A Context-aware Service for Authorizing Multi-Cloud Deployments

Yiannis Verginadis[1], Ioannis Patiniotakis[1], Gregoris Mentzas[1]
[1] Institute of Communications and Computer Systems,
National Technical University of Athens,
9 Iroon Polytechniou Str., Athens Greece
{jverg, ipatini, gmentzas}@mail.ntua.gr

Abstract. The most recent advancements in cloud computing, highlight the need for supporting deployments on virtualised resources bounded to the data-intensive application requirements and not limited by the strict boundaries of each available cloud provider. Towards this direction, one important challenge is the appropriate protection of the software infrastructure used for automatically performing application components deployment over multi-clouds. Specifically, we discuss the requirements and introduce a novel attribute-based access control mechanism, able to cope with potential cybersecurity threats that may compromise the deployment of multi-cloud applications. We attempt to tackle the authorization issues from two different perspectives; namely, coping with the "access control" to various platform components and the "pre-authorization" of application deployment and data placement actions using multiple cloud providers.

1 Introduction

In recent times, there is an abundance of cloud services offered from a vastly increasing number of cloud providers, each one with its own advantages or shortcomings. Finding opportunities where the benefits of each provider are exploited at the same time, while coping with the ever-increasing requirements of the modern data-intensive cloud applications, has become imperative [1]. A generic challenge is to overcome scalability, resiliency, and security issues faced by big data and data-intensive applications on distributed platforms by using transparent and optimized multi-cloud resource provisioning [1]. One of the critical issues towards tackling this generic challenge is to design and develop the appropriate methods and tools for adequately protecting the platform components that make possible the optimized application placement on multi-clouds. We use as an example the platform presented in [2] where decision and optimization components (called Upperware) are used for ingesting user-defined application placement requirements, constraints and optimization goals in order to provide optimized decisions on initial placement or reconfigurations of data-intensive applications on multi-clouds. A second part of such a platform involves a set of components that interface with the required providers for commissioning virtual resources and deploying application components according to the initial placement or the reconfiguration decisions of the Upperware components.

This platform [2] follows a model-driven engineering approach, where models as first class citizens, initially capture placement requirements and optimization goals, providing valuable input for analysis and decision making with respect to multi-cloud application placement and last they are transformed to executable service graphs that guide the placement or reconfiguration actions across clouds.

Nevertheless, the list of the main risks associated with clouds is very long, and even longer for multi-clouds [3]. At the top of this list, we find the improper identity management, insufficient credentials and access control, unsecured application programming interfaces (APIs) [4]. Although there are several security-related topics that a platform that undertakes applications placement on multi-clouds should cover (e.g. cloud providers credentials management, component authentication, encrypted communication etc.), in this work we are focusing on an advanced authorization module, based on the Attribute-based Access Control (ABAC) paradigm. This mechanism is designed to provide access control and pre-authorization of any deployment plans that may be implemented by such a platform. We note that such security related capabilities are applicable to any application placement scenario but we mainly focus on data-intensive applications due to the inherent additional requirements for efficiency that introduce. The introduction of this kind of authorization capabilities, ensures that only eligible entities (users or components) can access protected platform resources and apply certain operations on them. Each access attempt to a resource is checked against a set of access control policies. During authorization checking, various stated and contextual information must be considered; this information relates to the requestor (i.e. user or component), the resource being accessed (e.g. data, methods etc.), the attempted operation as well as other environment data (e.g. date, component operational status, etc.). Beyond access control, the authorization infrastructure should be consulted on whether a given application deployment plan, generated by the appropriate decision and optimization components, complies with a set of deployment policies. Such policies may encompass constraints and limitations referring to application deployment (e.g. total cost or number of virtual machines deployed or location of resources). In this way, the described security enhancement of the platform leads to a protected operation of all critical software that makes decisions and implements placement or reconfiguration actions regarding multi-cloud deployments.

The rest the paper involves a relevant state-of-the-art analysis which is provided in section 2, while in section 3, we discuss the requirements of the developed service called Melodic Authorization Service. In section 4, we provide the design details of this mechanism, while in section 5 the implementation specifics of this service are described. Last, we conclude with section 6, where the next steps of this work are discussed.

2 Related work on Access Control

Several access control models have been proposed in the literature and used in software products. These models provide a framework and a method of how resources, requestors, operations, and rules may be combined to produce and enforce

an access control decision. Some of the most well-known models are the *Discretionary Access Control (DAC)* [5], the *Mandatory Access Control (MAC)* [6], the *Identity-Based Access Control (IBAC)* [6], the *Role-Based Access Control (RBAC)* [7] and the *Attribute-Based Access Control (ABAC)* [7]. Out of the most prominent paradigms is the ABAC, especially if we consider the dynamic characteristics of the cloud computing domain. ABAC uses policies that comprise rules, which in turn comprise logical conditions on several different attributes. Typically, each rule contains at least a condition (i.e. boolean expression) and a decision (permit or deny) to reach when the condition is true. Policies combine the outcomes of rules and yield the final decision using certain outcome combination methods. Attributes can be properties of the requesting entity, of the resource being accessed, of the operation requested, or any other contextual information found in the so-called external environment enclosing the interacting entities. Thus, these attributes are not limited in number or in type as in previous paradigms (e.g. RBAC which only considers the role of the access requesting entity).

Table 1. Java-based, open source, XACML tools

Product	XACML Version	License	Latest Release	Findings
WSO2 Balana[1]	3.0, 2.0, 1.x	Apache 2.0	Mar 2018	Based on Sun's XACML The most used XACML implementation so far
Authzforce CE[2]	3.0	Apache 2.0	Apr 2018	Lack of clear documentation for extensions development
JBoss Picketbox[3]	2.0	LGPL 2.1	Feb 2011	Merged with Keycloak project since 2015
Xacml4j[4]	3.0, 2.0	GPL 3.0	Jul 2014	No recent activity at Github since 2014
XACML Light[5]	2.0	n/a	Apr 2013	PDP & PAP only
Heras AF[6]	2.0	Apache 2.0	Aug 2016	Basic XACML 2.0 implementation
OpenAZ[7]	3.0	Apache 2.0	Mar 2016	Retired since Aug 2016
Sun's XACML[8]	2.0, 1.x	Open source	Dec 2010	No active support anymore

There are a few reference implementations of the ABAC model, but among the most important ones are the eXtensible Access Control Markup Language (XACML) and the Next Generation Access Control (NGAC) [8]. It is important to note that XACML seems to be the most widely used ABAC implementation model, since it

[1] https://github.com/wso2/balana
[2] https://authzforce.ow2.org/
[3] http://picketbox.jboss.org/
[4] https://github.com/xacml4j-opensource/xacml4j.github.io
[5] http://xacmllight.sourceforge.net/
[6] https://bitbucket.org/herasaf/herasaf-xacml-core
[7] http://incubator.apache.org/projects/openaz.html
[8] http://sunxacml.sourceforge.net/

enjoys worldwide industrial adoption in sectors like banking, healthcare, and insurance [9]. XACML is an XML-based, open-standard language promoted by OASIS, for expressing authorization policies and querying access to resources. Evaluating an access request to a resource, with regards to an XACML policy, may result in one of these four values: Permit, Deny, Indeterminate (an error occurred or needed values were missing) or Not Applicable (no related policy found). The XACML specification defines five main components that handle access decisions; the *Policy Enforcement Point* (PEP), *Policy Administration Point* (PAP), *Policy Decision Point* (PDP), *Policy Information Point* (PIP), and a *Context Handler* (CH) [8]. Since XACML introduction, several compliant frameworks have been developed and offered, both as open source software (Table 1) as well as proprietary products.

We note that we have thoroughly discussed the numerous limitations of previous works that do not follow the ABAC paradigm or its reference implementations for coping with the challenges of data authorization in dynamic multi-cloud environments [4]. In addition, this work proceeds further than previous efforts that try to overcome the basic syntactic descriptions by capturing the knowledge that lurks behind policies and rules in the sense that the relations between the attributes used for access control are considered [4]. To the best of our knowledge such a context-aware access control mechanism has not been implemented for securing the model-driven engineering interactions, within platform components that enable multi-cloud deployments and reconfigurations. Such a mechanism aspires to alleviate the repercussions of any potential components' compromise, coming from external adversaries, by authorizing in a context-aware manner any updates on the models used.

3 Melodic Authorization Service Requirements

Authorization refers to a security mechanism that determines and enforces access privileges of a requesting entity, related to resources and application features. In our approach, this service materialises two objectives. First, the supply of a security-by-design access control framework for protecting all the critical platform components that undertake the task of deploying application components over several multi-cloud resources. A platform that undertakes the decision-making and control of the way that applications can be deployment in multi-clouds, comprises a set of network-connected microservices, distributed over an intranet or a (virtual) private network. Despite the significant advantages of this approach, certain attack vectors exploiting the networked and distributed nature of the platform are possible (e.g. database-related and denial-of-service attacks). Furthermore, we consider the *semi-honest* adversarial model, where a malicious cloud provider can intercept all messages and may attempt to exploit them in order to learn information that otherwise should remain private. In order to ensure a sufficient level of security, it is necessary to protect platform components from unauthorized internal or external access attempts. For this reason, additional parameters must be taken into account; for instance, the components' previous behaviour, the origin and time of an access request, or the current state and environment of the platform. Such information is usually termed as context [10]. Contextual information can be of various types and originate from diverse sources.

The Metadata Schema introduced in [1], provides a classification of these information types, in its Context-aware Security model (e.g. permission types, physical or network location of an incoming request, time-related patterns based on normal access requests, etc.). This classification can act as a common vocabulary for collecting and leveraging information for authorization purposes.

The second objective relates to the enforcement of policies and limitations regarding the deployment of cloud applications and their data across various cloud providers. By design, a decision-making component in Upperware is responsible for producing correct deployment plans conforming to any given constraints and limitations. However, a compromised component, due to a cyber-attack, could possibly yield invalid deployment plans. Therefore, a precautionary validation step, before the actual deployment, would mitigate the risk of deploying an application in a non-conformant manner. We refer to this step as pre-authorization. The enforced limitations can be regulatory (e.g. data not allowed to leave the EU), corporate, as well as budget-, resource- or security-related. A pre-authorization policy could for example, pose a limit on the number of virtual machines deployed on a cloud provider, or require the storage of data of a certain type to be stored in nodes located only in the EU. These two objectives are quite different in their business purpose and involve different authorization rules. However, the same authorization capabilities and toolset can be used in order to achieve both of them.

4 Melodic Authorization Service Approach

In this section the details of the developed authorization service approach are highlighted. Specifically, we discuss the conceptual architecture of the system and provide details on how the contextual attributes are managed by this system.

It is important to note that we have adopted the ABAC model for this service, according also to the analysis provided in section 2. Specifically, based on the information provided in Table 1, we opted to use WSO2 Balana engine for XACML 3.0 implementation. However, replacing it with another alternative is expected to be a relatively straightforward task, since the XACML policy engine resides inside the PDP component of the XACML architecture. Furthermore, the pluggable design of the server will allow the easy replacement of plugins pertaining to the specific policy engine with new ones.

4.1 Authorization Service Architecture

Figure 1 depicts the architecture of the developed Authorization Service that enhances the XACML reference architecture [9]. The server part of the service that receives and evaluates access requests is enclosed in a dashed box in the same figure. We consider one PEP embedded within each of the multi-cloud management platform components that must be protected. This is where incoming access requests to component resources enter the platform. A PEP intercepts any incoming request, interrupts the normal request flow, extracts request information and then contacts the Authorization Server relaying the extracted information. If the server returns a

positive decision, the access request processing flow resumes. Otherwise, the access is prevented. PEP is provided as an authorization service client library, which is embedded in the platform components that should be protected. The communication to PDP is achieved using the REST API exposed by the Authorization Server, over an encrypted transport layer security (TLS) connection.

These PEPs communicate through an appropriate load balancing to a certain Authorization Server. Depending on the use case, multiple Authorization Servers can be used for scaling the access control capabilities that are provided by our system. Specifically, an Authorization Server comprises a PDP, a Context Handler and several PIPs. Each PDP constitutes a web service that provides a RESTful API for receiving access request information from PEP's, evaluating them against policies and eventually authorizing or declining the certain access request. For this purpose, PDP contains a policy evaluation engine, namely the WSO2 Balana. Upon configuration, PDP will first invoke the Context Handler to collect additional (contextual) information from the request or the environment, and then evaluates the incoming request against a number of pre-defined access control policies. Several PDP nodes may coexist in a cluster to achieve high availability, fault tolerance and fast response times. Typically, all PDP nodes share the same configuration and the same policy repository. A second component embedded in the Authorization Server is the Context Handler. Upon activation, it invokes the configured plugins to collect additional information (as attributes) about the context of the request. This contextual information is subsequently stored in a PIP (in order to become available during the policy evaluation and access control decision. Furthermore, the Context Handler receives platform or environment-related context from Context Collectors. The PIP is responsible for providing values for all the involved attributes that participate in the access control rules and policies, based on which an incoming request is permitted or denied. Specifically, the PDP's policy evaluation engine, while processing a request, might require attribute values not contained in the request itself. In such case, it invokes PIP plugins to retrieve the required attributes that constitute key-value pairs, where keys are typically in the form of Uniform Resource Names (URN).

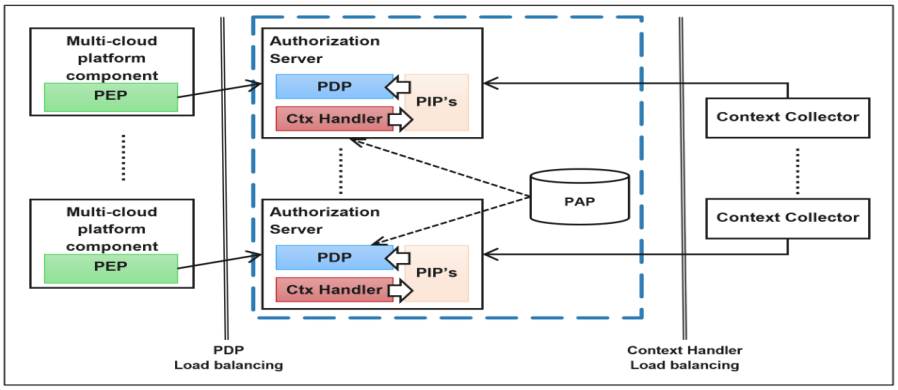

Fig. 1. Authorization Service Conceptual Architecture

The PAP is implemented in order to manage and store the XACML policies. These policies are stored in a shared place accessible by all PDPs. The architecture is complemented by a number of Context Collectors (CC), i.e. independent and domain-specific application components that aim at continuously collecting information about the multi-cloud management platform and its environment. Thus, the CCs are essentially mechanisms for collecting contextual information from the environment and propagate it to the Context Handlers. It is expected that different context information, and thus context collectors, will be needed in different multi-cloud deployments based on the application domain. Last, appropriate load balancing capabilities have been fused into our system. Specifically, the PDP Load-Balancing stands between PEP clients and the PDP nodes to prevent flooding of one or more of the Authorization Server instances and allowing the proper scaling of the system. This is implemented at the PEP-side as a dynamically configured list of PDP endpoints that are contacted either in successive order (round-robin) or selected randomly. Similarly, Context Handler load balancing capabilities can be used for enabling CCs to efficiently communicate with the appropriate CHs. We note that it is also feasible to add a third-party HTTP proxy or an open source load-balancer (e.g. Traefik[9]) by configuring the PEP clients. In either way, incoming access control requests are dispatched in a balanced way to PDP cluster nodes and the raw context is collected and handled proportionally by the CHs. We note that any of the incoming requests to be authorized come from authenticated entities which implies the implementation and integration of appropriate authentication mechanisms, a work that is out of the scope of this paper.

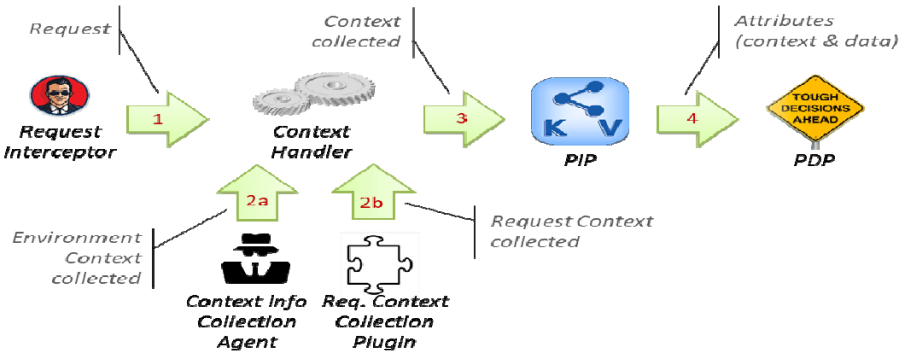

Fig. 2. Attribute values flow in the Authorization Service

4.2 Context Attributes in Authorization Service

The attributes handled by the Authorization Service can be of three types; (a) access request-related attributes (Fig. 2 – Step 1, e.g., requestor id, resource id), (b) request

[9] https://traefik.io/

context attributes in the sense that these are not stated in examined access request but are acquired from other sources (Fig. 2 – Step 2b, e.g., requestor location and device), and (c) environment/platform-related context attributes, not pertaining to a specific access request (Fig. 2 – Step 2a, e.g., operational status of a platform component). The difference in the context in the two latter cases is that the request context becomes invalid when the request has been processed, whereas environment/platform context evolves independently of the access requests. Figure 2 provides a high-level view of the attribute flow in the Authorization Service. Specifically, once the context is collected (Fig. 2 – Steps 1, 2a, 2b), it becomes available for the PIP (Fig. 2 – Step 3), which upon request provides them in the form of key-value pairs to the PDP for issuing an access control decision (Fig. 2 – Step 4).

5 Implementing Authorization Service Approach

In this section, we highlight the key components of a platform dedicated for managing applications deployments on multi-clouds that can be protected by exploiting the implemented Authorization Service. This kind of protection is enabled by deploying appropriate PEPs on each of the following platform components. In this way, we consider a minimal overhead in the implementation of the developed Authorization Service, since the approach implies that a PEP library is bundled with the application code (i.e. same Java Archive (JAR) or placed in the Classpath), while a Tomcat interceptor or "*@AuthorizationRequired*" annotations on guarded methods are added.

Business Process Management engine (BPM). It coordinates all the decision and optimization components (i.e. Upperware components in [2]) and executes the workflow to generate and execute an application deployment plan, according to a set of user-defined preferences, constraints and optimization goals. When necessary, it also repeats the whole process or parts of it to introduce deployment plan updates, as a response to changes in application demands or environment. A PEP client has to be embedded in BPM engine, in order to protect it from a potentially compromised or malfunctioning component or from outside-world interactions. In this way each PEP examines the origin and timeliness of the requests in order to authorize them.

Deployment and Adaptation mechanism. It is responsible for taking an application deployment plan and executing it by providing specific instructions to components able to interface with several cloud providers that should be used, for commissioning VMs and installing application components accordingly. In order to verify that a given deployment plan conforms to the application deployment policies, a pre-authorization step is taken. The plan parameters are checked against the relevant policies, and if rendered as conformant, the deployment starts. For this reason, the Deployment and Adaptation mechanism uses a PEP client to contact PDP to evaluate the plan against the posed policies. The plan pre-authorization policies are different from access authorization policies used for checking the access to previous components.

Modelling Editors. They are used to firstly create and maintain the Metadata Schema [1], valuable for providing a formal hierarchical view of the contextual attributes to be used for modelling the application placement problem. Secondly, it is

used for acquiring, from the DevOps, the appropriate application placement requirements, constraints and optimization goals that should guide the platform when undertaking the multi-cloud placement of the given application components. These editors comprise two layers; the User Interface layer, which executes in user browser, and the Backend, modelling management layer. The latter also communicates and interacts with models repository. For this reason, the second layer includes a PEP client to protect itself from unauthorized access to its functionality and data.

5.1 Use of Request Interceptor for Spring-Boot based components

Most the platform components have been implemented as Spring-Boot web applications[10]. This means they embed a minimal application server (e.g. Tomcat[11]) in order to accept incoming (HTTP) requests from other platform components, providing suitable REST APIs. The code implementing the REST API and receiving the requests needs to be protected with this Authorization Service.

One method for introducing the needed authorization capabilities is by configuring the embedded Tomcat server (of the Spring-Boot framework) to intercept the incoming requests and pre-process them before they actually reach the code that serves them. This is a standard step in the Tomcat HTTP request processing cycle and is implemented by adding special filters called interceptors. Interceptors can be added in Tomcat programmatically, during server initialisation. The interception process of an HTTP request is handled by a Login Interceptor. This interceptor is invoked three times: (a) Pre-Handle: before calling the code that is meant to service the request (i.e. MainController), (b) Post-Handle: after the MainController returns and before rendering the response, and (c) After-Completion: when response has been sent back to requestor. Regarding Spring-Boot web applications, interceptors can be added using an application configuration class that implements the WebMvcConfigurer interface. Therefore, all needed interceptors can be added in the Tomcat interceptor registry, before the server starts.

This approach does not require any modification of application source code. Instead, a new configuration class can be written to configure an authorization interceptor. This class must be packaged with existing code, and Spring-Boot will take care of using it at runtime. The downside of this method is that it applies only to Spring-Boot web applications using a Tomcat or Jetty server[12].

5.2 Use of Aspect and Aspect-Oriented Programming

A second approach for programmatically exploiting the developed Authorization Service consists of using the Aspect-Oriented Programming (AOP) paradigm. AOP is a programming approach for software modularization and separation of cross-cutting concerns [11]. This is achieved by adding extra functionality (called Advice) to existing code without significantly modifying the source code (i.e. annotations). This

[10] http://spring.io/projects/spring-boot
[11] http://tomcat.apache.org/
[12] https://www.eclipse.org/jetty/

addition typically occurs during the software building phase in a task called weaving, which is undertaken by specialised tools called weavers. The code to be modified is identified via pointcuts, which are specifications of those code artefacts (typically classes and methods) that need to be enhanced with advices. An advice, along with the pointcuts that specify the code it must be applied onto, is called an Aspect. We note that the *Before* aspects are used to enforce PEP functionality, which means that a call to a guarded method is intercepted, PEP checks that this call is permitted and then the actual method execution occurs.

AOP allows the non-core functionality of a software component (e.g. logging of code executions, measuring duration, and authentication/authorization) to be moved away from the code implementing the core business of the component. The non-core functionality is added and interleaved with the core functionality during the software build phase (via weaving). The Spring framework provides an AOP implementation. Spring AOP is proxy-based, meaning each code artefact that can be enhanced with advices will be wrapped by a suitable proxy object that is actually invoked by the calling code. The proxy can subsequently pass control to the actually requested code. Proxies are automatically introduced at code-level (during weaving), while source code remains intact. Thus, this process is transparent to the programmer.

Regarding the use of Aspects in the Authorization Service, an authorization aspect has been introduced. The corresponding advice (i.e. the wrapping proxy code) will intercept the code invocation in order to perform a series of authorization related tasks; namely, (a) create/reuse a PEP client object, (b) collect invocation information (i.e. method signature and arguments), (c) connect to a PDP server and pass the collected information, (d) receive the PDP server response (permit, deny, error), and (e) in case of permit (subsequently), call the actual (wrapped) code or raise an authorization error, otherwise. When the wrapped code is a method of a Web or REST controller class, and that method is mapped to a Web or REST endpoint, then the corresponding (HTTP) request object is introspected to extract all related information.

6 Conclusions

In terms of this work, a novel authorization service was proposed in the context of protecting components that undertake the data-intensive application deployments on multi-clouds. Specifically, this work is seen from two different perspectives; namely, the “access control” to various platform components and the “pre-authorization” of application deployment and data placement in cloud providers. In the first case, authorization capabilities are considered to be responsible for protecting the platform itself from illegal access attempts and interference with its normal operation. In the latter case, authorization capabilities refer to the pre-authorization of application deployment and data placement plans, produced by the decision-making and optimization components, considering a given set of policies, constraints and goals.

The next steps of this work, involve the appropriate integration with other critical security services that will address the security challenges in the multi-clouds deployment domain in a holistic manner. Such additional security mechanisms

involve the use of authentication services, encrypted communication (e.g. SSL/TLS), mechanisms for counterfeiting the man-in-the-middle attacks, using component verification and digital certificates among others.

Acknowledgments. The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731664. The authors would like to thank the partners of the MELODIC project (http://www.melodic.cloud/) for their valuable advices and comments.

References

1. Verginadis, Y., Patiniotakis, I., Mentzas, G.: Metadata Schema for Data-Aware Multi-Cloud Computing. IEEE (SMC) INISTA (2018) 1-9
2. Horn, G., Skrzypek, P.: MELODIC: Utility Based Cross Cloud Deployment Optimisation. In proceedings of the 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA), (2018), DOI: 10.1109/WAINA.2018.00112
3. CSA: The Treacherous 12 - Cloud Computing Top Threats in 2016. URL https://cloudsecurityalliance.org/group/top-threats
4. Verginadis, Y., Michalas, A., Gouvas, P., Schiefer, G., Hübsch, G., Paraskakis, I.: PaaSword: A holistic data privacy and security by design framework for cloud services. Journal of Grid Computing, (2017) 1–16, DOI: 10.1007/s10723-017-9394-2
5. Decker, M.: Modelling of location-aware access control rules. In: F. M. Maria Cruz-Cunha (Ed.), Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts, Information Science Reference, Hershey, PA, USA, 2011, incollection 57, pp. 912–929.
6. Hu, V.C., Ferraiolo, D., Kuhn, R., Friedman, A.R., Lang, A.J., Cogdell, M.M., Schnitzer, A., Sandlin, K., Miller, R. and Scarfone, K., 2013. Guide to attribute based access control (ABAC) definition and considerations (draft). NIST special publication, (2013) 800, 162
7. Ferraiolo D., Cugini J., Kuhn D.R.: Role-based access control (RBAC): Features and motivations. In Proceedings of 11th annual computer security application conference, (1995) 241-48
8. Ferraiolo, D., Chandramouli, R., Kuhn, R., Hu, V.: Extensible access control markup language (XACML) and next generation access control (NGAC). In Proceedings of the ACM International Workshop on Attribute Based Access Control, ACM, (2016) 13-24
9. Garcia-Alfaro, J. and Navarro-Arribas, G.: Prevention of cross-site scripting attacks on current web applications. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, (2007) 1770-1784
10. Abowd, G., D., Dey A., K., Brown, P., J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In International symposium on handheld and ubiquitous computing, Springer, (1999) 304-307
11. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., M., Irwin, J.: Aspect-oriented programming. In European conference on object-oriented programming, Springer (1997) 220-242