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computing*, pp. 1021-1028, Washington, DC, USA.
5. Hu, Vincent C., Kuhn, Rick and Yaga, Dylan (2017), *Verification and Test Methods for Access Control Policies/Models*, NIST Special Publication 800-192, DOI: https://doi.org/10.6028/NIST.SP.800-192.
6. Martin, E. and Xie, T. (2007), “A Fault Model and Mutation Testing of Access Control Policies”, *Proceedings of the 16th International Conference on World Wide Web (WWW 2007), Security, Privacy, Reliability, and Ethics Track*, Banff, Alberta, Canada, pp. 667-676, DOI: https://doi.org/10.1145/1242572.1242663.
7. Jia, Yu. and Harman, M. (2011), “Analysis and Investigation of the Development of Mutation Testing”, *IEEE Transactions on Software Engineering*, vol. 37, issue 6, pp. 649–678.

Надійшла (received) 30.11.2020
Прийнята до друку (accepted for publication) 10.02.2021

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Чернева Галіна Петкова – доктор наук (комунікаційні технології), професор, Транспортний університет “Тодор Каблешков”, Софія, Південно-Західний Університет, Благоєвград, Болгарія;
Galina Cherneva – Doctor of Sciences, (Communication Technology), Professor, “Todor Kableshkov” University of Transport, Sofia, South West Universityб Blagoevgrad. Bulgaria;
e-mail: cherneva@vtu.bg, gcherneva@swu.bg; ORCID ID: http://orcid.org/0000-0001-7441-0270.

Халімов Павло Вікторович – аспірант кафедри обчислювальної техніки та програмування, Національний технічний університет “Харківський політехнічний інститут”, Харків, Україна;
Pavlo Khalimov – Postgraduate student Professor of Computer Science and Programming Department, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine;
e-mail: pavlo_khalimov@gmail.com; ORCID ID: http://orcid.org/0000- 0003-0254-5015.

Мутаційне тестування політик керування доступом

Г. П. Чернева, П. В. Халімов

Анотація: Одними з найскладніших, важливих та невід'ємних складових сучасної комп'ютерної безпеки є системи керування доступу. Завдання системи керування доступом часто описується з точки зору захисту ресурсів системи проти невідповідного або небажаного доступу користувача. Проте, велика ступінь спільного використання може перешкодити захисту ресурсів, таким чином, досить детальна політика керування доступом повинна дозволяти виборчий обмін інформацією, коли в його відсутність, спільне використання може вважатися занадто ризикованим в цілому. Помилкові конфігурації, несправні політики, а так само недоліки в реалізації програмного забезпечення можуть призвести до глобальної незахищеності. Виявлення відмінностей між специфікаціями політик і їх передбачуваними функціями має вирішальне значення, тому що правильна реалізація і забезпечення дотримання політик конкретного додатка засновані на передумові, що специфікації даної політики є правильними. В результаті політики специфікації, представлені моделями, повинні пройти сувору перевірку і легалізацію шляхом систематичних перевірок і тестувань, щоб гарантувати, що специфікації політик дійсно відповідають бажанням творців. Перевірка відповідності політик і моделей контролю доступу нетривіальна і критична завдання. І одним з важливих аспектів такої перевірки є формальна перевірка на непослідовність і неповноту у моделі, і вимоги безпеки політики, тому що модель контролю доступу та її реалізація не обов'язково явно виражають політики, які також можуть бути приховані, вбудовані шляхом змішування з обмеженнями прямого доступу або іншого доступу моделі управління.

Ключові слова: керування доступом; системи контролю доступу; мутаційне тестування; тестування систем керування доступом; політики.

Мутационное тестирование политик управления доступом

Г. П. Чернева, П. В. Халимов

Аннотация: Одними из самых сложных, важных и неотъемлемых составляющих современной компьютерной безопасности являются системы управления доступа. Задача системы управления доступом часто описывается с точки зрения защиты ресурсов системы против несоответствующего или нежелательного доступа пользователя. Однако, большая степень совместного использования может помешать защиты ресурсов, таким образом, довольно подробная политика управления доступом должна позволять избирательный обмен информацией, когда в его отсутствие, совместное использование может считаться слишком рискованным в целом. Ложные конфигурации, неисправные политики, а так же недостатки в реализации программного обеспечения могут привести к глобальной незащищенности. Выявление различий между спецификациями политик и их предполагаемыми функциями имеет решающее значение, так как правильная реализация и обеспечение соблюдения политик конкретного приложения основаны на предпосылке, что спецификации данной политики являются правильными. В результате политики спецификации, представлены моделями, должны пройти строгую проверку и легализации путем систематических проверок и тестирований, чтобы гарантировать, что спецификации политик действительно соответствуют желанию создателей. Проверка соответствия политик и моделей контроля доступа нетривиальная и критическая задача. И одним из важных аспектов такой проверки является формальная проверка на непоследовательность и неполноту в модели, и требования безопасности политики, так как модель контроля доступа и ее реализация не обязательно явно выражают политики, которые также могут быть скрыты, встроенные путем смешивания с ограничениями прямого доступа или иного доступа модели управления.

Ключевые слова: управление доступом; системы контроля доступа; мутационное тестирование; тестирование систем управления доступом; политики.