# Semantic Smells and Errors in Access Control Models: A Case Study in PHP

François Gauthier, Ettore Merlo
Polytechnique Montréal, Canada
{francois.gauthier, ettore.merlo}@polymtl.ca

***Abstract*—Access control models implement mechanisms to restrict access to sensitive data from unprivileged users. Access controls typically check privileges that capture the semantics of the operations they protect. Semantic smells and errors in access control models stem from privileges that are partially or totally unrelated to the action they protect.**

**This paper presents a novel approach, partly based on static analysis and information retrieval techniques, for the automatic detection of semantic smells and errors in access control models. Investigation of the case study application revealed 31 smells and 2 errors. Errors were reported to developers who quickly confirmed their relevance and took actions to correct them. Based on the obtained results, we also propose three categories of semantic smells and errors to lay the foundations for further research on access control smells in other systems and domains.**

***Index Terms*—code smells, access control models, security, static analysis, information retrieval**

## I. Introduction

Every day, millions of people communicate, shop, bank, gather information, and perform numerous tasks using web applications. An increasing number of web applications now deal with private or security sensitive information. Such applications must implement access control mechanisms to protect the privacy of their users. Failure to do so results in access control vulnerabilities.

Access control vulnerabilities can take several shapes. For example, several studies [1], [2], [3], [4] target the identification of missing access controls, where sensitive operations are left unprotected. While missing access controls pose serious security threats, the focus of this paper is the identification of a more subtle, but no less relevant type of access control vulnerability: semantic smells and errors. Analogously to code smells [5] that reflect poor solutions to implementation problems, semantic smells reflect poor implementations of the semantic of an access control model. Semantic errors, on the other hand, are wrong implementations that must be corrected.

In the context of access control models, users are granted with privileges that allow them to perform certain actions. As such, a privilege should capture the semantic of the action it protects. In practice, however, the semantic of the privilege does not always clearly relates to the semantic of the protected action. In some cases, the semantic relationship is only partial and gives rise to semantic smells. In other cases, the absence of relationship leads to semantic errors. Listing 1 shows a semantic error in Moodle where the *user:update* privilege protects the download of user information instead of the update of a user account, as specified in the documentation.

```
/**
* script for downloading of user lists
*/
require_capability('moodle/user:update', CONTEXT_SYSTEM);
. . .
function user_download_xls($fields) {
. . .
}
```

Listing 1. Semantic error in Moodle. The enforced update capability is semantically unrelated to the download code it protects.

The focus of this paper is the automated identification of semantic smells and errors in access control models. The key insight behind our approach is the following: semantically related sections of source code usually perform similar actions and should therefore be protected by similar privileges. Otherwise, these sections of code may be affected by semantic smells and errors.

To our knowledge, we are the first to propose an approach for the automatic identification of semantic smells and errors in access control models. The main contributions of this paper are:

- A novel, statistically sound approach, based on static analysis, model checking and information retrieval techniques, to identify semantic smells and errors in access control models.
- A proof of concept that our approach can be applied to medium-size, open-source PHP applications, as shown by our case study of Moodle.
- Identification and classification of previously unknown semantic smells and errors in Moodle's access control model. Semantic errors were reported to developers who swiftly confirmed their relevance and took actions to correct them.

## II. Related Work

### A. Detection of Access Control Vulnerabilities

In [4], Dalton et al. proposed an approach, called Nemesis, that takes a user specified access control model as input and dynamically detects access control violations at runtime. From our experience, few developers provide such specifications.

To alleviate the need of manual workflow specifications, some researchers proposed techniques to automatically infer the security model of an application. In [2], Felmetsger et al. presented a tool to detect invariants from execution traces and report invariant violations at runtime. Semantic smells and errors induce *erroneous* invariants and cannot be detected with such strategies.

Recently, Son et al. [3] introduced RoleCast, a tool that statically detects missing access controls in Web applications. Their tool performs a control-flow *taint* analysis where unprivileged `INSERT`, `UPDATE` and `DELETE` database queries are reported as potential security flaws. From our experience, privileged actions in Web applications are not limited to database queries. For example, work by Gauthier et al. [1] targets the detection of *forced browsing* vulnerabilities, where no assumption is made about the nature of privileged actions.

All these approaches are solely based on static analysis and lack the necessary information to detect semantic smells and errors in access control models.

### B. Information Retrieval in Software Engineering

In recent years, information retrieval (IR) techniques have been used for a variety of software engineering tasks.

In [6], Asuncion et al. presented an approach, based on Latent Dirichlet Allocation (LDA) [7], to retrieve traceability links between software artifacts, such as documentation, source code, tests, etc..

In [8], Zhou et al. proposed a novel IR technique, called revised Vector Space Model (rVSM), for the identification of source code artifacts that are relevant to a particular bug report, with good results.

In [9], Grant et al. proposed the use of LDA for the reverse engineering of co-maintenance relationships. Their study revealed interesting co-maintenance patterns in several systems, written in different languages.

While their goals differ, these studies share a common denominator: they show that IR can efficiently identify code artifacts that are semantically related to text documents (e.g. a bug description) or to other code artifacts (e.g. co-maintenance relationships). In the context of this paper, we use LDA to report semantically related blocks of code for which the enforced privileges differ. To our knowledge, this is the first paper that addresses the detection of security flaws using IR techniques.

## III. Methodology

### A. Analysis Overview

As previously mentioned, the key assumption behind our work is that semantically related sections of source code usually perform similar actions and should therefore be protected by similar privileges. Otherwise, we assume there might be semantic smells and errors. In order to verify this assumption, we designed the following protocol:

1) Extract the mapping between privileges and source code using a model-checking based, inter-procedural, static analysis [10]. As a result, we obtain the list the privileges that are enforced at each statement of an application.
2) Perform unsupervised Latent Dirichlet Allocation [7] analysis to extract latent topics in the source code. It is assumed that sections of code that belong to the same latent topics are semantically related.
3) Identify the topics that are significantly associated to some privileges by the mean of logistic regression.
4) Infer the privileges that *should* protect each block of code based on the topics obtained at step 3. Report the blocks of code for which the enforced privileges differ from the semantically inferred ones.
5) Submit the observed discrepancies to developers.

### B. Step 1: Mapping Privileges to Source Code

As reported in [10] and [3], access control checks in Web applications are control-flow constructs. Consider the following snippet of PHP code:

```
if (has_capability("user:update")) {
  mysql_query($update_query);
}
```

Listing 2. Privilege check in Moodle. The update query is executed only if the check succeeds.

In this case, the update query is only executed if the access control check succeeds. We thus define the `mysql_query` statement as *protected* by the *user:update* privilege.

In previous work [1], [10], we presented a linear-time, interprocedural approach to statically map privileges to protected statements. In summary, a control-flow graph (CFG), annotated with access control checks, is first extracted from the source code. The CFG is then converted to multiple and independent model checking automata, each modeling one privilege. A custom-built model checker then processes each automaton and identifies the statements that are only reached by execution paths that contain an access control check for the corresponding privilege. Statements that are reached by both privileged and unprivileged execution paths are discarded. While similar results could have been obtained using regular data-flow analysis, we consider that model checking formalism simplifies the definition and implementation of our approach.

In this study, we post-processed the results to map privileges to *blocks* of protected statements: consecutive statements, enclosed in braces, that are all protected by the same privileges.

### C. Step 2: Topic Extraction with LDA

Originally developed for the analysis of natural language documents, Latent Dirichlet Allocation (LDA) [7], has been shown an efficient tool for program comprehension, bug localization and other software engineering tasks [6], [8], [9]. LDA probabilistically models text documents as mixtures of latent topics, where topics correspond to key concepts in the corpus of documents [7].

The first step toward the extraction of an LDA model from the source code is to build a collection of documents. Other studies usually define a document as a class in the source code. We adopted a slightly different approach. Experience taught us that privileges rarely protect entire classes. On the contrary, classes often comprise several blocks of code that are protected by different privileges. In order to accurately map privileges to documents, we defined documents as blocks of statements: consecutive statements enclosed in braces.

During LDA modeling, documents are treated as bags of words. The definition of "words" in a source code artifact varies from one study to another. In the context of this study, the term "word" refers to the identifiers (variable names, function names, etc.) in a block. Comments were discarded as they generally refer to whole classes or methods, not to specific blocks. Strings in Web applications often contain HTML, CSS or SQL code that mostly add noise and were also discarded.

LDA modeling was performed with the *GibbsLDA++* [11] tool with default parameters. When the modeling completes, a document-topic probability matrix is produced, showing the probability that a given document (block) belongs to a given topic. We then combine these block-topics probabilities with the previously extracted block-privileges mapping to identify the topics that are strongly associated to some privileges.

### *D. Step 3: Associating Topics to Privileges*

We now have two independent sources of information: on one hand, an exact mapping between privileges and blocks of protected statements and on the other hand a probabilistic mapping between blocks and latent topics. Our goal is now to identify those topics that capture the semantic of some privileges. We did so by the mean of logistic regression.

Logistic regression models the relationship between a dependent binary variable and one or more independent categorical or continuous variables. In its simplest form, a logistic regression models the influence of a single independent variable on a binary outcome:

$$\pi(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} \qquad (1)$$

where, (i) $X$ is the independent variable, (ii) $\beta_i$ are the model coefficients and (iii) $0 \leq \pi(X) \leq 1$ represents a value on the sigmoid regression curve. The closer $\pi(X)$ is to 1, the higher the likelihood that the outcome is true. In the context of this study, we investigated the relationship between binary privileges (protected or not) and latent topics.

Training a logistic regression models involves estimating the values of the $\beta_i$ coefficients in such a way to maximize the fit between the sigmoid regression curve and the observed outcomes. By the end of the training phase, a $p$-value is associated to each independent variable, representing the significance of the association between the variable and the outcome.

When performing logistic regression, one of the main prerequisite is for independent variables to be uncorrelated. LDA, however, provides no strong guarantee about the correlation between the extracted topics. In order to subtract our study from this statistical bias, we modeled the relationship between each privilege and each latent topic with a separate logistic regression model, resulting in 30,700 (100 topics $\times$ 307 privileges) logistic regressions.

On the other side, performing such a high number of statistical tests induces a higher probability of Type II errors, where totally random associations are misinterpreted as significant. All the $p$-values were thus corrected for multiple testing using the Bonferroni correction, where the $p$-value is multiplied by the number of statistical tests. Assuming a threshold of 0.05, it means that the original $p$-value, had to be $< 1.63 \times 10^{-6}$ to be considered significant after the Bonferroni correction.

TABLE I
SEMANTIC SMELLS AND ERRORS IN MOODLE'S ACCESS CONTROL MODEL

| | Name | # Occurrences |
|---|---|---|
| Smells | Implicitly granted privileges | 15 |
| | Semantically related privileges | 16 |
| Errors | Privileges used as a role | 2 |

### *E. Step 4: Inferring Privileges Based on Topics*

Once a logistic regression model is trained, it is possible to supply it with new observations in order to determine the likelihood of the outcome. For each topic that was found to be significantly associated to a privilege (corrected $p$-value $<$ 0.05), we performed privilege prediction on blocks, based on their topic probability.

The idea behind this procedure is to infer the privileges that *should* protect a block of code, based on semantic information. The output of the prediction is a value between 0 and 1, representing the probability that the block of code is protected by the privilege. In the context of this study, we fixed the prediction threshold at 0.95. In order to identify faulty access controls, we investigated the blocks of code for which the enforced privilege differs from the semantically predicted one.

## IV. RESULTS

As a proof of concept, we applied this methodology to Moodle, an open-source PHP course management system with an elaborate access control model. Moodle 2.3.2 counts 735,485 sLOC and 307 privileges.

Applying the proposed methodology, we obtained a list of 83 blocks for which the semantically inferred privilege differed from the enforced one. After a post-filtering step to eliminate embedded blocks of code, we obtained a list of 59 blocks.

Manual inspection of the results was completed in around two hours and revealed 31 smells and 2 errors that were further classified in three categories. Reported smells and errors were reviewed by a group of experts who assessed their significance. Table I details the categories and numbers of semantic smells and errors that were identified. Errors were reported to developers who quickly confirmed them and took actions to correct them. Overall, we identified two types of semantic smells and one type of semantic error:

**1. Implicitly granted privileges.** This smell captures the fact that some privileges are *implicitly* granted with other privileges. The *lesson:edit* and *lesson:manage* privileges are recurring examples of this smell in Moodle. As expected, routines that manage lessons are protected by the *lesson:manage* privilege and routines that edit lessons are protected by the *lesson:edit* privilege. However, edit routines often also perform manage related operations without being explicitly protected by the *lesson:manage* privilege. In that perspective, it seems that the *lesson:manage* privilege is *implicitly* granted with the *lesson:edit* privilege.

**2. Semantically related privileges.** In Moodle, this smell reflects a bad coupling between privileges and components. Consider this example: Moodle counts 82 view-related privileges. All of these privileges share a common semantic: they grant the right to "view", but for different components of the system. This coupling between privileges and components led to a steady increase of the number of privileges as Moodle evolved. In approximately three years, from version 1.9.5 to version 2.3.2, the number of privileges in Moodle grew from 217 to 307, mostly due to semantically related privileges.

**3. Privileges verified in place of a role.** This semantic error characterizes blocks of code that are protected by semantically unrelated privileges in place of a proper role verification. For example, in Moodle, the *user:delete* and *user:update* privileges are owned by administrators only. According to Moodle's documentation, these two privileges respectively grant the rights to update and delete a user account. However, we observed that they also protect semantically unrelated blocks of code that are responsible for the download of user information (see Listing 1) and the display of private data. Those two cases were submitted to developers (see [12] [13]) as potential bugs.

## V. Discussion

Implicitly granted privileges qualify as bad smells because of the confusion they cause among users, as shown by a discussion thread about the *lesson:edit* and *lesson:manage* privileges [14]. Interestingly, there exists some theoretical access control models that can explicitly handle such constraints between privileges [15].

Semantically related privileges increase the complexity of the access control model. Interestingly, semantically related privileges that are spread across several components can be straightforwardly modeled as cross-cutting concerns. Aspect-oriented approaches [16] seem well tailored to handle such type of semantic smells. Alternatively, since semantically related privileges are often granted to a very limited number of roles, many of these privilege checks could be replaced by a few proper role verifications.

Privileges used in place of a role suppose an equivalence relation between the privileges and the role. While such a relation might exist in the default configuration of the access control model, nothing prevents users from breaking it by altering the default model. Two occurrences of privileges used in place of a role were identified in Moodle.

Both cases were submitted to Moodle's bug tracker (see [12], [13]) as potential security issues. In order to minimize bias toward acceptance or rejection, both bug reports were filled in the most objective manner, deliberately omitting to mention university affiliation or the use of a research tool. For both bug reports, we obtained a response in less than 2 weekdays. A clarification discussion ensued and led to the acceptance of our claim that these pieces of code were inadequately protected. In one case, developers agreed to correct the bug in the next minor release. In the other case, they agreed to introduce a new, semantically related privilege in the next major release.

## VI. Conclusion and Perspectives

Access controls enforce protection by checking privileges that capture the semantic of sensitive operations. In this paper, we presented a novel approach for the identification of semantic smells and errors that can hinder the comprehension, increase the complexity and threaten the security of access control models.

The proposed methodology contrasts enforced privileges to semantically inferred ones and report discrepancies. Investigation of Moodle's access control model revealed 31 semantic smells and 2 semantic errors, distributed in 3 categories. Semantic errors were reported to developers who quickly confirmed their relevance and took actions to correct them.

The presented results are preliminary. In a further study, we plan to: (i) investigate several systems and domains to validate the proposed categories of semantic smells and errors and discover new ones, (ii) evaluate the accuracy of the logistic regression model through cross-validation experiments and (iii) test alternative approaches for topic extraction.

## Acknowledgments

The authors wish to thank Scott Grant for the inspiring discussions that sparked this work. This research is funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) under the Discovery Grants Program.

## References

[1] F. Gauthier and E. Merlo, "Fast detection of access control vulnerabilities in php applications," in *WCRE '12*. IEEE, 2012, pp. 247–256.

[2] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna, "Toward automated detection of logic vulnerabilities in web applications," in *USENIX Security*, 2010, pp. 143–160.

[3] S. Son, K. McKinley, and V. Shmatikov, "Rolecast: finding missing security checks when you do not know what checks are," in *OOPSLA '11*. ACM, 2011, pp. 1069–1084.

[4] M. Dalton, C. Kozyrakis, and N. Zeldovich, "Nemesis: preventing authentication & access control vulnerabilities in web applications," in *USENIX Security*, 2009, pp. 267–282.

[5] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[6] H. Asuncion, A. Asuncion, and R. Taylor, "Software traceability with topic modeling," in *ICSE '10*. ACM, 2010, pp. 95–104.

[7] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[8] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *ICSE '12*. IEEE, 2012, pp. 14–24.

[9] S. Grant, J. Cordy, and D. Skillicorn, "Reverse engineering co-maintenance relationships using conceptual analysis of source code," in *WCRE '11*. IEEE, 2011, pp. 87–91.

[10] F. Gauthier, D. Letarte, T. Lavoie, and E. Merlo, "Extraction and comprehension of Moodle's access control model: A case study," in *PST '11*. IEEE, 2011, pp. 44–51.

[11] P. Xuan-Hieu and N. Cam-Tu, "GibbsLDA++: A C/C++ implementation of latent Dirichlet allocation," 2007, http://gibbslda.sourceforge.net/.

[12] Moodle tracker, "MDL-36139," October 2012, http://tracker.moodle.org/browse/MDL-36139.

[13] ——, "MDL-36140," October 2012, http://tracker.moodle.org/browse/MDL-36140.

[14] Moodle forum, "Capabilities mod/lesson:edit and mod/lesson:manage," October 2012, https://moodle.org/mod/forum/discuss.php?d=89315.

[15] "Information Technology - Role Based Access Control ," ANSI and INCITS, Tech. Rep. 359-2004, 2004.

[16] D. Suvée, W. Vanderperren, and V. Jonckers, "JAsCo: an aspect-oriented approach tailored for component based software development," in *AOSD '03*. ACM, 2003, pp. 21–29.