# 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>

摘要

Java EE框架是web应用程序开发的流行技术，它为开发人员提供了定义访问控制策略的方法，以保护应用程序资源免受未经授权的信息披露和操作。不幸的是，这种安全策略的定义和操作仍然是一个复杂且容易出错的任务，需要专家级的知识来了解Java EE访问控制机制的语法和语义。因此，可以很容易地引入可能导致无意安全性和/或可用性问题的错误配置。针对这个问题，我们提出了一种基于模型的逆向工程方法，该方法自动评估一组安全属性，用于反向工程Java EE安全配置，帮助检测异常的存在。我们通过将原型工具应用于从GitHub中提取的真实Java EE应用程序的示例中，评估了我们的方法的有效性和针对性。

关键词:模型驱动工程、安全、逆向工程

1.介绍

Java EE是一种流行的技术，用于开发动态web应用程序(服务也是其他不太通用的框架的基础)，它向远程用户公开分布式信息和服务。在这种情况下，安全性是一个主要问题，因为构成web应用程序的web资源可能被许多用户通过不受信任的网络访问。因此，Java EE框架为开发人员提供了指定访问控制策略的方法，以确保web应用程序公开的资源的机密性和完整性。

不幸的是，尽管有这些安全机制，但是实现安全配置仍然是一个复杂的、容易出错的活动，需要高级专家来避免错误配置问题，这可能会造成严重的业务损失。事实上，开放的Web应用程序安全项目(OWASP)文档将Web应用程序错误的配置排列在最关键安全缺陷的前10位，因为它们很容易被利用，并且具有很强的业务影响。

对于访问控制和Java EE应用程序的具体情况，以及在应用程序代码中混乱的临时安全实现机制，基于角色的访问控制(RBAC)策略是通过使用具有两种不同的文本具体语法和相对复杂的执行语义的低级规则语言编写约束来指定的。具体地说，用户可以使用一组预定义的标记元素在XML web描述符文件中编写约束，直接在Java Servlet组件上编写注释(使用不同的语法和组织w.r.t. XML标记元素)，或者将这两种机制结合起来。然后，必须考虑约束和相应的执行语义之间的组合规则，以便理解有效执行的策略。

这种复杂性可能导致的异常和错误配置问题(例如,意想不到的规则的结果,意外的访问控制规则,之间的相互作用等)的影响不同的只是增加不必要的复杂性指定政策的引入等意想不到的行为给予资源访问未授权方或须授权的,正如第3节中所报告的在线调查的参与者所证实的那样。

为了解决这个问题，我们引入了逆向工程方法来自动检测Java EE web应用程序中的不一致和错误配置。首先，我们定义一个web应用程序必须满足的属性列表，以避免重要的异常，例如冗余(即:，说明不需要的约束，使策略过于复杂)和阴影(即:，规范的约束，从来没有执行过)。其次，我们提出了一种提取方法来解析给定web应用程序的安全配置(同时考虑到web描述符配置和Java注释)，并将其表示为针对Java EE web访问控制策略的特定于平台的安全模型(PSM)。然后，在该模型之上实现OCL查询和模型转换操作，以便在任何给定的Java EE web应用程序中自动评估定义的属性。此外，我们的工具会生成诊断报告，帮助识别对属性违规负责的源配置元素，从而帮助开发人员修复它们。

我们通过使用从GitHub(基于web的Git存储库托管平台)中提取的可公开的Java EE web应用程序来评估我们的方法的有效性。这个评估表明，相关的安全配置数量确实违反了我们推荐的属性，并且我们的工具能够成功地检测到这些违规行为。

论文的其余部分组织如下。第2节描述了Java EE的访问控制机制。第3节介绍了Java EE项目中安全性使用的动机调查。第4节描述了一些安全属性。第5节展示了如何从Java EE web应用程序中提取访问控制模型，而第6节详细介绍了我们对其属性进行评估的自动方法。第7节对我们的方法提出了一些附加的应用程序。第8节展示了评估结果，第9节给出了工具实现的详细信息。有关工作将在第10节讨论。最后，我们在第11节中给出结论和今后的工作。

2. Java EE web安全

粗略地说，在Java EE领域，当web客户端发出HTTP请求时，web服务器将请求转换为HTTP Servlet调用，以执行一些业务逻辑操作(Servlet和Java server Pages)。

在这个模式中，一个非常重要的需求是确保web应用程序管理的资源的机密性和完整性，因为它们可以被许多用户访问并遍历未保护的网络。从这个意义上说，Java EE框架提供了随时可用的访问控制工具。下面我们将简要描述Java EE为实现web应用程序中访问控制策略所提供的机制。

如前所述，Java EE应用程序通常是由jsp和Servlet组成的(jsp又被转换为Servlet)。这个层中的访问控制机制负责控制对这些元素的访问，以及其他可访问的构件(纯HTML页面、多媒体文档等)。这些访问控制策略可以使用两种不同的机制来指定:声明性安全性和程序性安全性，后者提供了需要用户上下文评估的高级访问控制的情况。尽管如此，Java EE规范建议在可能的情况下优先使用声明式安全性。

对于声明式访问控制策略，有两种选择:(1)在可移植的部署描述符(web.xml)中编写安全约束，以及(2)作为servlet Java代码的一部分编写安全注释(注意，不是所有的安全配置都可以通过注释来指定)。

代码清单1显示了在web中定义的安全约束。xml描述符。它包含三个主要元素:一个web资源集合，指定受安全约束影响的资源的路径和用于该访问的HTTP方法(在本例中为/限制/雇员/n路径和GET方法);一个自定义约束声明了哪些角色(如果有的话)可以访问资源(仅在示例中的角色Employee)和一个用户数据约束，该约束决定了用户数据必须如何从web应用程序和web应用程序(例如，任何类型的传输都被接受)。此外，尽管它不是强制的，但是web描述符可能包含角色声明(参见代码清单2)。

**代码清单1 在web.xml中的安全约束**

<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>

**代码清单2 在web.xml的角色声明**

<security-role>

<role-name>Employee</role-name>

</security-role>