On the Compliance of Access Control Policies in Web Applications

Thanh-Nhan Luong[1,2(✉)], Dinh-Hieu Vo[1], Van-Khanh To[1], and Ninh-Thuan Truong[1]

[1] VNU University of Engineering and Technology, 144 Xuan Thuy, Cau Giay, Hanoi, Vietnam
ltnhan@hpmu.edu.vn, {hieuvd,khanhtv,thuantn}@vnu.edu.vn
[2] Department of Informatics, Hai Phong University of Medicine and Pharmacy, 72A Nguyen Binh Khiem, Ngo Quyen, Hai Phong, Vietnam

Abstract. Model-View-Controller (MVC) architecture has commonly used in the implementation of web applications. These systems often incorporate security policies to ensure their reliability. Role-based access control (RBAC) is one of the effective solutions for reducing resources access violations of a system. This paper introduces an approach to check the compliance of a web application under MVC architecture with its RBAC specification. By investigating the system architecture and source code analysis, our approach conducts with extracting a list of resources access permissions, constructing a resources exploitation graph and organizing an access control matrix according to roles of a web application. The approach aims at checking two violation cases of web applications: (i) the presence of unspecified access rules and (ii) the absence of specified access rules. We illustrate the proposed approach by a case study of web based medical records management system.

Keywords: Compliance · Access control policy · RBAC
Web applications

1 Introduction

Web applications (WAs) which are designed according to Model-View-Controller (MVC) architecture [15] have been using widely in many fields of the social life such as training, e-commerce, healthcare, etc. Besides, these applications are almost executed in the internet environment and their data is transported via difference line types so they include many implicit security risks such as lost data, leak information, refuse users' service requests, or authorize for users incorrectly [10,16]. These problems can cause damage to system resources and users. Therefore, the applications need adequate security policies and mechanisms to ensure the interests of participants.

The confidentiality, integrity, availability, accountability, and non-repudiation are basic security properties of a secure software [12]. However, each type application demands some different properties. In practice, many techniques and

P. Cong Vinh and V. Alagar (Eds.): ICCASA 2018/ICTCC 2018, LNICST 266, pp. 58–69, 2019.
https://doi.org/10.1007/978-3-030-06152-4_6

mechanisms have deployed to guarantee these security properties. Role-based access control (RBAC) [9] is one of the solutions to guard the confidentiality and integrity of software systems. The implementing RBAC within the WAs helps to assign the access rights to users through their roles. This can reduce resources access violations for WAs.

WAs are increasingly complex and programming is prone to errors. In addition, the programmers may not be designers so they may not completely understand about security requirements. Therefore, the application may not be conformed to specified requirements in its model. Furthermore, the cost to repair flaws and overcome the consequence in the maintain stage is much higher than it in the design phase. Checking the compliance of access control policies may detect flaws, reduce costs and increase the reliability of software systems.

Checking the consistency between RBAC policy and its implementation have been explored by some researches [1–3]. However, these studies have not been considered checking of compliance between web applications under MVC architecture and RBAC policies by the static code analysis technique. Our work introduces an approach to deal with this issue. The contribution of this paper includes:

- Firstly, we introduce steps to build the access control matrix according to roles from the source code of WA by the static analysis technique.
- Next, we propose a verification algorithm which can detect two violation cases of WAs: (i) the presence of unspecified access rules and (ii) the absence of specified access rules.
- Lastly, we illustrate proposed approach with a web based medical records management system.

The rest of the paper is organized as follows. Section 2 presents some basic knowledge about RBAC model, MVC architecture. In the next section, we discuss related studies. Section 4 presents a small WA in medical records management system. Our proposed approach is described detail in Sect. 5. We draw some conclusions and future work in the last section.

2 Background

The RBAC model and MVC architecture are the background knowledge which is used in our approach. In this section, we briefly describe them.

2.1 Role-Based Access Control

The RBAC model [8,9] is depicted in Fig. 1. The *Users*, *Roles*, *Permissions*, *Objects* and *Operations* are five main elements of this model. A user is referred to as the agent that interacts with the system. Users do not perform actions directly but through their roles. The role which is the central component of an RBAC model, represents for a job position in an organization. The *Permissions*

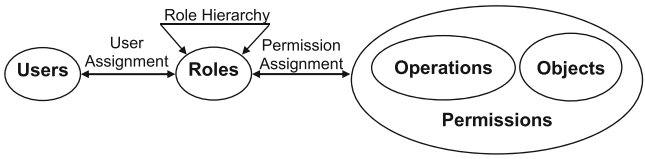

Fig. 1. Role-based access control.

includes *Operations* and *Objects*, this means that when users gain a permission, they are allowed to execute an operation on an object.

The principle of least privilege and separation of duty help users in the RBAC model only has sufficiently roles and permissions to carry out their duties. This can prevent attackers from accessing system resources. The $UserAssignment$, $PermissionAssignment$ and $RoleHierarchy$ are basic relationships of RBAC model. Specifically, a user can be assigned to many roles and a role can have several users. Each role is assigned with some permissions, and defined depending on the regulations of each organization. The inheritances between roles are described in $RoleHierarchy$.

2.2 Model-View-Controller Architecture in JavaEE

Model-View-Controller [13,15] is briefly called MVC which has been being a popular software architecture pattern for building web applications with many programming languages like Java, C#, Ruby, PHP, etc. Applications which are designed under MVC architecture aimed at code reuse and parallel development efficiently. Figure 2 depicts the MVC architecture in JavaEE.

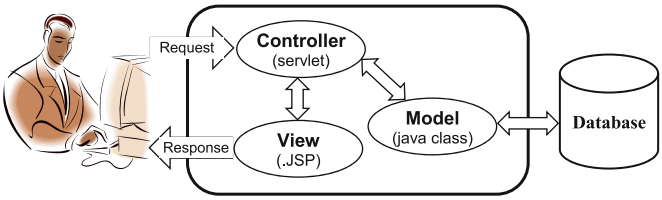

Fig. 2. The MVC architecture in JavaEE.

- *Model* is a component of the system that performs requests related to databases from *Controller*. It is the classes $*.java$ which include methods to connect to databases and interact with data sources.
- *View* includes codes such as $*.jsp$ to display user's graphical interfaces. It regulates the displayed data formula and communicates with *Controller*. In addition, it also supplies the way to gather data from the input.

– *Controller* creates the synchronization between *Model* and *View* by analyzing the requests. It interacts with *Model* and gets the data to create *View*. In other words, *Controller* is responsible for processing the events that are activated by user's interactions with the application or between processes of the system.

3 Related Work

Several papers have previously conducted for verifying web applications' access policies. In this section, we summarize some studies similar to our work.

The research works [1–3] recovered RBAC security model of a dynamic web application for verifying and testing of its security properties. Their approach builds a Prolog-based formal model from UML-based security model of the application. They illustrate proposed approach for analyzing, testing, maintenance, and re-engineering of the web application security. In the testing scenario for unauthorized access, the authors check if a guest user can using the links that an administrator can see. This study is similar to our research about user access, but their problem relates to the testing for unauthorized access. They use both of static and dynamic analysis to recovered SecureUML of WA. However, we only use static analysis on the source code of a WA under MVC architecture.

The paper [11] proposes a tool called SeWAT which is used to model MVC web applications graphically. In addition, the authors implement and validate role-based access control policy with an example of patient management system. The experiments with many realistic applications prove that their technique is useful for checking and deploying access policies of web applications. However, this study did not conduct with source code of WAs.

By using model checking in the design phase, Eun-Hye Choi and Hiroshi Watanabe introduced an approach to verify class specifications of WAs [5]. The authors modeled behavior of WA from its class diagram and method specifications. They proposed two aspects to verify the consistency: (i) between a class specification and a page flow diagram, (ii) between a class specification and a behavior diagram. The approach applied with real specifications of a certain company's developed WA and found several faults of the specifications which had not been detected in actual reviews. Graph-based modeling methods are presented by Di Sciascio *et al.* [6,7] and Castelluccia *et al.* [4]. The authors used model checking techniques to verify the UML design of a WA automatically. They introduced a tool called WAVer [4] which uses Symbolic Model Checking techniques to verify WA designs. The verification is conducted with three main stages: modeling WA in form of Finite State Machine (FSM), formalization of application correctness with CTL formal language, using NuSMV model checker to find a violation of specifications. These studies are only performed in the design phase.

In above presented studies, the works [1–3] analyzed the resources access policy from web application's source code. However, the authors use it for testing. The remained studies have not conducted with source code. In our study, we use static analysis technique to check the conformance of a WA under MVC pattern with its RBAC specification.

4 A Case Study

In a medical records management system, the security of patients' information are compulsory task according to regulations in HIPAA (USA's Health Care Insurance Portability and Accountability Act). Medical records must be confidentially kept because they contain private information and they only are used for monitoring and treating diseases or other tasks in regulated law. Medical records have had many risks [14] so protecting them from unauthorized disclosure has been attending.

The process of processing patient records in medical organizations or hospitals is usually attended by receptionist, physician, patient, etc. who have some privileges to do their tasks in the medical records management system. Medical records of patients needs to be secure and it can only read by authorized persons. Table 1 specifies access policy of system.

Table 1. RBAC policy to medical record.

Users	**Roles**
Ann	Receptionist
Bob	Doctor
Tom	Patient

(a) User-Role assignments.

Roles	**Permissions**
Receptionist	CreateMedicalRecord
Doctor	ReadMedicalRecord UpdateMedicalRecord
Patient	ReadMedicalRecord

(b) Role-Permission assignments

Permissions	**Operation**	**Object**
CreateMedicalRecord	Create	MedicalRecord
ReadMedicalRecord	Read	MedicalRecord
UpdateMedicalRecord	Update	MedicalRecord

(c) Permission mapping.

In our case study, the implemented medical records management system is written by J2EE and designed following to the MVC architecture. Medical records are stored in table *MedicalRecord* of the database management system *MySQL*. Figure 3 describes the components *Model*, *View*, *Controller*, and roles within our application. Where users must log into the system by their accounts and valid users are assigned some permissions to interact with resources corresponding to their roles in the system. In our example WA, a user assigned role *Doctor* can read, update and delete his/her patients' medical records. Users of role *Patient*, they can only read their medical records. If users are assigned role *Receptionist*, they can create new medical records. Pages *Doctor.jsp*, *Patient.jsp* and *Receptionist.jsp* help users perform their jobs corresponding to their roles in the organization. We aim at checking the web application's access policy against its specification.

Fig. 3. The presence of RBAC and MVC architecture in the medical records management system.

5 Our Approach

We use static analysis technique to check the conformance of WAs with specified RBAC policies. Our approach overview is described in Fig. 4. Generally, we use the library *JDT* in JavaEE to parse each file $*.java$ into an abstract syntax tree (AST). From analyzed ASTs, we extracted the necessary information to serve for each stage of the verification process.

5.1 List of Permissions

When users perform their functions within an application, the system has to invokes methods that interact with resources. Firstly, we need to determine the name list of system resources because software application often has may data but some of them are critical resources need to be protected. Next, we specify classes in *Model* that include resources interaction methods. These methods encompass SQL operations (*Select*, *Insert*, *Delete*, or *Update*) corresponding to action types (*Read*, *Create*, *Delete*, or *Update*) of users. Lastly, we use the static analysis technique to extract these methods into a list of permissions $L = \{\langle mp, ac, rs\rangle\}$. Where, the mp includes class name, return type of method, method name, data type list of parameters; ac is an action type in the action type set AC; and rs is an element of the protected resources set RS.

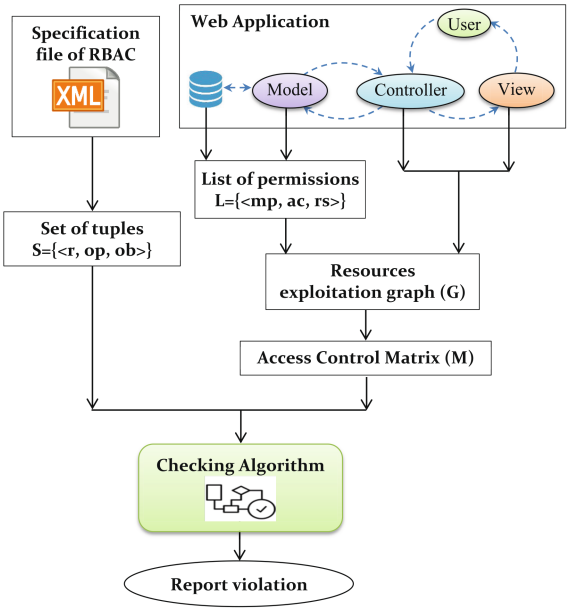

Fig. 4. The approach overview.

5.2 Resources Exploitation Graph of WA

The component *View* is the set of pages and the *Controller* is used to navigate transitions between pages. We can construct a direct graph which is a resources exploitation graph G of the WA. The information of each vertex includes name of each page in the component *View* and list of resources exploitation calls. Our graph is built by following steps:

Step 1. Building a diagram of pages according to the following rules:

- Each node of graph is a page in the *View*.
- Edge goes from node A to node B (A $\neq$ B) iff page A:
  - has a link to page B or
  - is redirected to page B or
  - includes page B.

Step 2. By analyzing source code of the component *Controller* and *View* of each page, we can gather all method calls from it and are also presented in the list of permissions L. Next we attach them on the information of graph's vertexes. The resources exploitation graph of the example of medical record management system is depicted in Fig. 5.

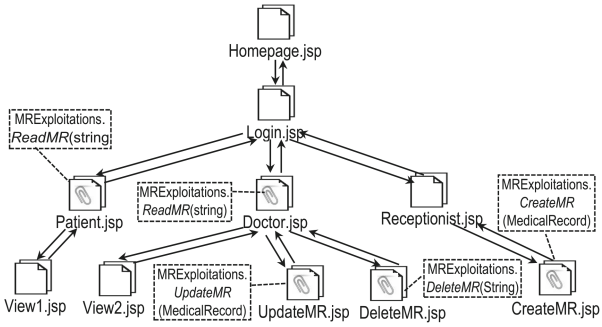

Fig. 5. The resources exploitation graph of the medical record management system.

5.3 Access Control Matrix

We use a three-dimensional matrix to describe actions on objects of roles within the WAs. It is constructed as follows:

- The first dimension of the matrix is $ROLE$. It contains the set of roles within WA ($R = \{R_1, R_2, .., R_n\}$) that can be taken from analyzing file $.java$ of controller that handles login page. Each user logs in to the system through an account ($username$, $password$). If the login is success then valid users are redirected to the pages corresponding to their roles.
- The second dimension is $ACTION$ which includes all executed atom actions to exploit resources ($AC = \{AC_1, AC_2, .., AC_m\}$).
- The third dimension is $RESOURCES$ which includes list of resources need to be protected ($RS = \{RS_1, RS_2, .., RS_p\}$).

Suppose that, WA includes n roles, m action types, and p resources. The value of an element $M[i][j][k]$ is 'Yes' if role R_i has permission to perform action AC_j on resources RS_k with $i = 1..n, j = 1..m, k = 1..p$. Users are controlled when they log into system to grant rights according to their roles. By visiting sub-graphs from vertexes corresponding to roles, we can gather all action types which are executed on resources of roles. Algorithm 1 converts the resources exploitation graph to access control matrix and Fig. 6 is the access control matrix of roles in the medical records management system.

5.4 Algorithm for Checking Compliance

The inputs of checking algorithm are S, M, R, AC, and RS, where M is the access control matrix of WA; S is the system resources access policy that is extracted through analyzing *.XML file structure; R, AC, RS are the set of roles, actions, and resources within application respectively. Each tuple $s = \{\langle r, op, ob\rangle\} \in S$ means that role r is allowed to perform operation op on object ob and S is extracted from RBAC policy.

Algorithm 1. Convert the graph G to the matrix. M

Input : **G**: the resources exploitation graph.
Output: **M**: the access control matrix.
Data : r, v: the element in the set of roles R, vertexes V, respectively.

```
Procedure Convert(G, M)
begin
    foreach r ∈ R do
        UnmarkAll(G);
        v ← GetStartVertex(r);
        Write(G, v, M);

Procedure Write(G, v, M)
begin
    if (¬Mark(v)) then
        if (v.attach ≠ ∅) then
            M ← Set(v.attach);
        Mark(v) ← true;
        V ← Adjacent(v);
        if (V ≠ ∅) then
            foreach v ∈ V do
                Write(G, v, M);
```

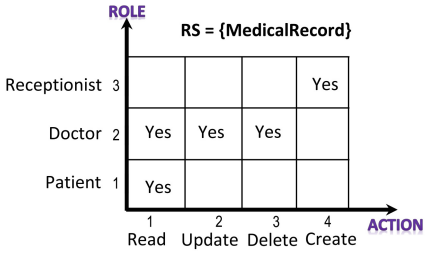

Fig. 6. The access control matrix of the medical records management system.

The proposed checking algorithm can detect two violation cases of WAs: the presence of unspecified access rules and the absence of specified access rules. The first case ⓘ shows that there is at least one role executed a permission which is not specified in the access policy of WA. The second case ⓘⓘ shows that specified permissions in the WA's access policy are not implemented inadequately. In two violation cases, the former can lead to leakage resources and it is usually more interested. The later is contradictory with the former but it helps to detect the role's permission shortcoming compared to its specification.

Algorithm 2. Checking compliance.

Input : **S**: the resources access policy
M: the access control matrix of WA
R, AC, RS: the set of roles, actions, and resources within application respectively
Output: Compliance result
Data : r, ac, rs: the element in the set of roles, actions, resources, respectively.

```
Function isConformance(S, M, R, AC, RS)
begin
    foreach r_i ∈ R do
        foreach ac_j ∈ AC do
            foreach rs_k ∈ RS do
                if M[i][j][k] =' Yes' then
                    m ← {⟨r_i, ac_j, rs_k⟩};
                    if m ∉ S then
                        return false; ①
                    else
                        S ← S \ {m};
    if (S ≠ ∅) then
        return false; ⑪
    else
        return true;
```

6 Conclusion and Future Work

Ensuring the security of web systems is a very complex issue. It usually requires the combination of many mechanisms and techniques. Therefore, verification of security policy in web systems is an attractive topic by researchers. In this paper, we have presented an approach to analyze source code of WAs into access control matrix and compare it to their specified policy. The example WA of medical records management system is deployed in the J2EE environment, so the proposed approach use JDT library to analyze source code into AST tree. From those, we extract *Model* into a list of permissions, build resources exploitation graph of WA, and construct access control matrix according to each role within the system. The proposed algorithm can be used to detect two specification violation types: a role executed at least one permission not be presented in access policy; at least one permission specified in access policy not be implemented in the application.

Currently, our case study does not cover all cases of access control policies. In the future, we are planning to investigate the approach with some larger applications in domains such as healthcare system, bank system, etc. In addition,

we intend to improve our approach for checking multi-level control and other security properties of WAs. A support tool is developing to check automatically the compliance of access control policy in web applications.

Acknowledgments. This work has been supported by VNU University of Engineering and Technology under Project QG.16.32.

References

1. Alalfi, M.H., Cordy, J.R., Dean, T.R.: A verification framework for access control in dynamic web applications. In: Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering, pp. 109–113. ACM (2009)
2. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Automated verification of role-based access control security models recovered from dynamic web applications. In: 2012 14th IEEE International Symposium on Web Systems Evolution (WSE), pp. 1–10. IEEE (2012)
3. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Recovering role-based access control security models from dynamic web applications. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 121–136. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31753-8_9
4. Castelluccia, D., Mongiello, M., Ruta, M., Totaro, R.: WAVer: a model checking-based tool to verify web application design. Electron. Notes Theor. Comput. Sci. **157**(1), 61–76 (2006)
5. Choi, E.H., Watanabe, H.: Model checking class specifications for web applications. In: 12th Asia-Pacific Software Engineering Conference, APSEC 2005, p. 9. IEEE (2005)
6. Di Sciascio, E., Donini, F.M., Mongiello, M., Piscitelli, G.: AnWeb: a system for automatic support to web application verification. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, pp. 609–616. ACM (2002)
7. Di Sciascio, E., Donini, F.M., Mongiello, M., Totaro, R., Castelluccia, D.: Design verification of web applications using symbolic model checking. In: Lowe, D., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 69–74. Springer, Heidelberg (2005). https://doi.org/10.1007/11531371_12
8. Ferraiolo, D., Kuhn, D.R., Chandramouli, R.: Role-Based Access Control. Artech House, Norwood (2003)
9. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. (TISSEC) **4**(3), 224–274 (2001)
10. Garg, A., Singh, S.: A review on web application security vulnerabilities. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **3**, 222–226 (2013)
11. Idani, A.: Model driven secure web applications: the SeWAT platform. In: Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems, p. 3. ACM (2017)
12. Mead, N.R., Allen, J.H., Barnum, S., Ellison, R.J., McGraw, G.: Software Security Engineering: A Guide for Project Managers. Addison-Wesley Professional, Boston (2004)
13. Principe, M., Yoon, D.: A web application using MVC framework. In: Proceedings of the International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government (EEE), p. 10 (2015)

14. Rubenstein, S.: Are your medical records at risk? Wall Street J. (2009)
15. Shklar, L., Rosen, R.: Web Application Architecture. Wiley, Hoboken (2009)
16. Touseef, P., Ashraf, M.A., Rafiq, A.: Analysis of risks against web applications in MVC. NFC IEFR J. Eng. Sci. Res. **5** (2017)