Towards an Access-Control Metamodel for Web Content Management Systems

Salvador Martínez[1], Joaquin Garcia-Alfaro[3], Frédéric Cuppens[2], Nora Cuppens-Boulahia[2] and Jordi Cabot[1]

[1] ATLANMOD, & École des Mines de Nantes, INRIA, LINA, Nantes, France
{salvador.martinez_perez, jordi.cabot}@inria.fr
[2] Télécom Bretagne; LUSSI Department Université Européenne de Bretagne, France
forename.surname@telecom-bretagne.eu
[3] Télécom SudParis; RST Department CNRS Samovar UMR 5157, Evry, France
joaquin.garcia_alfaro@telecom-sudparis.eu

Abstract. Out-of-the-box Web Content Management Systems (WCMSs) are the tool of choice for the development of millions of enterprise web sites but also the basis of many web applications that reuse WCMS for important tasks like user registration and authentication. This widespread use highlights the importance of their security, as WCMSs may manage sensitive information whose disclosure could lead to monetary and reputation losses. However, little attention has been brought to the analysis of how developers use the content protection mechanisms provided by WCMSs, in particular, Access-control (AC). Indeed, once configured, knowing if the AC policy provides the required protection is a complex task as the specificities of each WCMS need to be mastered. To tackle this problem, we propose here a metamodel tailored to the representation of WCMS AC policies, easing the analysis and manipulation tasks by abstracting from vendor-specific details.

1 Introduction

Web Content Management Systems (WCMSs) is the technology of choice for the development of millions [4] of Internet sites and increasingly, becoming a framework widely used for the development of Web applications. They provide an integrated environment for the definition of the design, layout, organization and content management of the application and, because of its relative ease of use, they enable users with little technical knowledge to develop fully functional systems.

This widespread use highlights the importance of security requirements, as WCMSs may manage sensitive information whose disclosure could lead to monetary and reputation losses. Due to the nature of the users, the focus has been often put in facilitating the WCMSs configuration. Although this systems are easy to use, a proper configuration is needed to minimize the introduction of vulnerabilities. As a consequence, tools for checking the configuration of WCMSs have been provided and analysed by the scientific communities. However, this tools are focused in low-level security aspects like management of cookies or prevention of SQL injection vulnerabilities [5,8].

[4] http://trends.builtwith.com/cms (15 April 2013)

Moreover, despite some approaches for extracting AC information from dynamic web applications source code[3,2], little attention has been brought to the analysis of how developers use the content protections mechanisms provided by WCMSs systems. Particularly, Access-control techniques, integrated in most WCMSs and capable of enforcing confidentiality and integrity of data must be analyzed so that no logical flaws are present in the security policy. Unfortunately knowing if an implemented AC policy on a WCMS provides the required content protection is a complex and error-prone task as the specificities of each WCMS vendor AC implementation must be mastered (e.g. the set of roles and permissions that can be defined vary largely among the different WCMSs).

In order to tackle this problem, we propose to raise the level of abstraction of the AC implementation so that it gets represented acording to a vendor-independent metamodel. This WCMS security metamodel must be able to represent WCMS specific information along with AC concerns. We can regard such a metamodel as an extension of typical AC models[6] specially tailored to the representation of security in WCMSs.

Ideally, this models should be automatically obtained from existing WCMS AC configurations. Therefore, here, along with the description of the metamodel for the representation of WCMS AC policies, we describe the process to automatically extract them from a Drupal[1] WCMS, one of the three most popular WCMSs. Note however, that the extraction from other WCMSs like Wordpress or Joomla will follow the same process. Once these models are available, they can be analysed in a generic way, focusing in the security aspects and disregarding the specificities of concrete vendors. Moreover, Model-driven tools for querying, performing metrics, provide visualizations, etc, become automatically available, easing the analysis tasks.

Combining the vendor-independent representation and the extraction process, migration and reengineering tasks are facilitated. Recovered AC policies represented in our metamodel can be used, after its analysis, correction, etc., as a pivot representation for automatically generating correct configurations or configurations for other WCMSs.

The rest of the paper is organized as follows. In Section 2 previous concepts are introduced whereas is Section 3 we describe our proposed WCMS AC metamodel. In Section 4, the extraction approach over a Drupal system is described. Applications of the extracted model are summarized in Section 5. Finally, Section 6 concludes the paper and discusses some future work.

2 Background and Motivation

WCMSs are Content Management Systems (CMSs) specially tailored to the authoring of content in the Internet. They integrate facilities for the definition of the design, layout, organization and collaborative content management of web sites and can also be used, due to the wide range of features they offer, as a framework basis for the development of web applications. They are, due to its relative ease of use (they allow users with little knowledge of web markup and programming languages to create and manage fully functional web sites) and low cost, the technology of choice for the development of millions of web sites.

In general, they are composed by a back-end, comprising the repository of contents and administrative tools and a front-end that displays this information to web clients.

Access-control[7] is a mechanism aiming at the enforcement of the Confidentiality and Integrity security requirements. Basically, AC defines the Subjects, Objects and Actions of a system and provides the means to describe the assignment of permissions to subjects. This permissions declare which actions the subject is authorized to perform on the objects of the system. It is, due to its conceptual simplicity w.r.t. other techniques, like cryptography, heavily used in multiple domains and it has been integrated, among others, in file systems, databases, network filtering languages and WCMSs.

There exist different models for the specification of Access-control policies, where the current trend is Role-based access control (RBAC)[6], as it simplifies the administration of security policies by granting privileges to roles and not directly to users.

Motivation. As discussed in the introduction, security is a critical concern in WCMSs as they may manage sensitive information. Therefore, security mechanisms have been integrated in most WCMSs where access-control mechanisms play a prominent role. However, WCMSs users often lack depth technical and security knowledge, so that the implemented access-control policies may contain security flaws. For instance, in Drupal, the permission *Delete any content* of type Article could be, by mistake, granted to the default role *Authorized User*. As all user-defined roles inherit by default from this role, this mistake will give them the capacity of deleting the content of the application. Furthermore, the frequent need of migrating from one WCMS to another, also highlights the need for understanding the current security policy so that it can be accurately translated especially, since the security concepts in each WCMS differ. Failing at doing so will often imply putting the new system under risk.

In this scenario, analysing and understanding the security policy enforced by a WCMSs turns up as a critical necessity. Unfortunately, each WCMSs vendor provides its own access-control model and management tools so that this analysis tasks requires in-depth knowledge of the concrete system in hand.

We believe that, in order to tackle this problem, the level of abstraction of the AC policies enforced in WCMSs needs to be raised, so that the information is represented in a vendor-independent manner. We propose thus to represent the AC policies as models corresponding to a WCMS access-control metamodel. In the rest of the paper, we will detail the proposed metamodel and extraction approach.

3 WCMS Access-Control Metamodel

Central to the process of recovering and analysing the access-control information of WCMSs is the definition of a metamodel able to concisely represent the extracted Access-control information in the domain of WCMSs. This metamodel must also be platform-independent, so that we can analyse the access-control information disregarding the especificities of the concrete WCMS security features and implementation.

Figure 1 depicts our proposal for such a metamodel. It is an RBAC-inspired metamodel, thus, containing all RBAC basic concepts along with WCMS specific infor-

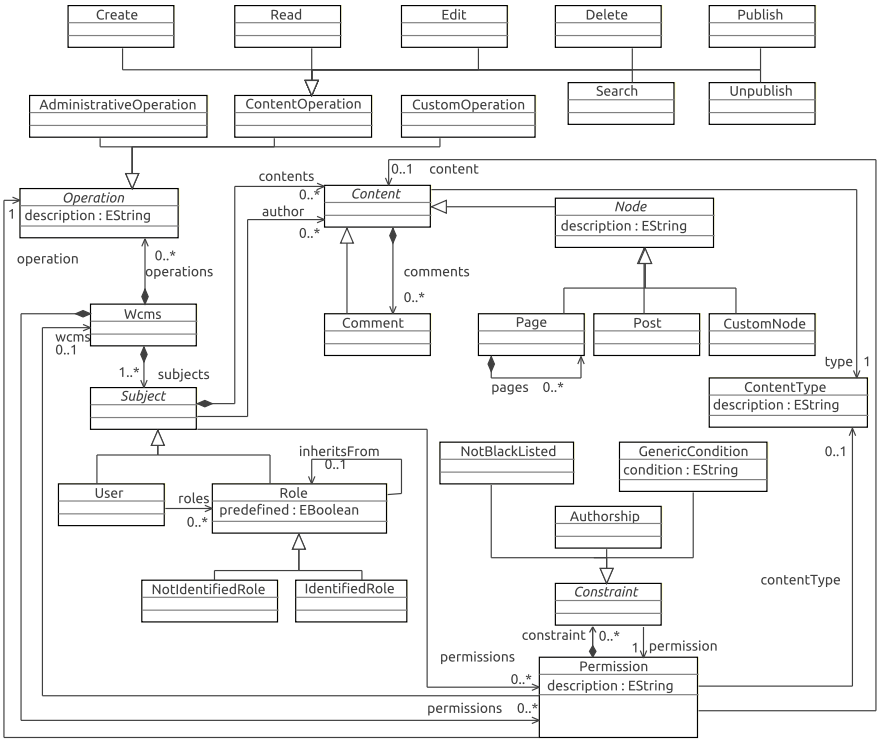

Fig. 1. WCMS metamodel excerpt

mation. It consists basically of four kind of elements. Contents, i.e., the information hosted by the system, Actions, i.e., operations that can be performed on the WCMS, Permissions, i.e., the right of performing these Actions and Subjects, i.e., the triggers of Actions. In the following, we will detail the metamodel concepts of these categories along with its rationale.

Content. The content of a WCMS is the information it manages. This is represented in our metamodel by the *Content* metaclass. Each WCMS defines its own kinds of content. To be able to represent that eventuality, our *Content* elements have a *ContentType* that identifies its type. This also allows for the representation of fine-grained and coarse grained access-control. Effectively, some WCMS access-control models allow for the definition of different permissions for individual content elements, while others only allow the definition of global permissions on all the contents of a type.

Then, we provide the users of our metamodel with some predefined kinds of content. In one side we have *Node*, representing the principal contents of the WCMSs. We have specialised them in two subclasses: *Page* that represents full content pages (that can contain other pages) and *Post* that represents individual blog posts. We also provide a *CustomNode* metaclass so that additional types of nodes ca be integrated. On the other side, we have *Comments* that represents comments that can be posted in any

other content element. We do not represent pages in the back-end of the WCMS used to administer content. That behaviour will be represented by permissions of executing administrative operations on the WCMS.

Operations are the actions than can be performed over the *WCMS*. We can divide all operations that can be done over a WCMS between two types, content operations and administration operations (e.g., operations to manage users and roles). The latter category is more WCMS specific and as such, we will uniquely represent the permissions on that category with the metaclass *Administration Operation*.

W.r.t. the content operations, in our metamodel, all CRUD actions are available: *Create* for the creation *Content* elements; *Edit* for the modification of already created *Content* elements; *Read* for reading/viewing created *Content* elements; *Delete* for the deletion of *Content* elements. The *Search* operation is also available. It is a very common action in WCMSs and, as it can be expensive, it is usually restricted only to certain users (e.g., logged users). Additionally, there are two special actions the *Subjects* of WCMSs can perform. *Publish* and *Unpublish*. In WCMSs it will not be surprising to find that some *Subjects* can create and manage contents using CRUD operatios while not being authorized to make it publicly available without revision-moderation from another authorized *Subject*. These two actions support this behaviour. *Publish* is the action of making available some created *Content* element and *Unpublish* is the action of removing a piece of *Content* from its place of publication without internally deleting it.

Finally, we also consider the possibility of new operations that may appear e.g., when extending the WCMS. In order to be able to represent these possible new operations, we provide the *Custom Operation* metaclass. This way, if the WCMS is extended with the capability of e.g., doing polls, an eventual new operation, vote, could be represented by this metaclass.

Permissions are the right of performing actions on the WCMS. They can define *constraints* that restrict the *Permission* to execute the corresponding action only when certain conditions hold. In our metamodel, we have identified two kinds of *Constraints* that typically appear in WCMSs: *Authorship* and *NotBlacklisting*. The former expresses that the permission is effective only if the *Subject* is the author of the *Content* whereas the latter restricts the applicability of the permission to the condition of not being blacklisted. Other conditions may exist and therefore, we provide the means to represent them by the *GenericCondition* metaclass. It holds in a text field the condition of the *Constraint*. The nature of the contents of this text field is left open to the metamodel users, so that in can hold conditions expressed in natural language or in more formal constraint languages like OCL. Similarly, in [4] the authors added the constraints to permissions represented by triggers to a metamodel tailored to represent Relation Database Management Systems (RDBMS) access-control by adding the source code of the triggers. As in there, the representation and extraction of the meaning of such custom constraints will require a further analysis. We leave such analysis as future work.

Subjects are the elements interacting with the contents of the WCMS by performing actions (note that a *Subject* can be the author of a piece of *Content* and that this may

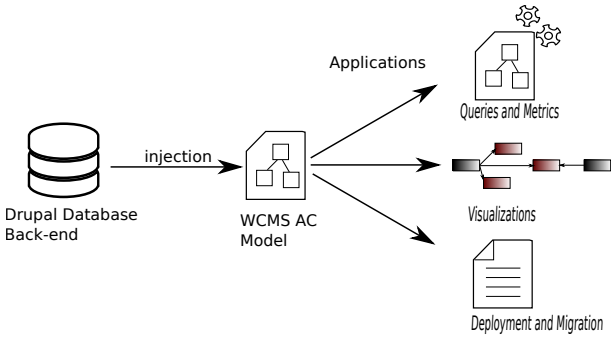

Fig. 2. Drupal AC extraction and analysis approach

influence the Access-control of the information. Thus, this is represented in our metamodel). Following a RBAC approach, in our metamodel we have two kinds of Subjects: *Users* and *Roles* where *Users* get *Roles* assigned. However, unlike RBAC we are more flexible in the permission assignment by allowing both *User* and *Role* to get permissions granted.

Depending of the WCMS in hand, *Roles* are predefined by the application or can be defined by the developer. Both cases can be discerned in our metamodel by using the *predefined* attribute of the *Role* metaclass. Moreover, we have identified two specific roles that often appear in WCMSs. *IdentifiedRole* and *NotIdentifiedRole* are often present in WCMS to discriminate between not logged and logged users. As such, we have decided to add them to the metamodel so that this behaviour can be easily modeled. Finally, role inheritance is also supported.

4 Approach

Although our metamodel could be manually filled by inspecting the AC information using the WCMS administration tools, ideally, it should be filled by an automatic reverse-engineering approach. In the following, we present such an automatic process for a Drupal WCMS although it can be easily adapted to work with other WCMSs. The process in depicted in Figure2.

In Drupal, contents, along with the corresponding access-control information (i.e., users, roles and permissions) are stored in a database back-end. Thus, in order to obtain a model conforming to our WCMSs metamodel with this information, an injection process need to be launched. This process performs SQL queries over the database back-end while creating, as output, the corresponding model elements. Note that, additionally, extra access-control rules could be defined or modified programatically, in the source code of plugins, etc. Techniques as the ones in [3,2] could be used to as a further step to complement our approach.

Note however, that this first step will require a previous step, i.e., the discovery of the data model of the WCMSs as each WCMS defines its own. For doing so, we can rely on the WCMS available documentation or in worse cases, to schema extraction tools. In the case of drupal, the relevant tables are the following ones: USERS, that contains all

DRUPAL	WCMS Metamodel
User	User
Default Role	Role
User-defined Role	Role with inheritance relation to the Authenticated Role
Page and Article types	ContentType for Page and Article
Node	If the type is page or article, Page with the proper ContentType. If it is blog post, Post. If there is another type of node, CustomNode
Content actions	ContentOperation.
Comment	Comment pointing to the corresponding Content
Permission	Permission with the corresponding links to the subject, object and operation. For content permissions, link to Content or ContentType (for permissions granted on all content of a given type). For administrative permissions, link to the WCMS instance.

Table 1. Mapping from drupal to our metamodel

system users; ROLE, containing all roles; USER_ROLES relating users with assigned roles; PERMISSION connecting roles with permissions; COMMENT for the special content of the type comment and NODE for all the other content types.

Drupal AC. Extraction evaluation: In drupal, there exist three main kinds of content, Pages, Articles and Comments and three roles by default, *Anonimous*, *Authenticated* and *Administrator*. By default, any new user-created role, what is allowed, inherits the permissions from the *Authenticated* one. Thus, to create roles more restricted than the *Authenticated* role, the *Authenticated* role needs to get permissions removed. Using the default modules, permissions can not be granted in concrete content, e.g., concrete pages but on content types. This way, permission to edit content can be granted and all Pages but not on individual pages (apart from the distinctions made wrt to ownership and publish/unpublished).

Our metamodel is capable of representing the AC information of Drupal by using an injector performing the mappings summarized in Table 1.

5 Applications

Once the injection process is finished, we can start analysing and manipulating the obtained model in a vendor-independent way. We summarize here some applications.

Visualization: Visual data is often easier and faster to analyze than textual or tabular data. Using MDE tools we can easily provide a visualization of our WCMS AC model so that the relation between subjects, objects and permissions can be easily grasp.

Queries: The most basic thing we may want to do with a security model is to query it to learn more about specific details of the security policies currently enforced in the WCMS. As an example, we could want to know what elements can be accessed by a given user, taking into account its assigned roles and also the permissions inherited from parent roles. This is very complex to do directly on the WCMS itself since

this information is scattered among a number of database tables which are completely vendor-specific. Instead, when using our model we can just use a standard model query language to traverse the information in the extracted model classes. The model query is defined just once and can be executed on security models extracted from any relational vendor.

WCMS migration: New requirements to be met by the application, discovered security vulnerabilities, technological choices, etc., may impose the migration from one WCMS to another. In this scenario, properly migrating the access-control information (users, roles, permissions) becomes critical. Our metamodel can be used as a pivot representation. Representing the AC information of the old WCMS in a model corresponding with our metamodel will facilitate its understanding and analysis, thus, helping to provide a good translation towards the AC model in the new WCMS.

6 Conclusions and Future Work

We have presented a metamodel specially tailored for the representation of Access-control policies of WCMSs in a vendor-independent manner along with an automatic process for extracting it from Drupal WCMSs. This model facilitates de analysis and manipulation of the implemented policies by isolating them from the specific details of each WCMS system.

As future work we plan to extend our reverse engineering process for the other major WCMSs. Moreover, we intend to continue working on the applications sketched in Section 4 and to investigate the benefits of a translation from our metamodel to XACML specifications to benefit from existing general security tools and research. Finally, we would also like to explore how Digital Rights Management (D.R.M.) concepts could be integrated in our metamodel as DRM appears to start becoming a common requirement for WCMSs under specific scenarios.

References

1. Drupal Open-source CMS. `http://drupal.org/`, 2013.
2. M. H. Alalfi, J. R. Cordy, and T. R. Dean. Recovering role-based access control security models from dynamic web applications. In *Web Engineering*, pages 121–136. Springer, 2012.
3. F. Gauthier, D. Letarte, T. Lavoie, and E. Merlo. Extraction and comprehension of moodle's access control model: A case study. In *(PST), 2011*, pages 44–51. IEEE, 2011.
4. S. Martínez, V. Cosentino, J. Cabot, and F. Cuppens. Reverse Engineering of Database Security Policies. In *DEXA 2013 (to appear.)*. LNCS, 2013.
5. M. Meike, J. Sametinger, and A. Wiesauer. Security in open source web content management systems. *Security & Privacy, IEEE*, 7(4):44–51, 2009.
6. R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, pages 47–63. ACM, 2000.
7. R. S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.
8. G. Vaidyanathan and S. Mautone. Security in dynamic web content management systems applications. *Communications of the ACM*, 52(12):121–125, 2009.