# 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.

Keywords：Model-driven engineering，Security，Reverse-engineering

# 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, 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 mis- conﬁgurations in 5th position on the top ten of most critical security ﬂaws, 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) policies are speciﬁed by writing constraints using a low-level rule-based language with two different textual concrete syntaxes and with relatively 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.). 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/n 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).

**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<1role-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>