Automated Verification of Role-based Access Control Security Models Recovered from Dynamic Web Applications

Manar H. Alalfi James R. Cordy Thomas R. Dean

School of Computing, Queen's University
Kingston, Canada
{*alalfi, cordy, dean*}*@cs.queensu.ca*

***Abstract*—This paper presents an original Model-Driven-Engineering (MDE) approach to support the verification and testing of security properties in dynamic web applications. Based on a previously recovered UML-based fine-grained security model, the approach begins by transforming the model into a Prolog-based formal model. The Prolog model is then checked to verify whether the application conforms to specified access control security properties. We demonstrate the use of our method on the popular open source bulletin board system PhpBB 2.0, in the context of three test scenarios: testing for unauthorized access, web application security maintenance, and web application re-engineering.**

I. Introduction

Software systems are becoming increasingly complex and interconnected, which increases the challenge in keeping them secure and reliable. Vulnerabilities have been found in critical infrastructure in healthcare, finance, energy and defence. Methods have been proposed to check for vulnerabilities such as SQL injection and cross-site scripting [27], but none attempt to detect broken access control.

Our previous work [5] presents a survey of models and methods for web application verification and testing. While many methods check static properties of web applications, and some check dynamic features, none is able to model the access control features of dynamic web applications. To address this lack, our own analysis framework [3] automatically recovers a UML Role-Based Access Control (RBAC) security model from dynamic web applications.

In this paper we use source transformation to transform this access control security model into a formal model implemented in Prolog. The TXL language [14], primarily used for source code transformation and design recovery, has recently been applied to model transformation [25, 21] and offers the scalability and flexibility we require. Although there are several available tools to transform from UML models into formal ones (e.g., UML2Alloy [10], XMI2SMV [11]), none is both complete and UML 2.0 compliant.

We have chosen Prolog to check for access control security conformance primarily for its ability to handle large scale industrial software models [29]. Opoka et al. [12] show that Prolog performance in model analysis is better than that provided by the Object Constraint Language (OCL). Störrle [29] provides a thorough comparison with other tools and approaches for UML model analysis.

The key contributions of this paper are:

- A scalable approach and tool to transform and formally analyze recovered web application security models using Prolog.
- A case study of the security analysis of a real production system, PhpBB 2.0, in three analysis scenarios.

We give an overview of our overall framework in Section II, and Section III presents the details of our approach. In Section IV we discuss the correctness and completeness of our analysis, and in Section V we demonstrate it on an example system, PhpBB 2.0. Section VII discusses related work. Finally, Section VIII concludes the paper and presents possible future work.

II. Security Analysis Framework

Our security analysis framework [3] (Figure 1) comprises two main phases. The first phase (Figure 1 (A,B)) uses an automated role mining process for the dynamic web application, in which roles and related information, such as permissions, constraints, and resources are identified. This process is based on a combination of static and dynamic analysis, using pattern matching to identify relevant security elements. In previous papers [2, 4, 6, 7, 8] we have described the approaches and tools that support this phase.

In the second phase (Figure 1 (C,D)), we use an MDE approach to construct a UML-based RBAC [1] security model from the security elements identified in the first phase. This phase is based on model transformation and composition, and makes use of structural and behavioural models recovered by the static and dynamic reverse engineering approaches of the first phase. The final step in this framework, Figure 1 (D), is the subject of this paper.

The RBAC security model recovered from a dynamic web application using our approach conforms to the SecureUML meta-model [9]. RBAC [1] was introduced in 1992 as a means to restrict access to the operations provided by a system based on the role of the user. RBAC has subsequently been adopted by several popular software platforms, such as the Java Platform Enterprise Edition (Java EE) [27]. In

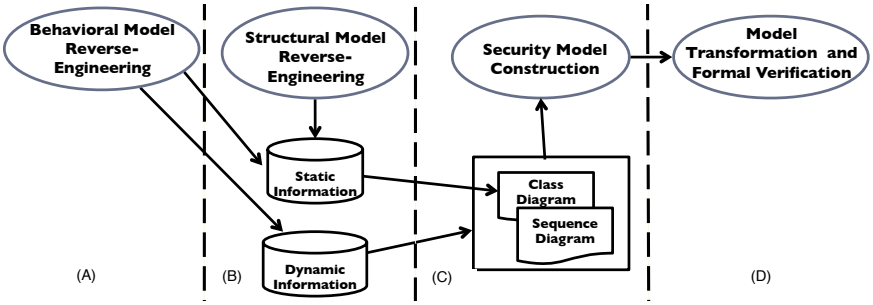

Figure 1. Overview of our dynamic web application security analysis framework

RBAC, a *permission* represents the right to perform a restricted operation, and a *role* represents a set of permissions that can be granted to users and groups of computer systems. RBAC permissions represent security-sensitive operations, as opposed to the security-sensitive data accessed by the system. When a user attempts a restricted operation, the user must be in a role that includes the necessary permissions.

SecureUML is an implementation of the Model Driven Security approach [9], a specialization of Model Driven Architecture. It explicitly integrates security aspects into the application's models and provides support for model transformation. The approach has been proposed to bridge the representation gap between the graphical languages used for specifying the application's design models, such as UML, and the textual languages used to specify the security models. It is built on a modular schema that comprises three basic elements: a language for security policy specification, a language for design model construction, and a dialect for defining integration points in the preceding languages. The abstract syntax for SecureUML is based on the RBAC meta-model. It defines a meta-model that extends RBAC with authorization constraints to enable formal specification of access control policies that depend on dynamic aspects of the system, such as the access date or values of the system's environment variables.

The modeling notation for SecureUML is based on a UML profile that uses UML stereotypes and tagged values to represent the abstract syntax elements in the meta-model schema. Users, groups, and roles are represented as classes with stereotypes $\ll User \gg$, $\ll Group \gg$ and $\ll Role \gg$ respectively, and permission is represented as an association class with a $\ll Permission \gg$ stereotype.

Figure 2 shows an example of a SecureUML model for a web forum application. The diagram shows two users in different roles who are permitted different sets of actions based on their roles. Bob, who is an anonymous user, is permitted to access the forum entities using read operations. So, Bob can access a forum via *ViewForum()*, read a forums' topics via *ViewTopic()*, read topic posts via *ViewTPosts()*, and can register in a forum. Alice, who is a registered user of the forum, can not only perform all the operations available to Bob but is also permitted write access to the forum. Thus, she can also reply to posts via *RPostReply()*, edit her own posts using *EditSelfPost()*, and so on.

III. Approach

In this paper, we present *SecureUML2Prolog*, a tool that automatically transforms a serialized SecureUML model into a formal Prolog model that can then be analyzed using Prolog queries. We define a set of mappings between SecureUML meta-model constructs and Prolog facts, and then use Prolog queries to check for role-based security concerns in the target application.

Our method is based on model transformations implemented in the TXL source transformation language [14]. We begin by defining a TXL grammar for the input SecureUML serialization, and a second ("override") grammar for our target Prolog format. The main grammatical form ([program] in TXL) allows both, but expects the Prolog part to be empty on input and full on output, and vice-versa for the SecureUML part. We used the TXL producer-consumer translation paradigm to make a set of transformation rules to "produce" the Prolog output while "consuming" the SecureUML input.

The transformation begins by parsing the SecureUML serialization to insure its conformance with its meta-model, with an initially empty Prolog output. It then replaces this entire input by constructing the Prolog output beginning with an empty set of Prolog facts and transforming it into the Prolog representation of the input SecureUML model.

As an example of the source transformation implementation, Figure 3 shows a TXL rule for transforming a SecureUML permission's action and condition into Prolog facts. The initial deconstructor pattern matches the SecureUML syntax to recognize an `ownedOperation` element and identifies the attributes of the action. The constructors `PermActionF` and `PermActionconst` then generate Prolog facts for the action and its condition, which are appended to the Prolog factbase using the replace-by transformation clause.

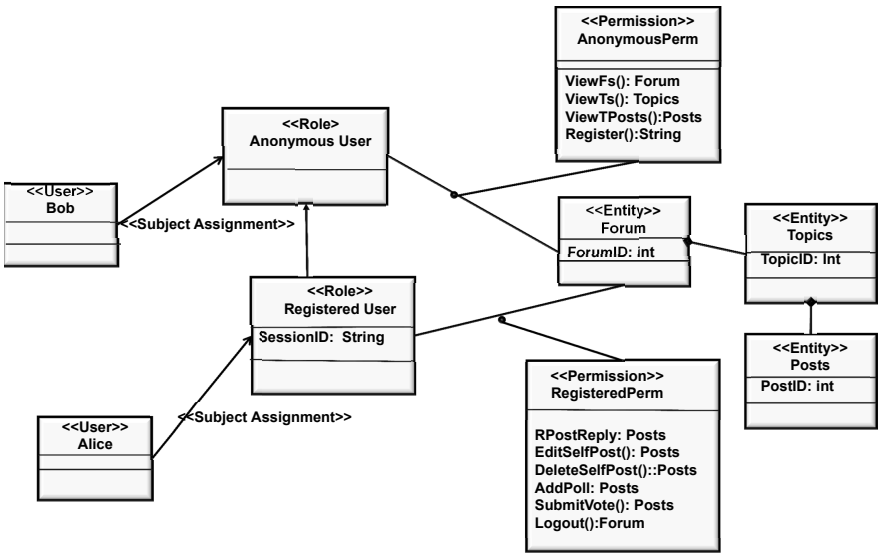

Figure 2. An example SecureUML model

```
function IsPermAction PermissionID [charlit] RoleID [id]
                      OwnedOperation [owned_operation]

   deconstruct OwnedOperation
      <ownedOperation 'xmi:type = "uml:Operation"
               'xmi:id = AttribID [stringlit]
               'name = OpName [stringlit]
               'precondition = PreconID [stringlit] >
         OwnedAttrib [XMItoken*]
      </ownedOperation>

   construct CharOpName [charlit]
      _ [+ OpName]

   construct PermActionF [Factdot]
      'permActionAssign (RoleID, PermissionID, CharOpName).

   construct PermActionconst [Factdot*]
      _ [IsPermActionConst OwnedAttrib PermissionID RoleID]

   replace * [Factdot*]
      PermActionsFactsSoFar [Factdot*]
   by
      PermActionsFactsSoFar [. PermActionF ]
                            [. PermActionconst]
end function
```

Figure 3. A sample TXL source transformation rule for generating permission facts from SecureUML model elements

Our tool generates the following categories of facts, based on mappings of each of the elements of the SecureUML meta-model (Figure 4) to Prolog.

- *Role facts*: The leftmost side of the secureUML meta-model Figure 4(A), presents the *Role* model element and two relations: *RoleHierarchy* and *userAssignment*. The model recovery aspect of our framework generates a separate SecureUML Role element for each role assigned to a specific user name for a specific session. A RoleNum attribute is added to all Prolog facts generated from the transformation process in order to bind them to the roles representing them. Currently, the RoleHierarchy relation is not automatically identified, so we do not map it in our target Prolog fact-base. The automated identification of the hierarchy relation is part of our planned future work. For Role elements of the SecureUML model, our tool generates facts for application roles and user assignments, such as:
  - *appRole (RoleNum, RoleName)* : Represents the application roles and takes as parameter the name of the role. Example: *appRole (1, "Admin")*.
  - *user (UserName)* : Represents the application users, and takes as parameter the user name or id. Example: *user ("Alice")*.
  - *userAssign (UserName, RoleNum)* : Represents the assignment of users to roles, and takes as parameter the role and user name. Example: *userAssign ("Alice", 1)*.
- *Permission facts*: The second part of the meta-model, Figure 4(B), has two model elements: the set of *Permissions* and *AuthorizationConstraints*, and three relations: *PermissionAssignment*, *ActionAssignment*, and *ConstraintAssignment*. These are mapped into the following Prolog facts:
  - *perm (RoleNum , PermissionID)* : Represents each application permission. Takes as parameter the role number and the application permission id. Example: *perm (1, 'phpbb_forumsPerm')*.
  - *permActionConst (RoleNum, PermissionID, ActionIDInCode, Constraint)* : Represents application permission constraints and takes as parameter the permission id, the action id in code and the action constraints. Example: *permActionConst (1, 'phpbb_forumsPerm', 366, 'forum_id = $forum_id')*.
  - *rolePermAssign (RoleNum, RoleID, PermissionID)* : Represents the application's role permission assign-

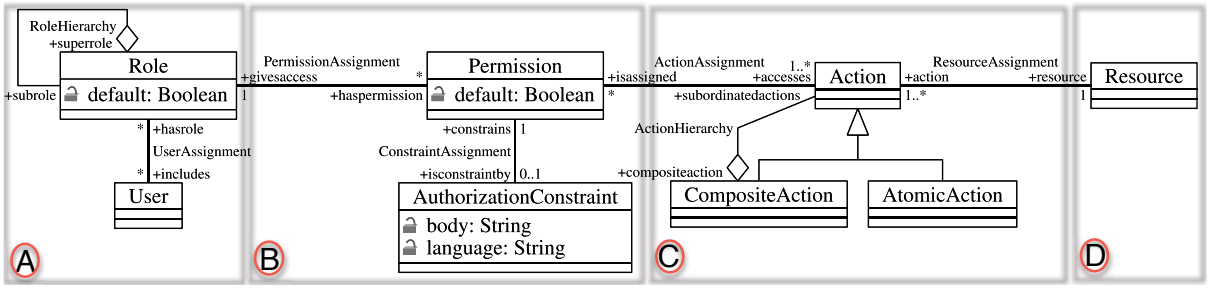

Figure 4. The SecureUML meta-model (adapted from [9])

ments. Takes as parameter the permission id and the role id. Example: *rolePermAssign (1, 'Anonymous', 'phpbb_forumsPerm')*.

- *permActionAssign (RoleNum, PermissionID, ActionIDInCode, ActionName)* : Represents the application's permission action assignments. Takes as parameter, the permission id and the permission atomic actions IDs and names. Example: *permActionAssign (1, 'phpbb_forumsPerm', 366,'Select (viewforum, forum_id = $forum_id)')*.

- *Action facts*: The third part of the meta-model, Figure 4(C), presents three model elements and their relations, representing the set of application *Actions*, where the granularity of the action (whole entity, attributes or association) and the type of access (read or write) is identified by *CompositeAction* and *AtomicAction* model elements. This category of security model elements is mapped into the following Prolog facts:
  - *permActionType (RoleNum, PermissionID, ActionIDInCode, ActionType)* : Represents the permission action types, that is, whether they are read or write operations. Example: *permActionType (1, 'phpbb_topicsPerm', 404, 'Update')*.
  - *permResourceAssign (RoleNum, permissionID, ResourceID)* : Represents application's permission resource assignments. Takes as parameter the permission id and the resource id. Example: *permResourceAssign (1, 'phpbb_forumsPerm', 'phpbb_forums')*.

- *Secure resource facts*: The last part of Figure 4(D) shows the secured *Resource* model element. In the SecureUML notation, the representation of resources is left open, so that developers can decide later which elements of the system should be considered secure and which should have access constraints. These elements are defined using a dialect. In our model recovery framework, we identify and recover secure resources from the web applications under test and represent them as an ER diagram. Facts resulting from this category are the application's secure resources (entities), their attributes, and their relationships, such as:
  - *entity (RoleNum, ResourceID)* : Represents the application's secure resources. Takes as parameter the role number and the resource name. Example: *entity (1, 'phpbb_forums')*.
  - *entityAttrib (RoleNum, EntityName, Attrib)* : Represents the application's resource attributes. Takes as parameter the role number, the resource name and its attribute names. Example: *entityAttrib (1, 'phpbb_users','username')*.
  - *entityAssociation (RoleNum, entityName, AssociationID1, AssociationID2 (PK))* : Represents relations between the application's resources. Takes as parameter the role number, the resource name, and the relation's end names including key attribute(s). Example: *entityAssociation (1, 'phpbb_forums', 'phpbb_auth_access', 'phpbb_auth_access(forum_id)')*.

- *Dynamic and contextual facts*: This category of Prolog facts represents environment information, such as execution timestamps, server environment variable accesses, and so on from the recovered model. Facts from this category are represented as a set of parameters associated with the set of *Action* model elements presented in Figure 4(C).
  - *permActionTimeS (RoleNum, PermissionID, ActionID, Timestamp)* : Represents an application action timestamp. Takes as parameter the role number, the permission id, the Action id and the execution timestamp for the action. Example: *permActionTimeS (1, 'phpbb_forumsPerm', 366, 1249518435)*.
  - *permActionInPage (RoleNum, PermissionID, ActionID, PageID)* : Represents an action's location in the code, that is, which page it is associated with. Example: *permActionInPage (1, 'phpbb_forumsPerm', 366, 365)*.
  - *Return (RoleNum , ReturnAttribs, pageID)* : Represents attributes returned from interacting with the secured resources. Takes as parameter the role number, the returned attributes, and the id of the page that has the interaction. Example: *Return (1, 'u.user_id, u.username', 365)*.

IV. Correctness and Completeness of the Recovered Model

A question that arises when analyzing security aspects of an application using a formal model is the correctness and completeness of the model. One of the advantages of using a formal source transformation system for deriving and exploring security models from source code is that it is easier to reason about these properties of the tools. By contrast with a hand-coded analyzers implemented in Java or C, source transformation rules can be tested and verified piecewise.

Because source transformations are based on parsing technology, the well-formedness of the results is guaranteed. TXL transformation rules are simply incapable of producing a result that does not conform to the syntactic forms of the target grammar/meta-model. The question of the semantic soundness of the constructed security model is also made simpler using a source transformation technique. Rather than having to reason about an entire hand-coded analysis program all at once, each TXL source transformation rule can be considered independently of the others. Whether the entire transformation is correct then becomes just a question of whether the set of rules forms a complete transformation, which can be checked separately. In our system this question is addressed by separating the process into a sequence of separate source transformation steps. Because each step yields a concrete intermediate text file representation that the next step parses as input, erroneous or incomplete results of a step are typically caught immediately by the next step. For example, if the data model extracted from the web application's schema is missing anything, there will be unresolved links when integrating the models that will make this fact immediately evident in the next transformation step.

Using source transformation rules to analyze the schema, source code and behavioral models also assists in guaranteeing completeness. For example, the TXL parser syntactically identifies all references to the SQL database in the source code, and the transformation rule for analyzing them simply transforms them to an instrumented form. The question of whether we have missed any database interactions in the extracted model is therefore easy to evaluate, simply by counting the number of SQL interactions in the model and comparing it to the number identified by the parser in the source. Dynamic behavioral completeness is handled by including coverage counters in the instrumentation, implemented using the DWASTIC tool discussed in [7].

V. Experiment

In the remainder of this paper we demonstrate our framework on PhpBB 2.0, a popular internet bulletin board system, in the context of three scenarios. Our system currently works on web applications built using Apache, PHP and MySQL. Apache is the most deployed web server on the internet with a 65% market share [24]. PHP, used on more than 20 million websites [26], has been the most popular server-side scripting language for years. And MySQL is the fastest-growing database in the industry, with more than 10 million active installations and 65,000 daily downloads [23]. However, our approach is not tied to these choices, and could be applied to other technologies as well.

VI. Testing Scenarios

In this experiment, we examine three sets of users (roles): anonymous users, registered users and the administrator. The web application under test is explored twice, once to collect the application's form inputs, and a second time to do the actual navigation using automated form filling. We developed site exploration test cases to implement each of these automated navigations using WATIR, Web Application Testing In Ruby, an open-source family of Ruby libraries for automating website exploration [30]. We tailored versions of each exploration for each of the three roles.

The first exploration dynamically collects all of the application's form inputs, and creates an Excel spreadsheet with entries for each of these inputs. Each row in the spreadsheet includes fields for the input's ID, name, type, and value. Figure 5 describes the algorithm for this first site exploration. The resulting spreadsheet is used to later fill in the value fields for each form input.

The second WATIR site exploration test case uses the spreadsheet to fill the application's input forms while navigating the application. The navigation process is similar to that described in the algorithm shown in Figure 5, but instead of collecting form inputs, it searches for forms' input IDs in the navigated pages, and chooses values for the matched fields from the spreadsheet to populate the form.

A. Testing for Unauthorized Access

Many web applications try to implement access control polices using obscurity, where sensitive links are not presented to unauthorized users. This method of protection is not sufficient since attackers may be able to access the hidden URLs, knowing that sensitive information and operations lie behind them. In this testing scenario we show how our framework can be used to check for unauthorized access to application resources. Specifically, we check if an anonymous user can access any unauthorized content of PhpBB2.0 using the links that an administrator can see when accessing the same forum.

To check for this vulnerability, we first implement and run the WATIR test case that explores and dynamically collects all the links and form inputs in all the PhpBB 2.0 pages of the administrator role, and stores them in a spreadsheet. We excluded administrator visits to the administration panel itself, which includes the forum's management's tasks. Using the data collected in the spreadsheet, we ran a second WATIR test case that uses this data to navigate the forum as an anonymous user. The execution

Algorithm CollectFormInputs
Input: The URL of the home page of a web application
Output: A spreadsheet with the application's forms elements (ids, types, and values)

1. Create a new Spreadsheet: SpS
2. Create a new internet Explorer (IE) instance: IE_ins
3. Point the instance to the web application under test
4. ***collect_formElements()***
5. collect all links in the current page: Lns
6. ***navigate(***Lns***)***
7. **function** collect_formElements() {
8. **for each** Form element(FE): text field, button, radio button, hidden field in the current page **do** {
9. insert a new raw in SPS (FE name, FE ID, FE type, FE value)
10. **if** the test case is for a non Anonymous user **then{**
11. **if** (a username text field) **and**
12. (a password textfield) **exist** in the current page **then**
13. { username.value = registeredUser_username
14. password.value = registeredUser_password
15. **if** the login button **exist**
16. press the login button } } }
17. }% end function
18. **function** navigate (Lns: list of links)
19. **for each** link(I) in Lns **do**
20. { click link I
21. collect all links in the page generated from link I: Lns2
22. ***collect_formElements()***
23. Navigate(Lns2)
24. }

Figure 5. The **CollectFormInputs** algorithm for collecting the application's form input elements

trace collected for the anonymous user attempting to access administrator links and data is fed to our previous PHP2XMI [4] and WAFA [6] tools to generate a sequence diagram to reflect this behaviour. We then used our PHP2SecureUML [8] to generate the SecureUML models, finally, we used the SecureUML2Prolog tool of this paper to generate the corresponding formal Prolog fact base for the scenario.

The goal is compare the access control facts collected in this scenario with those collected for a legitimate visit of an anonymous user to the forum. In other words, we want to know what resources the anonymous user can access if he accidentally got to see and use the part of the user interface only intended to be available to users with higher privileges, such as registered users and administrators. To evaluate this we implemented and ran Prolog queries to compute the set differences in page access, server environment variable access, and access to application entities. Figure 6 shows some of the Prolog queries used to implement this goal, specifically to compute the set difference in page accesses.

The rule *AccessAdmin_Actions (RoleName1, RoleName2)*, where *RoleName1 = Anonymous* and *RoleName2 = Anonymous_Using_Administrator_Links*, executes another Prolog query, *rolePagesActionList()*, on each role and collects the results in *Bag* and *Bag2* respectively. The *remBag()* function

```
% computes role actions per page access role
PagesActions(RoleName, PageName, PageID, ActionID, RList):-
    appRole(RoleNum, RoleName),
    pageNameID(RoleNum, PageName, PageID),
    return(RoleNum, RList, ActionID, PageID).

% computes a set of actions lists for a specific role
PagesActionsList(RoleName, Bag):-
    setof([PageName, ActionID, RList, PageID],
          rolePagesActions(RoleName,PageName ,PageID,
                                   ActionID, RList), Bag).

% computes the actions set difference between two roles
AccessAdmin_Actions(RoleName1, RoleName2):-
    rolePagesActionsList(RoleName1, Bag),
    rolePagesActionsList(RoleName2, Bag2),
    remBag(Bag2, Bag, [], Result),
    printlists(Result).
```

Figure 6. Prolog rules to check for unauthorized access to application entities

then computes the set difference between *Bag2* and *Bag*, gathering the results in the *Result* set, and the *rolePagesActionList()* computes the set of Actions allowed for each role, grouped by page. Query results are presented in Table I.

From the results of the Prolog queries, we can see that PhpBB 2.0 reacts to attempted access to unauthorized pages in three different ways:

1) Attempted access to some pages is redirected to the login page. Examples are access to the *posting* pages (225), *privmsg* pages (257), *adminindex* page (794), and the *search* pages (324). The Last eight entries in Table I show these redirections to the login page.
2) Other pages are not properly protected, as they are not redirected to the login page. Instead, an error message is returned, such as "invalid session ID". Access to the *modcp* (145) pages, shown in Table I, are examples on this case.
3) Some pages are not protected at all, thus a guest user can access the pages and execute all actions associated with them. Access to *faq* (18), *index* (109), and the last *posting* (255) page are examples of this case.

In addition to pages, we can also use our Prolog model to test for potential anonymous user access to secured server environment variables, actions, entities and attributes of the web application. Table II shows the result of executing the Prolog set difference query on server environment variables, and Table III shows the result of executing the Prolog set difference query on the application actions, entities and attributes.

A particular risk of anonymous user unauthorized access to SQL statements, such as those shown in Table III, is enabling the unauthorized user to build on these privileges to launch another kind of attack, such as an SQL injection attack. In SQL injection, the attacker tries to modify the logic of SQL statements that accept unsanitized inputs to break the system and possibly take over site administration.

PID	P_Name	P_Parameter	Link Name	PhpBB React
18	faq	?mode=bbcode	BBCode	Full Access
109	index	?mark=forums	Mark all forums read	Full Access
126	login	?logout=true , sid=fd3c..	Log out [alalfi]	Full Access
145	modcp	?f=2 , sid=0f206abbbc..	moderate this forum	Error Msg
145	modcp	?f=2 , start=0 , sid=0f2..	moderate this forum	Error Msg
145	modcp	?mode=ip , p=5 , t=3 , sid=fd3..	View IP address of	Error Msg
145	modcp	?t=3 , mode=delete , sid=0f20..	Delete this topic	Error Msg
145	modcp	?t=3 , mode=lock , sid=fd3c29..	lock this topic	Error Msg
145	modcp	?t=3 , mode=move , sid=0f2..	Move this topic	Error Msg
145	modcp	?t=3 , mode=split , sid=fd3..	Split this topic	Error Msg
225	posting	?mode=delete , p=5 , sid=0f2..	Delete this post	Error Msg
225	posting	?mode=editpost , p=5	Edit/Delete this post	Error Msg
225	posting	?mode=newtopic , f=2	new topic	Forced login
225	posting	?mode=quote , p=5	Reply with quote	Forced login
225	posting	?mode=reply , t=3	post reply	Forced login
225	posting	?mode=smilies	view moreEmoticons	Full Access
257	privmsg	?folder=inbox , mode=read , p=1	Inbox	Forced login
257	privmsg	?folder=inbox , sid=fd3c..	Log in to check your private messages	Forced login
257	privmsg	?folder=outbox	Outbox	Forced login
257	privmsg	?folder=savebox	Savebox	Forced login
257	privmsg	?folder=sentbox	Sentbox	Forced login
257	privmsg	?mode=post	newpost	Forced login
324	search	?search_id=egosearch	View your posts	Forced login
324	search	?search_id=newposts	View posts since last visit	Forced login
391	viewtopic	?t=3 , start=0 , postdays=0 , postorder=asc , highlight=		Full Access
391	viewtopic	?t=3 , watch=topic , start=0 , sid=fd3c..	Watch this topic for replies	Full Access
794	adminindex	?sid=fd3c..		Forced login
126	login	?redirect=admin/index.php , sid=67f7242f6		Redirection
126	login	?redirect=posting.php , mode=newtopic , f=2		Redirection
126	login	?redirect=posting.php , mode=quote , p=5		Redirection
126	login	?redirect=posting.php , mode=reply , t=3		Redirection
126	login	?redirect=privmsg.php , folder=inbox , mode=post		Redirection
126	login	?redirect=privmsg.php , folder=inbox , mode=read , p=1		Redirection
126	login	?redirect=search.php , search_id=egosearch		Redirection
126	login	?redirect=search.php , search_id=newposts		Redirection

Table I
UNAUTHORIZED PAGE ACCESS WITH PARAMETERS FOR A GUEST USER ATTEMPTING TO ACCESS ADMININSTRATOR LINKS

PID	P_Name	Var ID	Http Var_Name	Var_Assign	Http Var Value	Http Var Type
18	faq	19	mode		bbcode	GETDT
109	index	119	c	$viewcat	1	GETDT
109	index	120	mark	$mark_read	forums	GETDT
145	modcp	191	f	$forum_id	2	GETDT
145	modcp	193	p	$post_id	5	GETDT
145	modcp	200	t	$topic_id	3	GETDT
145	modcp	209	confirm	$confirm		POSTDT
145	modcp	210	start	$start	0	GETDT
145	modcp	211	mode	$mode	ip	GETDT
145	modcp	215	sid	$sid	0f206abbb	GETDT
225	posting	235	mode	$ $var	delete	GETDT
225	posting	238	p	$ $var	5	GETDT
257	privmsg	306	p	$privmsg_id	1	GETDT
257	privmsg	308	p	$privmsgs_id	1	GETDT
319	profile	320	sid	$sid	fd3c29892	GETDT
324	search	348	search_author	$search_author	alalfi	GETDT
324	search	350	search_id	$search_id	egosearch	GETDT
365	viewforum	380	mark	$mark_read	topics	GETDT
391	viewtopic	413	postorder	$post_order	asc	GETDT

Table II
UNAUTHORIZED SERVER ENVIRONMENT VARIABLE ACCESS FOR A GUEST ATTEMPTING TO ACCESS ADMININSTRATOR LINKS

PID	P_Name	Action ID	Action return-value	Action_constraint
145	modcp	152	f.forum_id, f.forum_name, f.forum_topics	t . topic_id = $topic_id and f . forum_id = t . forum_id
145	modcp	157	forum_name, forum_topics	forum_id = $forum_id
225	posting	226	*	forum_id = $forum_id
225	posting	227	f.*, t.topic_status, t.topic_title, t.topic_type	t . topic_id = $topic_id and f . forum_id = t . forum_id
225	posting	228	f.*, t.topic_id, t.topic_status, t.topic_type, t.topic_first_post_id, ... p.post_id, p.poster_id, p.enable_html,.. p.enable_smilies, ... pt.post_subject, pt.post_text,... u.username, u.user_id, u.user_sig,...	p . post_id = $post_id and t . topic_id = p . topic_id and f . forum_id = p . forum_id and pt . post_id = p . post_id and u . user_id = p . poster_id
225	posting	868	emoticon, code, smile_url	
324	search	329	post_id	poster_id IN ($matching_userids)
324	search	336	topic_id	topic_replies = 0 and topic_moved_id = 0
324	search	341	pt.post_text, pt.bbcode_uid, pt.post_subject, p.*, f.forum_id, f.forum_name, t.*, u.username, u.user_id, u.user_sig, u.user_sig_bbcode_uid	p . post_id IN ($search_results) and pt . post_id = p . post_id and f . forum_id = p . forum_id and p . topic_id = t . topic_id and p . poster_id = u . user_id
388	viewonline	389	forum_name, forum_id	
388	viewonline	390	u.user_id, u.username, u.user_allow_viewonline,u.user_level, s.session_logged_in, s.session_time, s.session_page, s.session_ip	u . user_id = s . session_user_id and s . session_time >= (time () - 300)
391	viewtopic	393	t.topic_id	t2 . topic_id = $topic_id and t . forum_id = t2 . forum_id and t . topic_moved_id = 0 and t . topic_last_post_id > t2 . topic_last_post_id

Table III
UNAUTHORIZED SQL STATEMENT ACCESS FOR A GUEST USER ATTEMPTING TO ACCESS AN ADMINISTRATOR'S LINKS

In Table III we have uncovered examples of such a possibility in PhpBB 2.0, with potential anonymous user access to administrator SQL statements in the actions with ActionIDs 152, 157, 226, 227, 228, 329, 341 and 393.

B. Web application maintenance

Our framework can also be used for maintenance of role-based web application security. For example, this can be useful when the testing engineer has identified an access control security feature that is presently permitted to a specific role, and would like to disable this feature. Our framework can help by locating all the pages and database statements that allow this feature, as a step towards fixing the code to prevent this access in future.

As an example, in PhpBB 2.0 we have noticed that an anonymous user is allowed to see registered users' profile information, a feature that may lead to a privacy violation for forum members. We executed a Prolog query that searches for profile information in all accesses represented in the recovered model of a registered user. The query returns information on all pages in the registered role that permit such access, and identifies SQL statements that retrieve this information. Identifying the pages and SQL statements that allow this feature will help the software engineer to update the code to restrict the guest access. Figure 7 shows one of the Prolog queries used to implement this goal, and Table IV shows the result of these queries, identifying all anonymous user actions that can gain access to registered user email addresses and what pages and links allow that access.

```
% search for pages & actions that allow a user
%  to access other user emails
anon_email_retrieved(PageID, PageName,
                          ActionID, AcTS, RoleName) :-
    appRole(RolNum, RoleName),
    return(RolNum, SelList, ActionID, PageID),
    split_string(SelList, ', ', SelAtoms),
    member(user_viewemail,SelAtoms),
    pageNameID(RolNum, PageName, PageID),
    permActionTimeS(RolNum, _, ActionID, AcTS).
```

Figure 7. Prolog rule to check for access to user email addresses

PID	*Page_Name*	*Action ID*	*Action return-value*	*Action_constraint*
109	index	140	username, user_id, user_viewemail, user_posts, user_regdate,user_from, user_website, user_email, user_icq, user_aim, user_yim, user_msnm, user_avatar,.... user_avatar_type,	user_id <> - 1 ORDER BY $order_by
139	memberlist			
257	privmsg			
319	profile			
324	search	400	u.username, u.user_id, u.user_posts, u.user_from, u.user_website, u.user_email, u.user_icq, u.user_aim, u.user_yim, u.user_regdate, u.user_msnm, u.user_viewemail,....	p . topic_id = $topic_id $limit_posts_time and pt . post_id = p . post_id and u . user_id = p . poster_id
126	login			
145	modcp			
391	viewtopic			
225	posting	863	u.user_id, u.user_email, u.user_lang	tw . topic_id = $topic_id and tw . user_id NOT IN ($userdata [user_id], - 1, $row [ban_userid]) and tw . notify_status = 0 and u . user_id = tw . user_id

Table IV
LIST OF ALL PAGES AND ACTIONS THAT PERMIT ACCESS TO A USER'S EMAIL ADDRESS

C. Web application Reengineering

In PhPBB 2.0 administrator management tasks are protected by providing a valid administrator username and password on the login page. Users who provide such information can then access all the forum management tasks. This is implemented by performing role validation at the beginning of each restricted page using a call to the *pagestart()* PHP function. The function implements the validation using session information, deciding on the allowed level of access. Figure 6 shows the source code of the *PageStart.php* file and highlights the part of it that is responsible for controlling access.

To address this issue along with the fact that many web applications' access control is implemented using obscurity, such applications need to be reengineered to employ a strict security model not only at the level of page access but also at the level of server environment variable access and access to application entities and attributes. Using our Prolog model, by commenting out the code in *pagestart()*, an anonymous user was able to access all the administrator management pages, given the URL for them. With this modification, the results of the query of Figure 6 yield an exhaustive list of all of the ways that the anonymous user could gain such

```
<?php
   if (! defined ('IN_PHPBB'))
   {
       die ("Hacking attempt");
   }
   define ('IN_ADMIN', true);
   include ($phpbb_root_path.'common.'.$phpEx);
   $userdata = session_pagestart ($user_ip, PAGE_INDEX);
   init_userprefs ($userdata);
   if (! $userdata ['session_logged_in'])
   {
     redirect (append_sid ("login.$phpEx
               ?redirect=admin/index.$phpEx", true));
   }
   else
   if ($userdata ['user_level'] != ADMIN)
   {
     message_die (GENERAL_MESSAGE, $lang ['Not_admin']);
   }
   if ($HTTP_GET_VARS ['sid'] != $userdata ['session_id'])
   {
     redirect ("index.$phpEx?sid=".$userdata ['session_id']);
   }
   if (! $userdata ['session_admin'])
   {
     redirect (append_sid ("login.$phpEx
               ?redirect=admin/index.$phpEx&admin=1", true));
   }
   if (empty ($no_page_header))
   {
       include ('./page_header_admin.'.$phpEx);
   }
   ob_flush ();
?>
```

Figure 8. The *PageStart.php* file in PhpBB 2.0. Code that controls access to administration management pages is highlighted.

access, by page, resource, and action, providing the security engineer with all of the changes that would need to be made to localize security checking in the application.

In general, the security models generated by our framework can also be reviewed by a security engineer either by using any modeling tool that supports UML2.0 (such as RSA [18]), or by exploring the generated Prolog representation of the security model using queries. Either way the engineer can easily explore scenarios to check for the absence of legitimate access or the existence of unauthorized access to the web application as a result of changes or planned changes to the code. Since the recovered model provides fined-grained access information down to the level of the application's entities' attributes, the software engineer can also explore the potential effect of a proposed update to the access model, either by updating the security model or by updating the Prolog representation.

Once the security model is revised to reflect the new access control requirements, it can be used to restructure the application's database schema such that the security check is employed on every access to any of the application's entities' attributes. Database access in the web application can be updated accordingly based on the new database structure. This is made possible by the unique numbering generated by our framework and associated with each SQL statement and server environment variable reference in any of the application's server pages.

VII. Related Work

Several methods in the literature propose tools for the translation of UML diagrams to formal models that can be checked using formal verification tools. In the context of web application verification, Castelluccia et al. [11] and Sciascio et al. [16] use the Canollen model to build a diagram for the web application to verify its design. The authors implement a component, XMI2SMV, that converts UML diagrams in XMI format into a Web Application Graph (WAG). The WAG can be translated into an SMV model, which is then given to the NuSMV model checker [13].

Bordbar and Anastasakis [10] use the Alloy [15] constraint solver to find bugs in interaction between the user and the browser in web applications. They designed a model translation tool, UML2Alloy, which maps from their proposed UML diagram, Abstract Description of Interaction, to an Alloy model, which can then be verified using the Alloy analyzer. Other model translation tools are discussed in [17].

In access control analysis research, Letarte and Merlo [20] use static analysis to extract a binary role model from PHP code, or more specifically from PHP database statements. The approach then checks if the recovered model is restrictive or permissive. Change of authorization level in the code is modeled using an inter-procedural control flow graph. The approach accounts for only two roles, admin and user, for which access to database statements may or may not be granted. It uses an application-dependent authorization pattern, and lacks for correspondence to source code.

Koved et al. [19] propose an approach to automatically compute access rights requirements in Java 2.0 applications and specifically for mobile code such as applets and servlets. The authors use context sensitive data and control flow analysis to construct an Access Right Invocation graph representing the authorization model of the code. This enables identification of classes in each path that contain a call to the Java 2.0 security authorization subsystems.

Pistoia et al. [28] propose an approach to detect inconsistencies in an application's RBAC policy. They statically construct a call graph to represent the flow of authorization by over-approximating method calls in the application and identifying access-restricted methods. The graph forms the basis of several security analyses, including whether the application's RBAC security policy is restrictive or permissive.

Mendling et al. [22] propose a meta-model based integration approach to enhance the security features of Business Process Management Systems that operate via Web Services (BPEL). The authors use an XSTL transformation script to extract roles and permissions from a BPEL process definition based on a proposed mapping. The extracted information is stored in XML, which can be imported into the tool xoRBAC, which enables the definition and enforcement of RBAC policies and constraints for web services.

In the area of web applications, many model-based testing techniques have been proposed. However, the main focus has been on testing structural, dynamic, or interaction aspects of web applications rather than on security testing or access control properties. A state of the art discussion of these techniques can be found on our recent survey [5].

VIII. Conclusions and Future Work

In this paper we have presented an approach and a tool to analyze RBAC security models automatically recovered from existing dynamic web applications. We used source transformation technology in SecureUML2Prolog to transform the recovered SecureUML models into Prolog. The resulting formal models can be used to check RBAC security properties in the application under test. We demonstrated our method on a production web application, PhpBB 2.0, and illustrated its use in security analysis, testing, maintenance and reengineering. In future work we are planning to conduct a larger scale evaluation to better test the effectiveness of our method, and to extend and adapt our framework to address other security analysis tasks.

Acknowledgments

This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the IBM Canada Centre for Advanced Studies.

References

[1] Gail-Joon Ahn and Ravi S. Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, 2000.

[2] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. SQL2XMI: Reverse Engineering of UML-ER Diagrams from Relational Database Schemas. In *WCRE*, pages 187–191, 2008.

[3] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. A Verification Framework for Access Control in Dynamic Web Applications. In *C3S2E*, pages 109–113, 2009.

[4] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications. In *ICSTW*, pages 295–302, 2009.

[5] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. Modeling methods for web application verification and testing: State of the art. *Software Testing, Verification and Reliability*, 19(4):265–296, 2009.

[6] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. WAFA: Fine-grained Dynamic Analysis of Web Applications. In *WSE*, pages 41–50, 2009.

[7] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. Automating Coverage Metrics for Dynamic Web Applications. In *CSMR*, pages 51–60, 2010.

[8] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. Recovering Role-based Access Control Security Models from Dynamic Web Applications. In *ICWE*, pages 121–136, 2012.

[9] David A. Basin, Manuel Clavel, and Marina Egea. A decade of model-driven security. In *SACMAT*, pages 1–10, 2011.

[10] Behzad Bordbar and Kyriakos Anastasakis. MDA and Analysis of Web Applications. In *TEAA*, volume 3888 of *LNCS*, pages 44–55. Springer, 2005.

[11] Daniela Castelluccia, Marina Mongiello, Michele Ruta, and Rodolfo Totaro. WAVer: A Model Checking-based Tool to Verify Web Application Design. *Electr. Notes Theor. Comput. Sci.*, 157(1):61–76, 2006.

[12] Joanna Chimiak_Opoka, Michael Felderer, Chris Lenz, and Christian Lange. Querying UML Models using OCL and Prolog: A Performance Study. *ICSTW*, pages 81–88, 2008.

[13] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A New Symbolic Model Checker. *Int. Journal on Soft. Tools for Tech. Transfer STTT*, 2(4):410–425, 2000.

[14] James R. Cordy. The TXL source transformation language. *Science of Computer Programming*, 61(3):190–210, 2006.

[15] Daniel Jackson. Alloy: A New Technology for Software Modelling. In *TACAS*, volume 2280 of *LNCS*, page 20. Springer, 2002.

[16] Eugenio Di Sciascio, Francesco M. Donini, Marina Mongiello, Rodolfo Totaro, and Daniela Castelluccia. Design Verification of Web Applications Using Symbolic Model Checking. In *ICWE*, volume 3579 of *LNCS*, pages 69–74. Springer, 2005.

[17] Maria Encarnación Beato Gutiérrez, Manuel Barrio-Solórzano, Carlos Enrique Cuesta Quintero, and Pablo de la Fuente. UML automatic verification tool with formal methods. *Electr. Notes Theor. Comput. Sci*, 127(4):3–16, 2005.

[18] IBM Corp. Rational Software Architect 7.0, last access October 2011.

[19] Larry Koved, Marco Pistoia, and Aaron Kershenbaum. Access rights analysis for Java. In *OOPSLA*, pages 359–372, 2002.

[20] Dominic Letarte and Ettore Merlo. Extraction of Interprocedural Simple Role Privilege Models from PHP Code. In *WCRE*, pages 187–191, 2009.

[21] Hongzhi Liang and Jürgen Dingel. A Practical Evaluation of Using TXL for Model Transformation. In *SLE*, pages 245–264, 2008.

[22] Jan Mendling, Mark Strembeck, Gerald Stermsek, and Gustaf Neumann. An Approach to Extract RBAC Models from BPEL4WS Processes. In *WETICE*, pages 81–86, 2004.

[23] MySQL. MySQL Market Share, *http://www.mysql.com/why-mysql/marketshare/*, last access Nov 26, 2011.

[24] Netcraft Ltd. November 2011 web server survey, *http://news.netcraft.com/archives/category/webserver-survey/*, last access Nov 26, 2011.

[25] Richard Paige and Alek Radjenovic. Towards Model Transformation with TXL. In *First Intl. Workshop on Metamodeling for MDA*, pages 163–177, 2003.

[26] PHP Group. PHP usage Stats for April 2007, *http://www.php.net/usage.php*, last access Nov 26, 2011.

[27] Marco Pistoia, Satish Chandra, Stephen J. Fink, and Eran Yahav. A survey of static analysis methods for identifying security vulnerabilities in software systems. *IBM Systems Journal*, 46(2):265–288, 2007.

[28] Marco Pistoia, Robert J. Flynn, Larry Koved, and Vugranam C. Sreedhar. Interprocedural analysis for privileged code placement and tainted variable detection. In *ECOOP*, pages 362–386, 2005.

[29] Harald Störrle. A PROLOG-based Approach to Representing and Querying Software Engineering Models. In *VLL*, pages 71–83, 2007.

[30] WatirCraft. Watir, *http://wtr.rubyforge.org*, last access March 2009.