# Model-based Analysis of Java EE Web Security Configurations

Salvador Martínez
AtlanMod Team
(Inria, Mines Nantes, LINA)
Nantes, France
salvador.martinez@mines-nantes.fr

Valerio Cosentino
AtlanMod Team
(Inria, Mines Nantes, LINA)
Nantes, France
valerio.cosentino@mines-nantes.fr

Jordi Cabot
ICREA - UOC
Barcelona, Spain
jordi.cabot@icrea.cat

## ABSTRACT

The widespread use of Java EE web applications as a means to provide distributed services to remote clients imposes strong security requirements, so that the resources managed by these applications remain protected from unauthorized disclosures and manipulations. For this purpose, the Java EE framework provides developers with mechanisms to define access-control policies. Unfortunately, the variety and complexity of the provided security configuration mechanisms cause the definition and manipulation of a security policy to be complex and error prone. As security requirements are not static, and thus, implemented policies must be changed and reviewed often, discovering and representing the policy at an appropriate abstraction level to enable their understanding and reenginering appears as a critical requirement. To tackle this problem, this paper presents a (model-based) approach aimed to help security experts to visualize, (automatically) analyse and manipulate web security policies.

## Keywords

Security, Access-control, Reverse-engineering

## 1. INTRODUCTION

Java EE is a popular technology of choice for the development of dynamic web applications, serving also as foundational layer for other less general purpose frameworks. Java EE facilitates the exposure of distributed information and services to remote users. In this scenario, security is a main concern [16], 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 provide developers with the tools to specify access-control policies in order to assure the confidentiality and integrity properties of the resources exposed by web applications.

Unfortunately, and despite the availability of these security mechanisms, implementing security configurations remains a complex and error prone activity where high expertise is needed to avoid inconsistency and misconfiguration issues, that could inflict critical business damages. As the resources managed by the web application can be accessed by many users and traverse unprotected networks. unintended data disclosures may lead to important losses both in terms of money and reputation.

For the concrete case of access-control in Java EE applications, declarative role-based access-control (RBAC) [20] policies for web applications are specified by writing constraints using a low-level rule-based access-control language with two different textual concrete syntax's and with relatively complex execution semantics. Concretely, the user can either:

1. write constraints in the XML web descriptor file by using a set of predefined tag elements.
2. write annotations (with a syntax and organization different w.r.t. the XML tag elements) on the Java Servlet components.

Both mechanism can be used in the same Java EE application, so that combination rules are needed to obtain the final security policy.

In this context, discovering and understanding which security policies are actually being enforced by a given web application comes out as a critical requirement. The main challenge for this discovery process is bridging the gap between the low-level, scattered policy representation, and a higher-level, easier to understand and manipulate, comprehensive representation. In order to tackle this problem, we provide, for the case of Java EE web applications:

1. a linguistic unification of the two declarative constraint specification mechanisms provided by the Java EE framework, so that the contributions of both mechanisms can be analysed by the same procedures and tools.
2. a reverse engineering process that extracts the security configurations implemented by using the diverse Java EE mechanisms and integrates them to a security model conforming to a proposed web security metamodel.
3. a demonstration of applications and benefits of using an integrated model-based representation given this enables the reuse in the security domain of the large number of model-driven techniques and tools that can be applied to visualize, analyze, evolve, etc the model.

Our approach complements existing Java reverse engineering works that skip security aspects [3, 5].

We demonstrate the feasibility of our approach through a prototype tool implementation and its application to a sample of real Java EE applications available GitHub.

The rest of the paper is organized as follows. Section 2 introduces the diverse access-control mechanism provided by the Java



*MiSE'16, May 16-17 2016, Austin, TX, USA*



ISBN 978-1-4503-4164-6/16/05...$15.00

DOI: http://dx.doi.org/10.1145/2896982.2896986

EE framework while Section 3 introduces our approach to extract an integrated model representation from them. Section 4 describes a number of relevant application scenarios. Evaluation of the approach is provided in Section 5, followed in Section 6 by a brief description of the tool support we provide to achieve automation. We conclude the paper by discussing related work in Section 7 and providing future research lines and conclusions in Section 8.

## 2. JAVA EE WEB SECURITY

Roughly speaking, in the Java EE framework, 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) that may directly answer or may, in turn, call Enterprise Java Beans (EJBs) in order to perform more complex business-logic operations. In this schema, a very important requirement is to assure the confidentiality and integrity of the resources managed by the web application. 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 for web applications.

### 2.1 Access-control

Java EE applications are typically constituted by JSPs and Servlets. The access-control mechanism in place are in charge of controlling the access to these elements along with any other web stored and accessible artifact (pure HTML pages, multimedia documents, etc.) [1] Two flavors are available to specify security policies at this level: declarative and programmatic security, being the latter provided for the cases where fine access-control, requiring user context evaluations, is needed.

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: 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>
```

Let's consider a corporate web application with a restricted area for employees and where a role *Employee* is defined. Listing 1 shows a security constraint defined in a *web.xml* descriptor that restricts the access (when using the HTTP method GET) to the employee area to users holding the *Employee* role.

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* (i.e., any kind of transport is accepted) in the example.

The equivalent security constraint, defined by means of annotations is shown in Listing 2. 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.

**Listing 2: Annotated Servlet**

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

### 2.2 Policy and rule combinations

Both aforementioned declarative 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. However, in case of conflicts, 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 using rule precedence, execution semantics 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 [8], along with other misconfigurations particular to the Java EE access-control.

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. Notice also that the Java EE specification recommends a preferential use of declarative security whenever possible.

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

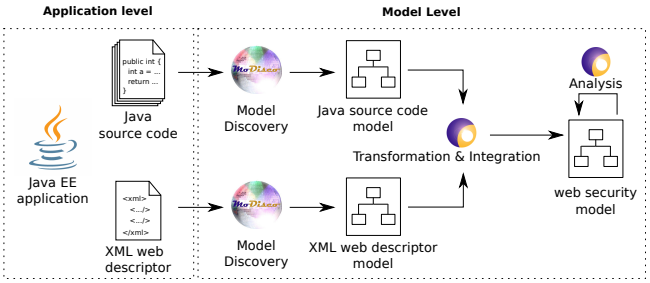


**Figure 1: Java EE web application analysis approach**

# 3. APPROACH

Our approach, depicted in Figure 1, collects the security information contained in a Java EE application by extracting it towards model representations. Then, it unifies and integrate them in a domain-specific model (by the means of model transformations), so that model manipulations, query operations, etc., can be later applied over it in order to analyse and manage the security configuration. The domain-specific metamodel and the extraction process are described in the remainder of this section.

## 3.1 Metamodel

Previous to the extraction process, the definition of metamodels able to accurately represent the information contained in the configuration source files is required. The Servlet Access-Control Security metamodel (hereinafter referred to as Servlet Security metamodel) depicted in Figure 2 will be used for that purpose. Both annotations and XML security definitions share similar semantics and therefore can be mapped to this metamodel abstract syntax allowing us to go directly from their low level model representations (Java model and XML model, already available in the MoDisco framework [3]) to our security metamodel without the need for defining intermediate specific security metamodels for their representation.

The Servlet Security metamodel allows to handle security roles (*SecurityRole* class) and security constraints (*SecurityConstraint* class) in a given web application. The former specifies the security roles defined by the security policy. The latter is used to represent access control constraints, coming from XML web descriptors or Java annotations (*source* attribute), on a collection of web resources (*WebResourceCollection* class). Each web resource collection identifies the resources (*urlPattern* attribute) and HTTP methods (*HttpMethod* class) to which a security constraint applies. It is important to note that if a HTTP method is defined as omitted (*omission* attribute), the security constraint applies to all methods except the omitted method.

The set of roles participating in the security constraint for a collection of web resources are grouped under an authorization constraint (*AuthConstraint* class). The absence of an *AuthConstraint* element in a given constraint represents public access whereas total preclusion is represented by its presence with no role association. A security constraint can be also associated with a given data protection requirement (*UserDataConstraint* class). This data protection requirement can be defined as: NONE indicating that the container must accept requests when received on any connection, INTEGRAL establishing a requirement for content integrity and CONFIDENTIAL, establishing a requirement for communication confidentiality.

## 3.2 Extraction Process

Once a metamodel able to describe Java EE access control definitions is available, we can extract the access-control information defined for the web application. The extraction process, shown in Fig.1 consists of three steps, namely model discovery, transformation & integration, and analysis.

The model discovery step relies on Modisco that parses the Java source code and XML web descriptor to obtain the corresponding model representations, namely, a Java model and an XML model. By doing so we move from the technical space of source code annotations and XML files to that of the modelware realm. The obtained models are then manipulated and the security information contained are mixed in the transformation & integration step to obtain a model conforming to the proposed Servlet security metamodel (web security model in the figure). This transformation & integration step rely on ATL[9], a model transformation language. The ATL code for the transformation & integration includes 19 rules and 31 helpers. 10 rules and 6 helpers deal with the information contained in the XML descriptor, while the remaining 9 rules and 25 helpers take care of the security information within the source code model.

The aforementioned process allows us to ease the manipulation operations required to analyze the policies w.r.t. a direct manipulation of the constraints separately defined in the XML and source code annotations with more basic techniques (like text manipulation or xslt transformation). Concretely, it will allow us to 1) use well-known model-drive tools and frameworks and 2) treat security information in an uniform way, disregarding whether it was specified by using XML or annotations. Notice however that these models lay in the same abstraction level of the original configurations and thus, no information-loss is produced in the extraction process.

Listing 3 shows an ATL rule, part of the transformation to extract access-control policies, that maps annotated Servlets to *SecurityConstraint* entities.

Notice that in order to make transparent to the users the low-level technical details needed to use MoDisco and ATL, we have developed a tool under Eclipse, that allows the user to select a given Java EE project and its web descriptor via a simple GUI to derive the corresponding Servlet security model. Such tool can be employed as support for different kind of applications.

**Listing 3: ATL rule to map security annotated servlets to security-constraints**

```
1  lazy rule createSecurityConstraint {
2    from
3      s : JAVA!Annotation
4      using {
5        servletAnnotation : JAVA!Annotation =
6        s.getContainerAnnotation('ServletSecurity');}
7    to
8      t: SEC!SecurityConstraint (
9        webResourceCollection <-
10          thisModule.createWebResourceCollection(
11          servletAnnotation.getWebServlet, s),
12       authConstraint <-
13         if s.getEmptyRoleSemantic = 'PERMIT' then
14           thisModule.createAuthConstraint(s, true)
15         else
16           thisModule.createAuthConstraint(s, false)
17         endif,
18       source <- #CODE)
19 }
```

The last step of our approach, namely, the analysis and manipulation of the extracted security configurations will be demonstrated in the next Section by the description of a series of application scenarios.

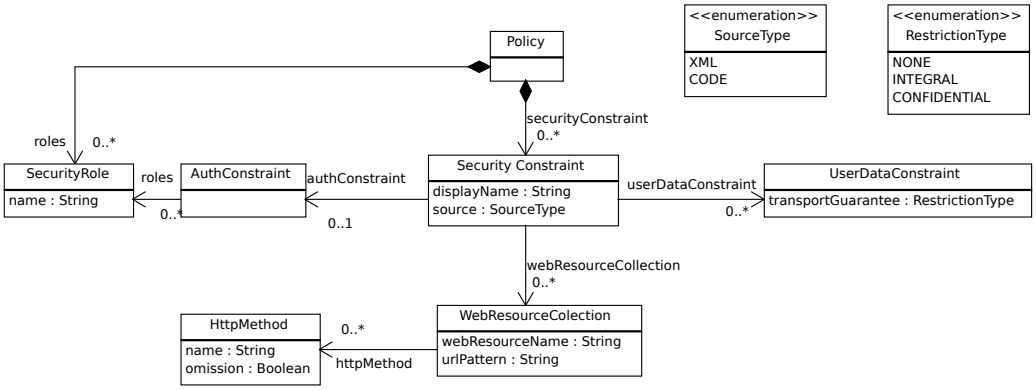


**Figure 2: Servlet Security metamodel**

# 4. APPLICATIONS

Having all the access-control information of a Java EE web application gathered and represented in the form of an integrated model corresponding to our Servlet Security metamodel enables the reusability of a wide range of proved off-the-shelf model-driven tools and techniques to derive interesting analysis applications. In the following, we will discuss a few of them, focusing in the calculation of queries and metrics and briefly discussing the rest.

## 4.1 Metrics

One of the most immediate applications of our metamodel is to use query languages such as OCL or IncQuery [2] to perform query and metric operations about the information in our model. From simple queries such as listing all the resources reachable by a given subject to more complex operations as detecting equivalent roles (a bad smell for a possible break in the least privilege strategy), our model representation, model extraction and integration approach ease the analysis of the security of a JEE application by reducing the complexity of performing such operations w.r.t. working with the original configuration mechanisms and making the needed integration by hand.

As a example, we will show here the definition of a metric that counts the freely accessible resources.

**Counting open-access resources query:**

Resources in Java EE applications can be either visible to a reduced set of users having certain roles or being freely accessible to everybody, no matter their roles. A manual inspection of the application to discover the open-access resources can be a tedious task, however leveraging on the proposed Servlet Security metamodel, we can easily define a metric to detect and count this kind of resources.

**Listing 4: metric query to count the open-access resources**

```
1 query unreachableResources =
2   SEC!SecurityConstraint.allInstances()
3   ->select(sc | sc.authConstraint.oclIsUndefined())
4   ->collect(sc | sc.webResourceCollection)->flatten()
5   ->collect(wrc | wrc.urlPattern)->flatten()
6   ->collect(up | up.value)->flatten()
7   ->asSet()->asSequence()->size();
```

Listing 4 shows a possible OCL query able to detect open-access resources. All the security constraints that do not define an authorization constraint (*authConstraint*) are selected, since they do not enforce any access restriction on the resources declared within the constraint (see lines 2-3). Then, the URLs within each resource contained in the selected constraints are collected (see lines 4-6) and finally counted (see line 7).

## 4.2 Correctness

Correctness properties of the security definition can be checked by performing queries over our extracted security model. The detection of misconfigurations (e.g., broken resource accessibility, leaving resources unreachable by any role) is thus enabled helping the developers to easily verify their security configurations and find possible errors. As an example, in the following we will give details about checking one very important security property for Java EE security configurations, e.g., the completeness property.

**Completeness property & query:** When defining an access-control constraint over a given resource, developers must assure they specify the policy for every way of accessing the resource as failing to do so may left important information unexpectedly available (or unexpectedly precluded). This is specially true for the Java EE framework due to the specific semantics of its security security policies.

If a HTTP method is named in a security constraint, all other standard HTTP methods must be specified in the same or other security constraint matching the same set of requests. Otherwise non-named HTTP methods will be considered as uncovered giving unconstrained access to them.

An example of a security constraint violating this property is shown in Listing 1. In the example, a constraint is defined for the GET method, stating that only users holding the *Employee* role are allowed. However, nothing is said about any other HTTP methods. In the absence of other constraints with identical URL pattern, this constraint will be matched, and thus, unconstrained access will be granted to any HTTP method different from GET.

In Listing 5 we show the query that calculates the completeness property.

The OCL code defines two main variables, *HTTP_METHODS* that lists all the standard HTTP methods, *ALL_HTTP_METHODS* that collects all the elements in the input model which class is *HttpMethod*. This collections of elements are used then to check if each *SecurityConstraint* element, containing a given instance of a *HttpMethod* element, includes all the other standard HTTP methods or if they are listed in another *SecurityConstraint* element with same *URLpattern* element value (directly or by the use of the tag method_omission).

The evaluation of properties over an extracted model is fully au-

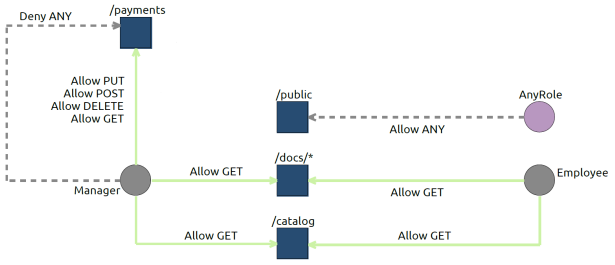


**Figure 3: Sirius Policy Visualization Viewpoint**

tomated. We have specified our properties as OCL queries. The execution of the OCL Query over the model returns the model elements that are violating a property, if any.

**Listing 5: OCL query to calculate the completeness property**

```
context HttpMethod inv:
let HTTP_METHODS : Sequence(OclAny) =
  Sequence{'OPTIONS','GET','HEAD','POST',
  'PUT','DELETE','TRACE','CONNECT'} in
let ALL_HTTP_METHODS : Sequence(PSM!HttpMethod) =
  PSM!HttpMethod.allInstances() in
let httpMethodsToCheck : Sequence(String) =
  if self.omission then
    HTTP_METHODS->select(m | m = self.name)
  else
    HTTP_METHODS->reject(m | m = self.name)
  endif in
  let selfUrlPatterns : Sequence(PSM!UrlPattern) =
  self.refImmediateComposite().urlPattern in
  selfUrlPatterns->iterate(sup ; output : Boolean = true
      ↪|
    let declaredHttpMethods : Sequence(PSM!HttpMethod) =
        ALL_HTTP_METHODS->reject(hm | hm = self)
        ->select(hm |
            ↪hm.refImmediateComposite().urlPattern
        ->exists(up | sup.value = up.value)) in
    if declaredHttpMethods->isEmpty() then
      false
    else
      output and httpMethodsToCheck
      ->forAll(m | declaredHttpMethods
      ->exists(dhm | dhm.name = m))
    endif
```

## 4.3 Visualization

Graphical information is often easier to grasp at a glance than textual one. In this sense Model-driven workbench generation tools as Sirius[2] allow the definition of different viewpoints for a given domain-specific language so that different graphical representations can be obtained without the need of manipulating the source model to create the view. This way, we can obtain general representations, summarizing the access-control policy of a given application, or more detailed representation showing, for example, the detailed security information related with a given resource or role. Figure 3 shows the visualization obtained for a simple web application where web resources, roles and the defined constraints are visible (note that, for readability, some relations are omitted).

## 4.4 Pivot representation

Our metamodel and extraction process can be also used as a pivot representation in order to build bridges towards other (model) representations. Translations from our metamodel to more generic access-control languages like SecureUML or XACML [17] would enable the reuse of the vast amount of research performed in the general field of access-control analysis. Notably, formal validation and verification techniques[14], along with change impact analysis[6] for access-control policies would become available.

Moreover, by integrating our extraction process and model in Java EE reengineering frameworks we can use the approach presented here to 1) extract the access-control configurations of a web application 2) apply model analysis and manipulation techniques to find a correct possible existing misconfigurations or implement security refactorings and 3) regenerate the web application with a correct security configuration.

# 5. EVALUATION

In order to demonstrate the feasibility and efficacy of our approach we have constructed a prototype tool and conducted an evaluation on a sample of real Java EE projects extracted from GitHub.[3]

From the collected projects, we performed two filtering steps aimed at removing toy projects. First, we manually filtered out those projects that included one of the following words in their name or path: *test*, *sample*, *demo*, *example*, *tutorial*, *training*, *exercise*, *lesson*. Second, from the projects left, we derived the model representation of their security configurations, and we filtered those that declared less than 5 security constraints, obtaining in this way a sample composed of 16 projects.

We have conducted three analysis on the previous sample: first, we have applied our tool to obtain model representations of their security configurations and measured the time to perform the task, secondly, we have automatically evaluated the correctness query presented in Section 4. Finally, and in order to check the existence of false positives, we have manually checked this property on the list of projects.

Table 1, showing for each Java EE project the number of constraints composing its security configuration, and for the security correctness query, if an anomaly is detected manually and/or automatically,summarizes the obtained results.

Table 1 also shows the time taken by our tool to extract models out of the corresponding Java EE web projects (gen.) and to evaluate (eval.) our security correctness property over them. The time for model-extraction depends directly on the number of Java files whereas the time for calculating the properties depends on the number of security rules in the policy and on the complexity of the interactions between rules. Both times are reasonably low for an offline analysis and concretely, the time of analyzing the properties is always below one second.

This evaluation shows that our approach: 1) have a good performance and thus is suitable to be used in real scenarios 2) the relevant security information is not lost in the process and thus, can be analysed as if it was manually done over the original configuration artifacts. This is demonstrated by the absence of false positives or false negatives in the evaluation of the completeness property. The same results are obtained by both, our tool and by manual inspection.

# 6. TOOL SUPPORT

We provide tool support that covers two different aspects of our work. On the one hand, we have created a tool that relies on Selenium[21] to randomly sample GitHub projects (to be then analyzed

[2]http://eclipse.org/sirius/

[3]The sample is obtained using the GitHub Search API. From an initial set of 1000 projects we randomly selected two groups of 50 projects containing security constraints. One group consisted of projects including a: 1) XML file descriptor with at least a security constraint 2) a size greater than 5000 KB, 3) contained in a WEB-INF folder. The second consisted of projects containing security annotations in Java source code.

| Project Name | Files | Cons-traints | Comple-teness | Time (sec.) gen. | eval. |
|---|---|---|---|---|---|
| ConnectrStage | 126 | 8 | | 7.26 | 0.22 |
| Dar | 143 | 11 | ■□ | 10.5 | 0.26 |
| HrsWeb | 4 | 5 | ■□ | 0.41 | 0.13 |
| Itrust | 457 | 10 | | 9.39 | 0.07 |
| Jersey | 27 | 6 | | 0.74 | 0.09 |
| PlanetsSuite | 1260 | 6 | | 28.8 | 0.08 |
| slim3-twitter | 48 | 5 | | 1.09 | 0.10 |
| Adware | 174 | 8 | | 3.86 | 0.09 |
| Avicena | 163 | 8 | | 3.29 | 0.07 |
| BridgeMonitoring | 90 | 8 | ■□ | 1.90 | 0.09 |
| Docked_Gae | 96 | 7 | | 2.39 | 0.06 |
| eTrade_Web | 22 | 14 | | 0.31 | 0.06 |
| IrssiNotifierWP7S | 21 | 11 | | 0.52 | 0.10 |
| Smsr | 435 | 10 | | 19.1 | 0.08 |
| TribalEducationN | 38 | 6 | ■□ | 1.07 | 0.07 |
| uratalk | 8 | 5 | | 0.23 | 0.08 |

■ automatic detection
□ manual detection

**Table 1: Evaluation Results**

for security issues). On the other hand, a tool has been developed as plugin under the Eclipse[22] platform with the purpose of automatically extract security models out of existing configurations. Both tools are available on GitHub[4].

# 7. RELATED WORK

Several other works tackle the problem of extracting access control policies from dynamic web applications. However, they are either focused in specific kinds of applications such as CMSs, or focused in recovering only the low level, inter procedural access-control enforcement. Our work focuses instead in recovering the security configurations of any Java EE web application covering the mechanisms provided by web application frameworks.

Concretely, in [1] the authors use dynamic analysis techniques to extract SecureUML models from PHP web applications. Also for PHP applications, in [15] inter-procedural privilege violations are detected by analyzing automata's extracted from the source code. In [7] a similar approach is studied for the Moodle Web Learning Platform. Finally, not focused in web applications but on Java code, the authors present an approach to perform interprocedural privilege analysis for Java applications [13]. All these works focus on extracting some security information by analysing the run-time behaviour of the application instead of doing a static analysis of the defined access-control policies responsible for that behaviour, as we do. More similar to the present work, in [19] the authors present a security metamodel and reverse engineering approach specially tailored to extract the security configurations of web Content Management Systems (CMS) (a similar language is presented in [23]). Instead, we do not restrict ourselves to a specific class of applications.

Regarding (web) security DSLs in [18] an RBAC profile for UML models is presented. UmlSEC, another extension of the UML language to high-level security requirements is introduced in [10] and used in [11] to model high-level security properties (requirements) for forward engineering purposes. More focused in web applications, web modeling approaches like [12] and [4] include specific security constructs, at least to specify users and roles and their access permissions on parts of the web navigation model.

We consider however that a better starting point for a security analysis requires a more specific security metamodel. Moreover, none of those frameworks and languages provide a reverse engineering approach to automatically obtain corresponding models out of existing Java EE security configurations. In this sense our approach could be a good complement to them by acting as a pivot model, as translations from our language to those other representations would be possible, for instance, as part of a reengineering horse-shoe model.

# 8. CONCLUSION

We have presented a model-driven reverse engineering approach to extract access-control policies from the diverse security configuration mechanisms of Java EE web applications. As a result of the process, a platform-independent access control model integrating in a single place the various partial security constraints is created, facilitating the comprehension and analysis of the policies. We have demonstrated the feasibility and pertinence of our approach by developing a proof of concept tool that we have applied on a set of real projects retrieved from GitHub.

As future work, we envisage several possible extensions. First of all, we believe that a forward engineering approach aiming at the automatic deployment of corrected-refactored policies would be a natural evolution of our approach. Then, we believe that extending it to include other security configurations is equally interesting. In that sense, we intend to extend our approach to include: 1) programmatic security constraints, 2) other sources of security configurations (e.g. from the underlying database system) to obtain a more complete version of the system security as a whole and 3) more complex frameworks built on top of Java EE (who typically provide their own security mechanism) as the Spring framework.

Finally, and following the path opened in Section 4 with the encoding and evaluation of a security property as an OCL query, we intend to extend our approach and tool to define and automatically analyse a set of important security properties beyond the correcteness property described here.

# 9. REFERENCES


[1] M. H. Alalfi, J. R. Cordy, and T. R. Dean. Recovering role-based access control security models from dynamic web applications. In *ICWE'12*, pages 121–136. Springer, 2012.

[2] G. Bergmann, Z. Ujhelyi, I. Ráth, and D. Varró. A graph query language for emf models. In *ICMT'11*, pages 167–182. Springer, 2011.

[3] H. Bruneliere, J. Cabot, G. Dupé, and F. Madiot. Modisco: A model driven reverse engineering framework. *IST*, 56(8):1012–1032, 2014.

[4] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (webml): a modeling language for designing web sites. *Computer Networks*, 33(1):137–157, 2000.

[5] V. Cosentino, J. Cabot, P. Albert, P. Bauquel, and J. Perronnet. A model driven reverse engineering framework for extracting business rules out of a java application. In *RuleML'12*, pages 17–31. Springer, 2012.

[6] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE'27*, pages 196–205. ACM, 2005.

[7] F. Gauthier, D. Letarte, T. Lavoie, and E. Merlo. Extraction and comprehension of moodle's access control model: A


[4]https://github.com/atlanmod/web-application-security

case study. In *PST'11*, pages 44–51. IEEE, 2011.

[8] H. Hu, G.-J. Ahn, and K. Kulkarni. Anomaly discovery and resolution in web access control policies. In *SACMAT'11*, pages 165–174. ACM, 2011.

[9] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: a model transformation tool. *Science of Computer Programming*, 72:31–39, 2008.

[10] J. Jürjens. UMLsec: Extending UML for secure systems development. In *UML'02*, pages 412–425. Springer, 2002.

[11] J. Jürjens, J. Schreck, and P. Bartmann. Model-based security analysis for mobile communications. In *ICSE'08*, pages 683–692. ACM, 2008.

[12] N. Koch and A. Kraus. The expressive power of uml-based web engineering. In *IWWOST'02*, volume 16. CYTED, 2002.

[13] L. Koved, M. Pistoia, and A. Kershenbaum. Access rights analysis for java. In *ACM SIGPLAN Notices*, volume 37, pages 359–372. ACM, 2002.

[14] Y. Ledru, N. Qamar, A. Idani, J.-L. Richier, and M.-A. Labiadh. Validation of security policies by the animation of z specifications. In *SACMAT'11*, pages 155–164. ACM, 2011.

[15] D. Letarte and E. Merlo. Extraction of inter-procedural simple role privilege models from php code. In *WCRE'09.*, pages 187–191. IEEE, 2009.

[16] X. Li and Y. Xue. A survey on server-side approaches to securing web applications. *ACM Computing Surveys (CSUR)*, 46(4):54, 2014.

[17] H. Lockhart, B. Parducci, and A. Anderson. OASIS XACML TC, 2013.

[18] T. Lodderstedt, D. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. In *UML'02*, pages 426–441. Springer, 2002.

[19] S. Martínez, J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and J. Cabot. Towards an access-control metamodel for web content management systems. In *ICWE MDWE'13 Workshop*, pages 148–155. Springer, 2013.

[20] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *RBAC'00*, pages 47–63. ACM, 2000.

[21] Selenium, a portable software testing framework for web applications. http://www.seleniumhq.org/.

[22] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[23] V. Svansson and R. E. Lopez-Herrejon. A web specific language for content management systems. In *OOPSLA DSM Workshop, Montréal, Canada*, 2007.