Alternate *ACM* SIG Proceedings Paper in LaTeX Format

Michal Trnka
Dept. of Computer Science and Engineering
Czech Technical University
Technická 2, 166 27 Praha, Czech Republic
trnkami1@fel.cvut.cz

Tomas Cerny
Dept. of Computer Science and Engineering
Czech Technical University
Technická 2, 166 27 Praha, Czech Republic
tomas.cerny@fel.cvut.cz

ABSTRACT

Security of software applications is very a challenging and extensive topic. To keep up with the trend of personalized context aware applications the security design must adapt to it. This paper presents context awareness into the role based access control. It will describe already existing solutions, point out their key ideas and propose our RBAC lightweight extension. It is universal and allows instant enhancement of current RBAC even in current applications. The proposed solution is based on security levels which are assigned to users based on context. Security levels represent how the users can be trusted and they are determined during the login procedure. The levels are used as additional security constraints to access resources. In application, the user needs to possesses not only the right permission granted through RBAC roles, but also have a corresponding level.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: Protection mechanisms

General Terms

Design, Security, Theory

Keywords

Role-based access control, Context-aware security, Security levels

1. INTRODUCTION

Contemporary applications move toward context-awareness (CA) [1, 11]. It is caused by emerge of the amount of the mobile technologies [5], as well as by the users demand for personalized applications. Applications provide personalized content based on the user's or the application's context [7]. This results in a completely new experience for the application's operators and users. However, the security design fails to consider context-awareness. Usually, users are assigned various roles in applications or permissions for resources and security rules are independent of the context. There is only a few applications, which has security based on context (e.g. Google provides content based on history and Facebook required additional authentication if logged from unusual place). We can expect that users and application owners would take advantage of application security that is based on context to provide specific resource access based on context.

Applications using Context-Aware Security (CAS) can be much less obtrusive for users. They can be asked for different authentication methods based on context. They can also be authorized for the same resource in various ways depending on their context. For example, access from Prague can differ from access rights required from Brno. They can even sometimes omit authentication because their context is trustworthy by itself (e.g. access from inner company network). Similar to users, also application operators can profit from the context-based authentication. They might define stricter security rules for suspicious users' behavior (e.g. Internet access to system confidential resources at night). Using context allows system administrators to define finer-grained security rules that would be otherwise tangle through multiple rules and make them unsustainable for maintenance.

Application operators and software developers are well aware of the added value of CAS; however, none of the existing proposals on its use are widely used. [3, 4, 8, 9, 12, 13, 14, 15, 16, 17, 20, 23]. They are either too complex to use or they are too innovative, requiring complete system redesign that many find too difficult to incorporate into existing solutions.

This paper presents one possible solution that will extend the standard RBAC security architecture with CA elements. This extension bases on users security levels that consider their context. Accessing application resources requires that the user posses particular level in addition to his/her usual access rights. It will allow an extension to various current security architectures with CA elements.

2. BACKGROUND

Large applications and information systems need some form of authentication and authorization. Existing systems have been available for many decades and are almost as old as computers. As and example, the Memex information system was proposed as early as 1945[2]. Concerns regarding application security have existed for many decades and have been addressed in various ways.

Two of the oldest principles for securing application re-

RACS'15 October 9-12, 2015, Prague, Czech Republic.

sources are Mandatory Access Control (MAC) [18] and Discretionary Access Control (DAC) [18]. These two principles do not explain how application security should be implemented, but rather define core principles in its authorization. In MAC, there exists an authority that has the responsibility to grant permissions to access all resources. On the contrary, in DAC, the permission can be granted by anyone with sufficient permission for the resource.

However, granting permissions to every user in a system is unpractical when numerous users are involved. Role-based access control (RBAC) [6] provides another level of abstraction. It has the approach that permission is given to an abstract role and users are assigned these roles. Usually, there exists fewer roles than permissions in the system. Unlikely users, the roles do not change significantly over time.

Nevertheless, these authorization principles and methods are static. They do not reflect the changing state of the systems and users. Once they are set, they do not take in account any other factors. Fine tuning becomes difficult.

CAS can overcome these difficulties and even provide new experience for users and application operators. CA applications are much more personalized then the static ones and the same comes for the security. The application can get a lot of information about users from the context and therefore does not require that the users provide additional information. For example, if an application has knowledge of the user's usual IP address, then connections from another IP can be viewed as suspicious. The application context is also a valuable source of information for application owners. The owner could restrict a certain range of IP addresses, times of the day, etc., to access resources in the application.

With emerge of CA applications, and inherent need for CAS arises. The idea of CA applications has existed since 90's [19]. Such CA applications adapts according to the user's location and access times, the collection of nearby people, hosts, accessible devices, etc., as well as to changes to such variables over time. A system with these capabilities can examine the user's environment, compute it and then react to changes to the environment. Naturally, such applications require a security architecture that incorporates all of the above-mentioned principles to be fully CA. However there is a significant lack of standardized methods or best practices how to address the CAS.

To illustrate how can CAS improve applications consider the following example. Consider an information system in use within a company. Users in the inner company network are allowed to access noncritical resources. When a user comes from the Internet to access sensitive resources, he/she is required to authenticate. Not only would the users benefit from this procedure, but CAS can determine any suspicious behavior on the part of the user. To cite an example, should a user log into the system within a short time frame from different parts of the world, a flag could be raised in order to report the incident for further investigation. Furthermore, the company can set access hours for various resources, such as orders, to limit the possibility of their abuse (e.g., restrict access in non-working hours, to plan the next day delivery).

3. RELATED WORK

Multiple attempts have been made to extend classic RBAC with CA elements as well as to make RBAC more fine-grained.

One of the approaches is to add another set of roles to RBAC. Moyer et. al. [14] proposes creating two additional sets of object roles and environmental roles and tying permissions with trio of roles. Covington et. al. [4] simplifies that by confining it to just one additional set of environmental roles. They are hierarchical composed and represent the current state of system. A similar approach by Seon-Ho et. al. [16] suggests an additional set of context roles; however creating completely new set (or sets) of roles increases design and computational complexity and therefore reduces the advantages of RBAC.

A different solution proposed by Sladić et. al. [20] grants roles to users following authentication based on context. In this proposal, users can obtain new roles that reflect his/her context. This idea is further developed by Kulkarni et. al. [9] into Context-Aware RBAC, which allows roles to be granted based on context but permitting a second layer of authorization architecture. It's responsibility is to grant and revoke roles when the context changes. This allows roles to dynamically reflect context. The problem with this solution is that it cannot be mixed with traditional RBAC. Once the system starts assigning roles based on context, all authorization rules are affected.

There is also possibility of solving that problem by adding another element not based on roles. Neumann et. al. [15] suggests adding context constraints to security policies. When the permission is checked user needs to posses not only the permission for the resource (based on his role) but also fulfill context constraints. Similar approach by Mostéfaoui et. al. [12] proposes that security rules should consist of four elements - permission, role, context and authentication methods. However, difficulties arise in defining context constraints for every permission. It would also be hard to maintain and would repeat over without any form of their abstraction and aggregation. The proposal does not describe how to define context constrains themselves, how to check context during the authorization process and how to measure performance of the proposal in real applications.

Lima et. al. [10] adds another context dimension to current security rules. This proposal makes security policies three dimensional with context, permission and role. It differs from xoRBAC [15] in that it takes context more abstract and complex. Corrad et. al. [3] suggests omitting the roles completely and assigning permissions to contexts. Both of those approaches are interesting in that they consider and compare contexts to make decisions on how similar they are. However, they do not discuss how the contexts should be compared or how the context should be similarity defined. Also permanent context checking might consume significant amount of application resources. Mowafi et. al. [13] has described solutions for mobile applications where every service in application should run in sandbox, which would be responsible for determining security rights based on contexts and security rules.

One interesting idea has been proposed by Hung et. al. [8] He proposes three entities in security rules - object, user and activity. All entities must have some credentials. If a user wants to perform an action on an object, he/she needs to pose credentials required for both the object and the activity. Although this approach is not connected with RBAC, it can provide some interesting ideas that might be used in RBAC.

Another interesting idea is proposed by Wendong et. al. [23] He suggests adding user security level in addition to RBAC and define needed security levels to perform actions.

Idea is with adding security levels is very interesting in connection to this paper, however grants the security level manually to user and not based on context. Therefore, there should be some higher authority which decides what level a user should poses.

Neuman et. al. [17] presents some very interesting and complex approaches to extend Access Control Lists (ACL). He describes multiple conditions including context constrains and the way they should be enforced on objects. An object is protected with access rights, which can be both positive or negative and with optional set of associated conditions. Even thought he describes the solution for ACL, many of his proposals can be used in any security concept including RBAC.

4. PROPOSED SOLUTION

Security policies in organizations are very consistent and are changing just slightly over time. Most of the organizations do not want or do not need to apply any radical changes. Therefore CAS must be another logical step to evolve current security. This allows us to build new security rules on existing and well proven solution making the solution more accessible for people who are familiar with current solutions.

We propose creation of a security level, which is based on context in addition to traditional roles in RBAC. Level can be understood as quantification how is the user trustworthy and it is dynamically tied to user. The security level creates second security constraint beside traditional permission and therefore resources in application now can have two different kind of security rules - classic permission tied with role and security level.

```
@AllowedRoles('admin','manager')
@RequiresLevel(3)
public Resource getResource(int Id){
...
}
```

Listing 1: Example of using security levels for securing resources

As the context of the user and the application is changing, the level needs to reflect the dynamic nature of context. There are several moments when the level can be calculated. First moment is to calculate the level during user's account creation. However, this does not reflect the dynamic nature of context and therefore is unsuitable for our needs. The opposite extreme is to determine the level on every authorization request. This would reflect changing context most reliably but it is too demanding for computational resources and also time consuming, as the context check might not be trivial. As the best compromise seems to determine the level during user's log in into application. It decreases number of context checks by several orders and at the same time it provides very accurate snapshot of the user's context. In cases when the context changes rapidly, the user can perform relogin or even the application can enforce a new level calculation manually.

The level resolution is achieved by context resolvers. Each resolver takes the responsibility for checking one particular part of context. For example, one resolver would determine network, from which the user comes. Another would check time of the day and so on. Every resolver would return, which level it grants to the user. As the security resolver is written within the application, it has access to users information (e.g. his request, information about him stored in database, etc.), as well as it can use information about the application (e.g. number of requests, number of users). Furthermore, it can even consider the machine the application is running on (e.g. load of the machine, resource usage, location of the server, etc.). The level does not need to be set in resolver and it does not decide just if to grant it or not, the resolver itself makes decision, which level to grant. After every resolver performs its inner logic and determines the level on its own, the highest level is used as the final user's security level.

The level representation by itself is very abstract. It is only necessary for the level to be comparable with other levels to know whether the given level is higher or lower then required one and also to determine the highest one. Therefore it is not important whether number, string or even some more complex structure represents the level. This leaves a lot of space for customization for a given application.

The proposed solution has many advantages. The most important ones are:

- Lightweight - it does not require any complex structures in application nor it does not consume significant system resources.
- Easy to use - it just requires adding another type of constrain to resources that need to poses CAS.
- Voluntary - if someone wants to use plain RBAC he can and just to chosen resources he might add level restrictions.
- Scalable - there is not any predefined set of levels nor there is no limit in amount of levels in application.
- Universal - the solution can be modified and used with other security architectures, not just with RBAC.

However the solution poses few limitations, which needs to be worked further on. Among them the most significant are:

- Hard to determine exact context - sometimes can happen that some resource should be accessible just from given context. For example, some resources are accessible only during the day and some just during the night. Such scenario is impossible to secure with proposed solution.
- Levels are linear - structure of the levels is strictly linear and therefore it is impossible to build some tree or even more complex structure of levels. Often happen that there are multiple context rules, which are granted different set of right. For example, levels can't model geographical situation when users from same state have some rights but people in different location of the state got additional specialized rights.

5. CASE STUDY

To demonstrate proposed solution we conduct a small case study. We implement a functional mock of e-shop with multiple actions and different security rules. User without any form of authentication is able to browse items in shop and add them to a cart. User who has logged in using his login and password can view his order history and delivery address. Finally, there is third level of authentication, lets call

User's status	Actions	Obtained
none	Browse e-shop	default
logged in	View order history	username/pwd
verified	Pay for purchase Change delivery address Set trusted IP	SMS code verification Access from set IP address

Table 1: User's status and allowed actions

user possessing it 'verified user', which allows user to change his delivery address and to pay for the purchase. This level is obtained by additional authentication done by one of two ways. The first possibility is to use specially generated code, delivered to phone by SMS. Second possibility is, that user can set trusted IP address (it can be set only if the user is already verified) and when he logs in from that IP address he is automatically considered verified.

In every application is important to know what are the operations that user might do and then determine security rules for them. In the showcase application is multiple actions, which is user allowed to perform on different authentication levels. This is shown in Table 1. It also shows how the level can be obtained. You can see that security rights are simple for our application, however for real application they most likely will be very complicated.

To demonstrate difference and effectiveness in our novel approach we implement application describe above twice. Once using levels and once using traditional methods. Levels are implemented in Java EE and integrated into PicketLink[1] framework and currently are part of production version. Therefore both implementation of application are also build on top of Java EE specifications.

In implementation without using levels needs to be code determining trusted user manually included in every method requiring, which requires user to be verified. As Listing 2 shows it brings few lines of unrelated code into those methods as well as new declaration of thrown exception. Code exhibits obvious concern tangling [22] represented by classes 'IpCheck' and 'SmsCheck'.

```
@HasRole('customer')
public void makeOrder(Order o) throws
    NotTrustedUserException{
    if(!ipCheck.isIpTrusted()&&!smsCheck.
        isSmsVerified()){
      throw new NotTrustedUserException();
    }
    ...
}
```

Listing 2: Method secured traditional way

In Listing 3 you can clearly see that method using security levels is significantly shorter and does not have any unrelated code inside. Concern separation [22] increases cohesion [21] of method and in the same time reduces coupling [21]. The class 'IpCheck' has been changed to level resolver, which reduces dependencies, because all resolvers are invoked automatically during log in. The class SmsCheck was deleted completely, because the framework allows setting up level on authenticator as is shown in Listing 4.

```
@HasRole('customer')
@RequiresLevel('2')
public void makeOrder(Order o){
    ...
}
```

Listing 3: Method secured with levels

You can clearly see that approach with levels adds to code of secured methods just one line with annotation. In addition it keeps code for determining level separated from business logic of the application and therefore allows faster development once the levels are programmed and easier maintenance and testing of the code. Without using levels there needs to be a condition for every possibility how to obtain given level. Therefore complexity of the code is unnecessary increased. Even if the security rules were extracted to another class it would add one more dependence for given class. The proposed solution can also decrease number of total classes in application because some levels are determined automatically by annotations (e.g. over authenticators).

```
@SecurityLevel("2")
public class SmsAuthenticator extends
    BaseAuthenticator {
...
}
```

Listing 4: Authenticator for SMS verification

In given example the implementation with levels removes 3 lines of code and exception declaration while adding one annotation in half of the secured methods. It also removes one class completely (while adding one annotation to authenticator) and second class is changed and there are no dependencies to it. It is obvious that with more complicated application the benefit will be even more significant.

6. CONCLUSION

In this paper we focused on the area of the CAS with focus on RBAC architecture. As is covered by related works the main issue of CAS is no existing standard or efficient best practice solution. There are multiple approaches how to add CA elements into RBAC. However they suffer from multiple inconveniences. They are either too complicated and therefore they significantly decreases one of main advantage of RBAC, which lays in its simplicity to maintain and develop. In some cases they also demands a lot of computational resources or even change the RBAC so extensively that it can be hardly called RBAC anymore.

We introduced a novel approach based on adding another security constraint beside classic permissions tied to roles. The constraint is called security level and it is based on the context. Basically level describes how much user can be trusted. To access resource in application user is required not only to posses permission through roles but also to have corresponding security level. This approach keeps advantages of RBAC and extends them further with CA elements. The solution has few limitations but brings multiple advantages to support CAS.

In future work we want to focus on transfer of security levels to other security architectures since our preliminary

[1] http://picketlink.org/

results show the potential and flexibility to utilize the advantages. Apart from that we want to examine options to overcome linearity of the levels and how to model more complex context security constraints and generally reduce downsides of the proposed solution.

7. ACKNOWLEDGMENTS

Research described in the paper was supported by the Grant Agency of the Czech Technical University in Prague, under grant No. SGS14/198/OHK3/3T/13.

8. REFERENCES

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, UK, 1999. Springer-Verlag.

[2] V. Bush and A. W. M. Think. The atlantic monthly. *As we may think*, 176(1):101–108, 1945.

[3] A. Corrad, R. Montanari, and D. Tibaldi. Context-based access control management in ubiquitous environments. In *Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on*, pages 253–260, Aug 2004.

[4] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, pages 10–20, New York, NY, USA, 2001. ACM.

[5] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 59–68, New York, NY, USA, 1999. ACM.

[6] M. Hitchens and V. Varadharajan. Design and specification of role based access control policies. *Software, IEE Proceedings -*, 147(4):117–129, Aug 2000.

[7] J. Hong, E.-H. Suh, J. Kim, and S. Kim. Context-aware system for proactive personalized service based on context history. *Expert Systems with Applications*, 36(4):7448 – 7457, 2009.

[8] L. X. Hung, J. Hassan, A. Riaz, S. Raazi, Y. Weiwei, N. Canh, P. Truc, S. Lee, H. Lee, Y. Son, M. Fernandes, M. Kim, and Y. Zhung. Activity-based security scheme for ubiquitous environments. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 475–481, Dec 2008.

[9] D. Kulkarni and A. Tripathi. Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, SACMAT '08, pages 113–122, New York, NY, USA, 2008. ACM.

[10] J. Lima, C. Rocha, I. Augustin, and M. Dantas. A context-aware recommendation system to behavioral based authentication in mobile and pervasive environments. In *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, pages 312–319, Oct 2011.

[11] M. Macik, T. Cerny, and P. Slavik. Context-sensitive, cross-platform user interface generation. *Journal on Multimodal User Interfaces*, 8(2):217–229, 2014.

[12] G. K. Mostéfaoui and P. Brézillon. A generic framework for context-based distributed authorizations. In P. Blackburn, C. Ghidini, R. Turner, and F. Giunchiglia, editors, *Modeling and Using Context*, volume 2680 of *Lecture Notes in Computer Science*, pages 204–217. Springer Berlin Heidelberg, 2003.

[13] Y. Mowafi, D. Abou-Tair, T. Aqarbeh, M. Abilov, V. Dmitriyev, and J. M. Gomez. A context-aware adaptive security framework for mobile applications. In *Proceedings of the 3rd International Conference on Context-Aware Systems and Applications*, ICCASA '14, pages 147–153, ICST, Brussels, Belgium, Belgium, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[14] M. Moyer and M. Abamad. Generalized role-based access control. In *Distributed Computing Systems, 2001. 21st International Conference on.*, pages 391–398, Apr 2001.

[15] G. Neumann and M. Strembeck. An approach to engineer and enforce context constraints in an rbac environment. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, SACMAT '03, pages 65–79, New York, NY, USA, 2003. ACM.

[16] S.-H. Park, Y.-J. Han, and T.-M. Chung. Context-role based access control for context-aware application. In M. Gerndt and D. Kranzlmüller, editors, *High Performance Computing and Communications*, volume 4208 of *Lecture Notes in Computer Science*, pages 572–580. Springer Berlin Heidelberg, 2006.

[17] T. Ryutov and C. Neuman. The specification and enforcement of advanced security policies. In *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pages 128–138, 2002.

[18] R. Sandhu. Access control: The neglected frontier. In J. Pieprzyk and J. Seberry, editors, *Information Security and Privacy*, volume 1172 of *Lecture Notes in Computer Science*, pages 219–227. Springer Berlin Heidelberg, 1996.

[19] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90, Dec 1994.

[20] G. Sladić, B. Milosavljević, and Z. Konjović. Context-sensitive access control model for business processes. *Computer Science and Information Systems*, 10(3):939–972, 2013.

[21] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Syst. J.*, 13(2):115–139, June 1974.

[22] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, Jr. N degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of the 21st International*

Conference on Software Engineering, ICSE '99, pages 107–119, New York, NY, USA, 1999. ACM.

[23] Z. Wendong and Z. Kaiji. A role-based workflow access control model. In *Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on*, volume 2, pages 1136–1139, March 2009.