Tackling access control complexity by combining XACML and Domain Driven Design

Paweł Rajba

Institute of Computer Science, University of Wrocław,
Joliot-Curie 15, 50-383 Wrocław, Poland,
pawel@cs.uni.wroc.pl,
WWW home page: http://pawel.ii.uni.wroc.pl/

Abstract. In the paper we propose a solution for designing software architecture with access control solution which reflects real business needs in a consistent, maintainable and complete way. We identify and analyze key drivers and requirements for access control, show the complexity of authorizations and propose an approach based on XACML and Domain Driven Design.

Keywords: access control, authorization, XACML, Domain-Driven Design, application architecture, hexagonal architecture

1 Introduction

Access control is a security service ([5]) to protect information from unauthorized access. It executes authorization rules (also referred as policies or just authorizations) to support basic security requirements ([6]) like confidentiality, integrity and availability. One of the areas where access control is widely adopted is the development of information systems for different industries (like banking, healthcare, automotive) to which we will refer further as business applications. During the last two decades we can observe a strong focus on making sure that the software reflects the real business needs and that is maintainable over time, especially when it comes to applying changes and new requirements. As a result a number of papers ([3], [9], [15]) has been published in the field of Model Driven Engineering and related field of Model Driven Security ([1], [11], [14]).

Domain-Driven Design, firstly described by Eric Evans in ([4]), is a philosophy of application development where attention is focused on the complexity in the business domain itself and can be considered as one of the model-driven development methodologies. One of the key elements is the development of a domain model where business and IT experts work together and where all the business logic is implemented. Domain-Driven Design introduces strategic and tactical levels where complexity is handled and structured on different levels of details.

In the paper we propose a way how to apply access control in Domain-Driven Design. Firstly we analyze the complexity of authorizations' decisions in the large

business applications as being a part of the overall business logic, highlight the risks and propose an approach which is aligned with the Domain-Driven Design. Then we review all constituents of DDD and apply appropriate access control aspects, so in the end we get a secured, DDD-based business application. Finally, we show how the proposed approach supports the common requirements towards access control system being a part of the business application.

The paper is structured as follows. In the next section we provide a basics of Domain-Driven Design. In Section 3 we define the access control model we are going to analyze and propose as a part of the solution. Section 4 shortly explains the complexity of authorization logic and in Section 5 we analyze key aspects which influence access control consideration. Finally in Sections 6 and 7 we describe the proposed solution and conclude in Section 8.

2 Basic Domain Driven Design concepts

There are many publications where Domain-Driven Design is described and analyzed ([4, 17]), however, there are not so many where security aspects are considered and discussed. In this section we describe the main concepts relevant to our further access control considerations. As indicated in the previous section, Domain-Driven Design introduces strategic and tactical levels. Strategic DDD is about the (a) split the business into *domains* and *subdomains* (what business does), (b) identify *bounded contexts* where happen all processes, activities and map them to *subdomains*,(c) establish relationships and interactions between bounded contexts and create *context map*.

Within every bounded context there is one *domain model* essential to all the business logic (data and operations). This model is developed and built by joint cooperation between business and IT experts. To make sure the cooperation results in a solution that meets business needs, both teams need to communicate using a language that is understood by both sides in the same way. This *ubiquitous language* doesn't appear on day one, but rather is being developed over time and it is a very core concept in Domain-Driven Design.

Domain-Driven Design defines a number of elements on the tactical level that can be used to create a single bounded context: services, aggregates, entities, value objects and factories. In our consideration we exclude factories as they play more syntactic role, however we differentiate services into application services, domain services and infrastructure services. In the next step we map those concepts into an architectural setup. Very often a layered architecture is used, however we consider a hexagonal architectural pattern ([2]) as more expressive and flexible one. The Domain-Driven Design concepts are mapped as follows: the domain model, domain services and application services (operations offered by the domain model to the consumers) are in the center within use case boundary, on the left side there are services (precisely: application services) which are consumed by external components (like UI) and on the right side there are services (precisely: infrastructure services) which are delivering to the application (like DBMS).

3 Access Control Model

There many approaches how access control model can be structured or organized ([16]). In our consideration we hire XACML OASIS Standard ([18]) where we can distinguish roles like Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Information Point (PIP) and Policy Administration Point (PAP) which provide a very good separation of different responsibilities.

In a common identity and access management scenario we can split activities into configuration and operation parts. In the configuration part the main activities are identity registration and provisioning as well as obtaining and provisioning permissions a.k.a. authorizations (provisioning is delivered in Policy Administration Point). In the configuration part we can also include identity removal and access revocation – important and very often underestimated activities. The operation part contains identification, authentication and access control. Moreover, in the access control usually we distinguish the following constituents: subject, resource, policy enforcement point, policy decision point and policy information point. The complete access management scenario (without the identity management part) is presented on the Fig. 1.

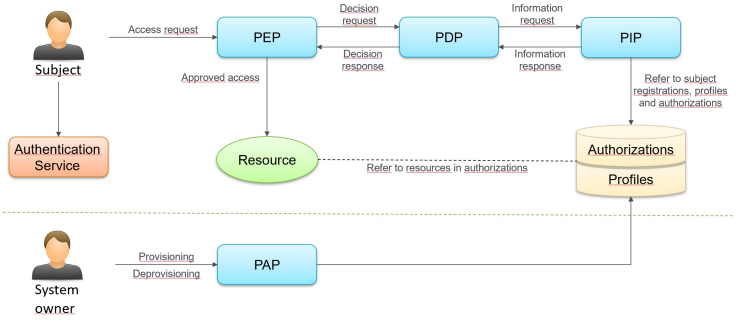

Fig. 1. Access management scenario

In the further consideration we assume that the configuration part has been already delivered and we focus on the operation part. Moreover, we assume that the authentication has been also executed and as a result we get a proof like an ID token with attributes describing the subject.

4 Complexity of authorization

Authorizations can be described in many different ways, also including many details ([10, 12, 7, 18]), but a general definition is as follows.

Definition 1. *A function* $(subject, object, action, context) \rightarrow \{authorized, denied\}$ *defines authorizations (or authorization rules, or policies). A subject (a user, a*

system, etc. with a set of attributes including roles) wants to perform an action (e.g. edit, read) on an object (e.g. an order or list of products) in a specific context (e.g. the current time, a location, related objects).

For the simplicity of our consideration, let's narrow down the definition and assume that *context = related objects* which are necessary to evaluate the policy.

Next, we can quickly come to the conclusion that authorization rules might be very simple, based on e.g. basic subject's attributes like role or age, but in most of the cases business objects not being part of the user's profile like ownership of an order or amount of transactions from the last month are needed to execute the rules which may become very complicated relatively quickly (e.g. to show the right discount offer). That lead us to the conclusion that authorization rules might be (usually quite often) a part of the business logic what is well known fact in the Model Driven Engineering area. From the Domain-Driven Design perspective that means that authorizations rules may become a part of the domain model. Let's formulate that as a statement:

Remark 1. Authorization rules may be considered as a business logic, i.e. as a part of the domain model. As a consequence, PDP logic may become a part of the domain model.

In both cases we use word "may", because there are simple solutions where it is not applicable. However, in our further consideration we put our attention to the systems where it is applicable, so the word "may" can be replaced by "is".

On the other hand there is a well recognized design principle to distinct features or main components that are loose-coupled with a minimal overlap ([8]). We can observe that a majority of the Model-Driven Security (MDS) approaches are following that principle what result in frameworks where security and business are separated ([13]). That obviously is in conflict with Remark 1.

5 Architectural building blocks influencing the access control solution

Now let's investigate the following aspects which have a significant impact on designing the access control solution: trust boundaries, models and types of operations.

5.1 Trust boundaries

A typical business solution nowadays has usually the following components: business logic, database, API (including integration capabilities), web UI and mobile UI. As a consequence we can easily distinguish the following trust boundaries: (a) domain model with application and domain services, (b) supporting components like e.g. database, (c) consuming components like e.g. mobile or web UI. In the above example we assumed API layer built on top of application services and hence in the same trust boundary. However, there are other scenarios where

API layer might be an external component (e.g. API gateway in Microservices Architecture), there are also other influencing factors like e.g. infrastructure.

Regardless of the trust boundaries setup in our solution, each of those needs to meet requirements related to the access control area. From our access control model perspective we need to apply the following: (a) PEP is mandatory in every trust boundary, (b) PDP needs to be available for PEP, however preferably as a proxy to the master PDP offering policies in a model (language) valid in a specific context (see further consideration), (c) PIP and PAP are optional and not recommended.

5.2 Types of operations

We are going to follow a very common Command-Query Separation (CQS) principle where every request is classified as a command (which is changing the state of the model) or a query (which is only reading). As we will see the classification (command vs. query) also reflects the hardness level in the authorization area. When we discuss about different operations we can hire the REST approach where there are 4 main operations reflected by the HTTP methods. It allows us to simplify reasoning and keep in the same time possibility to express any type of transaction request.

Many discussions in literature regarding access control are related either to commands or to query, but only in case of querying a single item. What is missing in that consideration is a scenario where authorization rule require filtering a list of objects. The complexity results from the fact that in practice the authorization needs to be a part of the query down to the data source (because otherwise in most of the cases performance would be not acceptable) and in most of the complex systems queries are complex and they are created in many places of the application as well as by many developers, what results in the high probability of a mistake. That lead us to the conclusion that authorization rules are reflected in the queries do data sources and it would be very hard to extract those in order to create a separate, loose-coupled component.

5.3 Models

Each trust boundary or context may introduce its internal model (language) and it is very important that authorization rules are expressed in that model. Based on our example we can easily enumerate the following models:

- M_1. Business logic model (domain model),
- M_2. Database model,
- M_3. API (REST) model,
- M_4. Web UI presentation model,
- M_5. Mobile presentation model.

Of course in general there might be different set of models, but for the simplicity of our analysis let's assume the ones enumerated above. Referring to the previous

section we can quickly conclude that in many cases authorization rules are being part of the business logic and hence the domain model. From the discussion about the types of operations we have already seen that authorization rules might be reflected both in domain model and database model. Moreover, it is a common need for the user experience purposes to apply some authorization rules in the user interface. That means that authorization rules will be also reflected in the web UI or mobile clients' models. At this point we can observe that even though the domain model is the central one, many other components' models also require some part of authorization logic. To the large extent it is the same logic what requires model transformation and that is also a well known fact in the MDS area and it has been investigated a lot in the literature.

6 Proposed Access Control Solution

In this section we provide a guideline on how to approach access control based on XACML in a business application based on Domain-Driven Design. We consider every access control function (PDP, PEP, PIP, PAP) and show how to implement it within Domain-Driven Design and the hexagonal architecture mindset. Since we want to achieve business goals in the access control, let's try to make explicit common requirements we want to have towards access control in a business application.

- R_0. Trusted identification and authentication process.
- R_1. Proper authorization rules reflecting business needs.
- R_2. Correct implementation of authorization rules.
- R_3. Completeness. All information is protected appropriately.
- R_4. Consistency. The same protection level of information in all contexts.
- R_5. Maintainability and adaptability, especially in the area of meeting new requirements and handling changes.
- R_6. Re-usability of authorization rules description in different places.

According to our assumption, the R0 requirement is out of scope of our consideration, but we assume the architecture is designed in a way to support that.

6.1 Policy Decision Point

PDP is the most challenging element in supporting enumerated security requirements and as we have already previously concluded that authorization logic is scattered in different components, there are going to be more than on PDP in our solution. To handle the relationship between let's introduce the concept of *master PDP* and *shadow PDP* as well as a PDP hierarchy.

Definition 2. *Let M_C be the core model, A_{M_C} be the complete set of authorization rules expressed in the model M_C and $M_1, \ldots, M_j$ be other models where we need some part of authorization decisions. Let a PDP based on A_{M_C} be the master PDP. Then, in order to apply the correct protection in models $M_1, \ldots, M_j$*

we need to define projections $\Pi_1, \ldots, \Pi_j$ (to scope the needed protection) and transformation functions $T_1, \ldots, T_j$ which are making transformations A_{M_C} into authorization rules expressed in the target model. Combining all together we get the set of authorization rules $T_1(\Pi_1(A_{M_C})), \ldots, T_j(\Pi_j(A_{M_C}))$. Let PDPs based on transformed authorization rules be shadow PDPs. The structure constitutes a tree with levels. Ideally should have max. 2 levels, but for obvious reasons this is not always possible. An example tree is presented on the fig. 2.

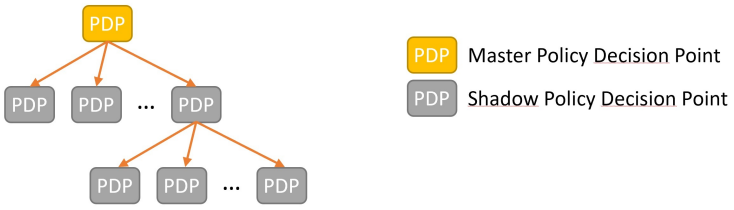

Fig. 2. Hierarchy of Policy Decision Points

All the business operations (transactions) are implemented in the business logic (domain model) and according to the Domain-Driven Design philosophy, the domain model should reflect business requirements towards business operations supported by the solution, so $M_C := M_1$. By that we can consider A_{M_C} as a main source of authorization rules. From a design perspective we recommend creating appropriate domain services to capture authorization rules. If relevant, part of that logic can be moved to respective entities and value objects, however dedicated domain services should embrace overall authorization logic.

M_2 is a little tricky, because it can be considered from 2 perspectives: on the one hand we need to make sure that appropriate queries are executed, but on the other hand we want to make sure that appropriate access control is deployed in the DBMS directly. In practice, due to strong relationship (some kind of trust) between application and database, very often access control model in the DBMS is very simplified. Nevertheless appropriate projection Π_2 and model translation and T_2 need to be applied on A_{M_C}, especially in the area of creating appropriate queries to a database. As stated earlier very helpful is automation supported by ORM system and the right implementation of *specification pattern*.

When it comes to web UI client model (M_4), it is in an untrusted domain, so usually needs are related to adjustment of presentation layer to reflect the actual permissions of a user and by that increase the user experience, so projection Π_4 and translation T_4 are to support implementation of showing/enabling and hiding/disabling appropriate functions, presenting correct messages, etc. Consideration gets a little tricky for M_5 as we have the same needs as for web UI, however in the current mobile OS devices are in a partly trusted domains, so there are more possibilities related to the data protection. Nevertheless, still projection Π_5 and translation T_5 are needed to some extent.

Finally, we can state that since REST and stateless and all requests are forwarded to the domain model where the actual authorization decisions are taken, either $\Pi_3(A_{M_C})$ is very limited (e.g. rough checks) or even $\Pi_3(A_{M_C}) = \emptyset$.

Starting from and putting in center the domain model we strongly support R_1. Doing correctly those projections and transformations, we meet $R_1 - R_4$ and by applying the Domain-Driven Design approach we increase the probability that we succeed. The best approach would be to have a product which is providing all P_i and T_i automated, then we would contribute to R_5 and R_6. In addition, by setting A_{M_1} as a single master PDP, we additionally contribute to R_6.

There is one more important aspect: in our consideration we assume a single master PDP. Please note that this master PDP is a part of a domain model and every change in business requirements needs to be reflected there. Now, the more master PDPs we introduce, the more scatter business logic is and it is harder to maintain such an environment. It is also in conflict with Domain-Driven Design principle where all business logic should be in the domain model.

To sum up, the proposed PDP setup is supporting all the stated requirements.

6.2 Policy Enforcement Point

Enforcements points are very much dependent on trust boundaries. If we assume that the REST layer is an application layer, we can quickly state that PEP is included in that layer and in order to get authorizations decisions PEP in application services call the appropriate domain services defined in the domain model (PDP). Moreover, since the database is considered as a separate component, PEP needs to be there as well. When it comes to web UI and mobile, they are both in untrusted domain, so there is no need for PEP. Here, by the correct trust modeling we can contribute to R_3 and enable R_1, R_2 and R_4.

6.3 Policy Information Point

Information point is another tricky point. The common approach for access control implementation is that actual rules are implemented in the code of the domain model. Then the purpose of the PIP is to provide permissions level of the users. If we combine PDP and PIP in a good way, we contribute again to $R_1 - R_4$. However, there are products on the market which allow to have a central PDP and define policies separately at a central point. This contributes to R_6, but actually affects R_3 and R_4, because without context visibility there is a risk that someone who defines those policies centrally makes a mistake or misses something. So, the goal is to find the right balance.

6.4 Policy Administration Point

In our consideration administration point is quite straightforward since it is to provide an endpoint where a service desk can provision access rights based on the permission levels. However, in general the topic is much more complicated,

especially in organization which has a dedicated service desk to support hundreds of applications. Then the way of managing permissions is getting very tricky, especially in situation where there are many different COTS/SaaS solutions in the portfolio. Introducing a clear and simple way of describing permission levels (e.g. roles) is critical to make sure that support team provisions the correct access rights. In the same time the actual authorization rules supporting appropriate permission levels might be much more complicated.

6.5 Complete solution

Combining all the functions described in the previous section we get a complete solution. We can see that applying Domain-Driven Design thinking combined with right trust modeling and automation, increases the probability of delivering all the expected requirements. Nevertheless we would like to emphasize several aspects. First, as we discussed previously there is strong push towards defining security as a separate concern from the business logic. However, as we have seen already, this is a very risky approach and not according to the Domain-Driven Design philosophy. Nevertheless when designing a system the right balance of centralization and decentralization must be taken very consciously.

Another aspect is that in literature (including famous Vaugn book [17]) very often Identity and Access Management bounded context is introduced as a separate one. That might be considered as a way for centralization, but we must be very careful again what we put there. We recommend to consider that context more as a PIP and PAP, but much less (if at all) as PDP. For sure it can't be considered as PEP for obvious reasons.

Finally, the architecture diagram with all the policy points applied may look as on the Fig. 3.

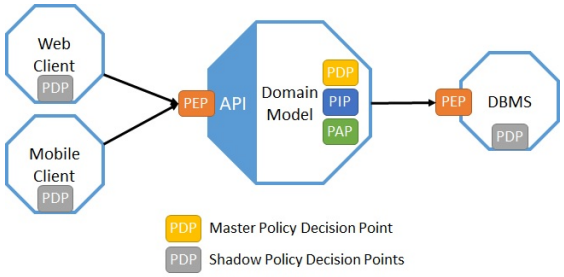

Fig. 3. Proposed Access Control Solution

7 Conclusions

In the paper we proposed a solution for designing software architecture with access control based on XACML and Domain Driven Design. We showed that

many authorization rules are indeed a part of the business logic and investigated consequences of that fact. We reviewed several common requirements and architectural building blocks influencing and driving the access control function and analyzed how those can be supported by the proposed approach.

References

1. Basin, D., Clavel, M., & Egea, M. (2011, June). A decade of model-driven security. In Proceedings of the 16th ACM symposium on Access control models and technologies (pp. 1-10). ACM.
2. Cockburn, A. Hexagonal Architecture: Ports and Adapters ("Object Structural"). June 19, 2008.
3. Cysneiros, L. M., & do Prado Leite, J. C. S. (2002, May). Non-functional requirements: from elicitation to modelling languages. In Proceedings of the 24th international conference on Software engineering (pp. 699-700). ACM.
4. Evans, E. (2004). Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional.
5. ISO 7498-2:1989, https://www.iso.org/standard/14256.html [24.03.2019]
6. ISO/IEC 27000:2018, https://www.iso.org/standard/73906.html [24.03.2019]
7. Jiang, H., Bouabdallah, A. (2017). A Lightweight JSON-based Access Control Policy Evaluation Framework.
8. Jurjens, J. (2005, May). Sound methods and effective tools for model-based security engineering with UML. In Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005. (pp. 322-331). IEEE.
9. Kleppe, A.G., Warmer, J., Bast, W., 2003. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley.
10. Lobo, J., Bhatia, R., Naqvi, S. (1999). A policy description language. AAAI/IAAI, 1999, 291-298.
11. Lucio, L., Zhang, Q., Nguyen, P. H., Amrani, M., Klein, J., Vangheluwe, H., & Le Traon, Y. (2014). Advances in model-driven security. In Advances in Computers (Vol. 93, pp. 103-152). Elsevier.
12. Margheri, A., Masi, M., Pugliese, R., Tiezzi, F. (2017). A rigorous framework for specification, analysis and enforcement of access control policies. IEEE Transactions on Software Engineering.
13. Nguyen, P. H., Klein, J., Le Traon, Y., & Kramer, M. E. (2013, December). A systematic review of model-driven security. In 2013 20th Asia-Pacific Software Engineering Conference (APSEC) (Vol. 1, pp. 432-441). IEEE.
14. Nguyen, P. H., Kramer, M., Klein, J., & Le Traon, Y. (2015). An extensive systematic review on the Model-Driven Development of secure systems. Information and Software Technology, 68, 62-81.
15. Schmidt, D. C. (2006). Model-driven engineering. COMPUTER-IEEE COMPUTER SOCIETY, 39(2), 25.
16. Uzunov, A. V., Fernandez, E. B., & Falkner, K. (2015). Security solution frames and security patterns for authorization in distributed, collaborative systems. Computers & Security, 55, 193-234.
17. Vernon, V. (2013). Implementing domain-driven design. Addison-Wesley.
18. OASIS XACML Technical Committee: "eXtensible access control markup language (XACML) Version 3.0. Oasis Standard, OASIS (2013), URL: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-specos-en.html [24.03.2019]