Computer Languages, Systems & Structures 49 (2017) 36–61

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Model-based analysis of Java EE web security misconfigurations

Salvador Martínez [a,b,*], Valerio Cosentino [a,d], Jordi Cabot [c,d]

[a] AtlanMod team (Inria, Mines Nantes, LINA), Nantes, France
[b] CEA, LIST, Laboratory of Model Driven Engineering for Embedded Systems, Gif-sur-Yvette, France
[c] ICREA, Barcelona, Spain
[d] UOC, Barcelona, Spain

ARTICLE INFO

Article history:
Received 10 October 2016
Received in revised form
27 January 2017
Accepted 3 February 2017
Available online 10 February 2017

Keywords:
Model-driven engineering
Security
Reverse-engineering

ABSTRACT

The Java EE framework, a popular technology of choice for the development of web applications, provides developers with the means to define access-control policies to protect application resources from unauthorized disclosures and manipulations. Unfortunately, the definition and manipulation of such security policies remains a complex and error prone task, requiring expert-level knowledge on the syntax and semantics of the Java EE access-control mechanisms. Thus, misconfigurations that may lead to unintentional security and/or availability problems can be easily introduced. In response to this problem, we present a (model-based) reverse engineering approach that automatically evaluates a set of security properties on reverse engineered Java EE security configurations, helping to detect the presence of anomalies. We evaluate the efficacy and pertinence of our approach by applying our prototype tool on a sample of real Java EE applications extracted from GitHub.

1. Introduction

Java EE is a popular technology of choice for the development of dynamic web applications (serving also as the basis for other less general purpose frameworks) that expose distributed information and services to remote users. In this scenario, security is a main concern [1], as the web resources that constitute the web application can be potentially accessed by many users over untrusted networks. As a consequence, the Java EE framework provides developers with the means to specify access-control policies in order to assure the confidentiality and integrity properties of the resources exposed by web applications.

Unfortunately, despite the availability of these security mechanisms, implementing security configurations remains a complex and error prone activity where high expertise is needed to avoid misconfiguration issues, that could inflict critical business damages. In fact, the Open Web Application Security Project (OWASP) document ranks web application misconfigurations in 5th position on the top ten of most critical security flaws [2], since they are easy to exploit and can have strong business impact.

For the concrete case of access-control and Java EE applications, and disregarding ad hoc security implementation mechanisms tangled in the application code, role-based access-control (RBAC) [3] policies are specified by writing constraints using a low-level rule-based language with two different textual concrete syntaxes and with relatively

* Corresponding author at: AtlanMod team (Inria, Mines Nantes, LINA), Nantes, France.
E-mail address: salvador.martinez@mines-nantes.fr (S. Martínez).

http://dx.doi.org/10.1016/j.cl.2017.02.001
1477-8424/

complex execution semantics. Concretely, the user can either write constraints in the XML web descriptor file by using a set of predefined tag elements, directly write annotations (with a different syntax and organization w.r.t. the XML tag elements) on the Java Servlet components or combine both mechanisms. Then, combination rules between constraints and the corresponding execution semantics must be taken into account in order to understand what policy is being effectively enforced.

This complexity may lead to the introduction of anomalies and misconfiguration problems (e.g., unexpected rule outcomes, unexpected interactions between access-control rules, etc.) with effects varying from simply increasing unnecessarily the complexity of the specified policies to the introduction of unexpected behaviors such as granting access to resources to unauthorized parties or precluding it to the authorized ones, as confirmed as well by the participants in the online survey reported in Section 3.

In order to tackle this problem, we introduce a reverse engineering approach to automatically detect inconsistencies and misconfigurations in Java EE web applications. First, we define a list of properties a web application must satisfy in order to be free from important anomalies, such as redundancy (i.e., specification of unneeded constraints that overcomplicate the policy) and shadowing (i.e., specification of constraints that are never enforced). Secondly, we present an extraction method to parse the security configuration of a given web application (taking into account both, the web descriptor configuration and the Java annotations) and represent it as a Platform Specific security Model (PSM) specific to Java EE web access-control policies. Then, OCL queries and model transformation operations are implemented on top of that model in order to enable the automatic evaluation of the defined properties on any given Java EE web application. Additionally, our tool produces diagnosis reports that help to identify the source configuration elements responsible for the property violations, thus, helping developers to fix them.

We evaluate the efficacy of our approach by exercising our tool on a battery of publicly available Java EE web applications extracted from GitHub, a web-based Git repository hosting platform. This evaluation has shown that a relevant number of security configurations do violate our recommended properties and that our tool is able to successfully detect those violations.

The rest of the paper is organized as follows. Section 2 describes the access-control mechanisms of Java EE. Section 3 presents a motivation survey about the use of security in Java EE projects. Section 4 describes a number of security properties. Section 5 shows how to extract access-control models from Java EE web applications while Section 6 details our automatic approach to evaluate our properties on them. Section 7 presents a number of additional applications to our approach. Section 8 shows evaluation results and Section 9 gives details about the tool implementation. Related work is discussed in Section 10. Finally, we conclude the paper in Section 11 by presenting conclusions and future work.

2. Java EE web security

Roughly speaking, in the Java EE realm, when a web client makes a HTTP request, the web server translates the request into HTTP Servlet calls to web components (Servlets and Java Server Pages) to perform some business-logic operations.

In this schema, a very important requirement is to ensure the confidentiality and integrity of the resources managed by the web application as they can be accessed by many users and traverse unprotected networks. In that sense, the Java EE framework provides ready-to-use access-control facilities. In the following we will briefly describe the mechanism offered by Java EE for the implementation of access-control policies in web applications.

As introduced before, Java EE applications are typically constituted of JSPs and Servlets (JSPs are in turn translated to Servlet). The access-control mechanism in place in this tier is in charge of controlling the access to these elements along with any other accessible artifact (pure HTML pages, multimedia documents, etc.).[1] These access-control policies can be specified using two different mechanisms: declarative security and programmatic security, the latter being provided for the cases where fine access-control, requiring user context evaluations, is needed. Nevertheless, the Java EE specification recommends a preferential use of declarative security whenever possible.

Regarding declarative access-control policies, two alternatives are available: (1) writing security constraints in a *Portable Deployment Descriptor* (*web.xml*) and (2) writing security annotations as part of the Servlets Java code (note however that not all security configurations can be specified by means of annotations).

Listing 1 shows a security constraint defined in a *web.xml* descriptor. It contains three main elements: a *web-resource-collection* specifying the path of the resources affected by the security constraint and the HTTP method used for that access (in this case the */restricted/employee/** path and the GET method); an *auth-constraint* declaring which roles, if any, are allowed to access the resources (only the role *Employee* in the example) and a *user-data-constraint* that determines how the user data must travel from and to the web application (set to *None* in the example, i.e., any kind of transport is accepted). Additionally, although it is not mandatory, the web descriptor may contain role declarations (see Listing 2).

[1] http://download.oracle.com/otndocs/jcp/servlet-3.0-fr-oth-JSpec/

Listing 1. Security constraint in web.xml

```
<security-constraint>
  <display-name>GET To Employees</display-name>
  <web-resource-collection>
    <web-resource-name>Restricted</web-resource-name>
    <url-pattern>/restricted/employee/*</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
     <role-name>Employee</role-name>
  </auth-constraint>
  <user-data-constraint>
     <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Listing 2. Role declaration in web.xml

```
<security-role>
    <role-name>Employee</role-name>
</security-role>
```

The equivalent security constraint, defined by means of annotations is shown in Listing 3. The *@WebServlet* annotation identifies the Servlet and the resource path in the web container. Then, the main security annotation is *@ServletSecurity* that has two attributes: value, that corresponds to a nested annotation *@HttpConstraint* and httpMethodConstraint that contains a list of nested *@HttpMethodConstraint* annotations. The *@HttpConstraint* is used to represent a security constraint to be applied to all HTTP methods while the second is used to define per-HTTP method constraints. Both *@HttpConstraint* and *@HttpMethodConstraint* contain as attributes a list of allowed roles (*allowedRoles*), the data protection requirements (equivalent to the *user-data-constraint* element in the *web.xml*) and the behavior when the list of allowed roles is empty. Finally, as in the web descriptor, security roles may be declared with annotations by using the annotation @DeclareRoles. E.g., *@DeclareRoles("employee")*.

Listing 3. Annotated Servlet.

```
1 @WebServlet(name = "RestrictedServlet", urlPatterns ={"/restricted/employee/*"})
2 @ServletSecurity((httpMethodConstraints = {
3 @HttpMethodConstraint(value = "GET", rolesAllowed = "Employee")
4 transportGuarantee = TransportGuarantee.None)})
5 public class RestrictedServlet extends HttpServlet { ... }
```

Both alternatives can be used at the same time and the final security policy is the result of combining the security constraints specified with both mechanisms. When in conflict, the constraints specified in the *web.xml* file take precedence and moreover, constraints defined by using annotations may be completely ignored if so is established in the *web.xml* descriptor (the *metadata-complete* parameter set as true) which may clearly create confusing situations to non-experts.

Besides, access-control policies defined with the mechanisms described above may contain inconsistencies (rules stating different access actions for the same resource). These inconsistencies are resolved by execution semantics such as rule precedence and combination algorithms as defined in the Java EE Servlet specification. Unfortunately, while this process eliminates inconsistencies in the policy, it may introduce typical access-control anomalies such as shadowing and redundancy [4–6], along with other misconfigurations particular to the Java EE access-control (e.g., not declaring all HTTP methods in a given access constraint). As these anomalies may lead to unexpected security and/or availability problems, the automatic detection of their presence in a given web security policy appears as a critical requirement for the security assessment.

Note that, as mentioned above, fine-grained access-control constraints requiring context information may also be defined in the Java EE framework. These constraints cannot be declaratively defined and require the use of programmatic security. Examples would be constraints checking that a user holds two roles or that a connection may only be accepted in specific time slots. We leave the analysis of these fine-grained programmatic constraints as a future work.

Q: In your opinion, how critical are security aspects in the development of web application(s)?

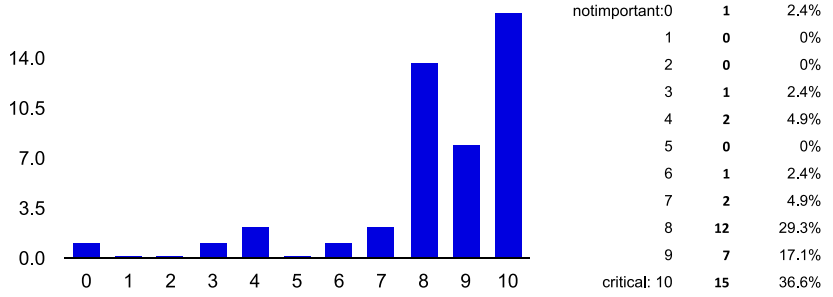

Fig. 1. How critical are security aspects in the development of web application(s)?.

3. Motivation

In order to assess the relevance of security concerns in Java EE web development, we have elaborated an online survey addressed to Java EE web developers. From the set of answers, we have identified a number of important security properties developers find critical. We describe the details of our online survey in the reminder of this section.

3.1. Survey settings

Our survey has been directly sent to a pool of Java EE developers. We have obtained this pool by (1) searching projects using the Java EE technology in GitHub and (2) looking for the contributors (owner, collaborators, reporters) involved with those projects. Additionally, we have also openly published it in programming discussion forums and blogs. From this process, we have obtained a total of 41 answers, 87% of them corresponding to developers with at least 2 years of experience in Java EE web development.

3.2. Survey results

In the following, we will present a summary of the survey contents and obtained answers. The complete set of questions and answers can be found online.[2]

Developers were asked to rate the importance of security properties between 0, unimportant and 10, very critical. 88% of developers consider security aspects critical during the development of web applications, rating its importance between 8 and 10 (see Fig. 1).

As can be seen in Fig. 2, 78% of developers frequently (values between 7 and 10) define access-control policies as part of the creation of a web application.

Developers were asked about the degree of difficulty of defining access-control policies, 0 being easy and 10 extremely difficult. 75% of developers feel that defining access-control policies, while not extremely difficult, is a challenging activity with most answer grouped between 5 and 8 (Fig. 3).

From the previously presented set of questions we can conclude that security concerns are considered critical for the majority of participant Java EE developers. Moreover, defining an access-control policy for their projects is both, a difficult and common activity. After these generic questions, developers were asked to evaluate the prevalence and severity of some common security errors.

We asked Java EE developers about the importance of having complete access-control policies, i.e., policies that completely cover the access space for a given resource. We also asked them to evaluate how likely it is to produce incomplete policies. From Fig. 4 we can see that having complete policies is seen as important for almost all developers (most answer above 5). Moreover, 29 out of 41 consider that producing incomplete policies is very likely.

W.r.t. redundancy in security policies (i.e., policy elements that can be removed without affecting the set of access decisions and that make the policy over-complex), from Fig. 5 we can conclude that Java EE developers consider, in general, redundancy important but less harmful than incomplete policies.

It is not uncommon in security policies to rely on implicit or external information in their contained rules while also allowing for a formal declaration of these elements (e.g., roles may often be taken from the environment whereas they may be directly declared in the security policy as well). We have asked Java EE developers about the importance of declaring all

[2] https://docs.google.com/forms/d/1K3hK4rY5yMkI3HbXOxRNfTHAQqR_tvLbf1s5RbvwaIA/viewanalytics