A VERIFICATION FRAMEWORK FOR ACCESS CONTROL IN DYNAMIC WEB APPLICATIONS

by

MANAR H. ALALFI

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen’s University
Kingston, Ontario, Canada
April 2010

Copyright © Manar H. Alalfi, 2010
Abstract

Current technologies such as anti-virus software programs and network firewalls provide reasonably secure protection at the host and network levels, but not at the application level. When network and host-level entry points are comparatively secure, public interfaces of web applications become the focus of malicious software attacks. In this thesis, we focus on one of most serious web application vulnerabilities, broken access control. Attackers often try to access unauthorized objects and resources other than URL pages in an indirect way; for instance, using indirect access to back-end resources such as databases. The consequences of these attacks can be very destructive, especially when the web application allows administrators to remotely manage users and contents over the web. In such cases, the attackers are not only able to view unauthorized content, but also to take over site administration. To protect against these types of attacks, we have designed and implemented a security analysis framework for dynamic web applications. A reverse engineering process is performed on an existing dynamic web application to extract a role-based access-control security model. A formal analysis is applied on the recovered model to check access-control security properties. This framework can be used to verify that a dynamic web application conforms to access control policies specified by a security engineer. Our framework provides a set of novel techniques for the analysis and modeling of web applications for the purpose of security verification and validation. It is largely language independent, and based on adaptable model recovery which can support a wide range of security analysis tasks.
Co-Authorship

All papers resulting from this thesis were co-authored with my supervisors Dr. James R. Cordy and Dr. Thomas R. Dean. In all cases I am the primary author.

Part of Chapter 2 was published at the International Conference on Web Engineering (ICWE 2007) [9], and in the Journal of Software Testing, Verification, and Reliability (STVR) [13]. Part of Chapter 3 was published at the Canadian Conference on Computer Science and Software Engineering (C3S2E 2009) [11]. Part of Chapter 4 was published at the Working Conference on Reverse Engineering (WCRE 2008) [8]. Chapter 5 was published at the international workshop on Web Testing (WebTest 2009) [?]. Chapter 6 was published at the international symposium of Web Systems Evolution (WSE 2009) [14]. Chapter 7 will appear in the proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2010). Part of Chapter 8 is submitted to the International Symposium on Software Testing and Analysis (ISSTA 2010).
Dedication

In loving memory of my whole-hearted father, Hassan Alalfi, the candle that burned to shed the light for me to go. To you I dedicated the fruit of my efforts, it was your dream and I thank Allah for helping me to make it come true. May Allah bless you.
Acknowledgments

In the name of Allah, the Most Gracious, the Most Merciful. First and foremost praises and thanks to Allah, the Lord, the Almighty, the All-Knowing and the Glorious who bestowed upon us all the blessings and the faculties of thinking, searching, and learning.

Praise be to Allah, who says in his glorious book "If you give thanks, I will give you more of my blessings" so I praise Allah for his favour to me in completing this dissertation and for surrounding me with people who care and love.

This dissertation would not have been a real fulfillment without the backing and cooperation from my supervisors Professors James Cordy and Thomas Dean. Their collaborative approach in supervision was instrumental in planting the seed for a successful scholar. They always were respectful of my views and ideas. I’m really fortunate to have the opportunity of working with them. They both are great role models of open minded and brilliant researchers who one should strive to become.

I would also like to thank the members of my oral defense committee, Ettore Merlo, T.C.N. Graham, T.P. Martin, and K. Rudie for their time and insightful questions. My appreciation and gratefulness to all my colleagues and to the friendly members of the School of Computing who have helped in one way or another along the way. In particular, I would like to thank Aseel Almonaies for her sincere friendship.

Most importantly, none of this would have been possible without the love and patience of my family. Mom, thank you so much for your love and payers, you have been a constant
source of love, concern, support and strength all these years. It was under my father and your watchful eyes that I gained so much drive and an ability to tackle challenges head on. I would like to express my heart-felt gratitude, to all my brothers and sisters, especially my brother Salah Alalfi, for their faith in me and allowing me to be as ambitious as I wanted. Also I warmly appreciate my mother-in-law’s love and prayers.

I owe my deepest gratitude to my husband, Shaker Jarrar, who has been proud and supportive of my work and who has shared the many uncertainties, challenges and sacrifices for completing this dissertation. His support, encouragement, quiet patience and unwavering love were undeniably the bedrock upon which the past five years of my life have been built.

My son, Muhammad Jarrar, who has grown into a wonderful 4 years old in spite of his mother spending so much time away from him working on this dissertation. Thanks my little one for your patience and for filling my moments with happiness and joy.
Statement of Originality

I, Manar H. Alalfi, certify that the research work presented in this thesis is my own and was conducted under the supervision of Dr. James R. Cordy and Dr. Thomas R. Dean. All references to the work of other people are properly cited.
# Table of Contents

Abstract ii

Co-Authorship iii

Dedication iv

Acknowledgments v

Statement of Originality vi

Table of Contents vii

List of Tables x

List of Figures xii

1 Introduction 1

1.1 Motivation 1

1.2 Research Statement 2

1.3 Contributions 3

1.4 Structure of the dissertation 5

Chapter 2: Related Work - Analysis Models and Methods for Web Application Verification and Testing 7

2.1 Motivation 8

2.2 Web Application Modeling 10

2.3 Comparison and Categorization Criteria 17

2.4 Comparative Analysis 20

2.5 Proposed Methods That Do Not Fit Our Comparison Criteria 44

2.6 Models Proposed But Not Yet Used for Verification and Testing 46

2.7 Conclusions and Open Problems 47
Chapter 3:
A Verification Framework for Access Control in Dynamic Web Applications ........................................... 51
3.1 Motivation .................................................................................................................. 52
3.2 Research Approach .................................................................................................. 53
3.3 Evaluation and Preliminary Results ......................................................................... 57
3.4 Conclusion ................................................................................................................ 58

Chapter 4:
Lightweight Transformation of Data Models from SQL Schemas to UML-ER ................................. 59
4.1 Motivation .................................................................................................................. 60
4.2 Previous Work ......................................................................................................... 62
4.3 SQL2XMI .................................................................................................................. 64
4.4 An Example: PhpBB ............................................................................................... 74
4.5 Conclusions and Future Work ................................................................................. 78

Chapter 5:
Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications .......... 79
5.1 Motivation .................................................................................................................. 80
5.2 PHP2XMI .................................................................................................................. 82
5.3 An Example Application ......................................................................................... 90
5.4 Related Work .......................................................................................................... 93
5.5 Conclusion and Future Work ................................................................................. 96

Chapter 6:
WAFA: Fine-grained Dynamic Analysis of Web Applications .......................................................... 97
6.1 Motivation .................................................................................................................. 98
6.2 Approach .................................................................................................................. 100
6.3 Instrumentation Methodology .................................................................................. 101
6.4 Evaluation ............................................................................................................... 112
6.5 Related Work .......................................................................................................... 114
6.6 Future Work and Conclusions ................................................................................. 116

Chapter 7:
DWASTIC: Automating Coverage Metrics For Dynamic Web Applications ................................... 117
7.1 Motivation .................................................................................................................. 118
7.2 Web Application Coverage Metrics .......................................................................... 120
7.3 Constructing the Coverage Database ........................................................................ 122
7.4 Evaluation ............................................................................................................... 133
7.5 Related Work .......................................................................................................... 135
7.6 Conclusion ............................................................................................................... 138
List of Tables

2.1 Desirable Properties for Website modeling ........................................... 14
2.2 Desirable Properties for Website modeling (Cont.) ................................. 15
2.3 Reference linking summary tables with text .......................................... 21
2.4 Summary of Methods Categorized by Modeling Level .............................. 22
2.5 Detailed Comparison of Methods by Properties Covered ........................... 23

4.1 Mappings between MySQL schema elements, ERD elements, and XMI 2.1  elements ........................................................................................................... 68

6.1 Sample trace elements for the Server Pages database table ...................... 111
6.2 Sample trace elements for the Http Variables database table .................... 111
6.3 Sample trace elements for the Database Interactions database table .......... 112
6.4 Trace statistics for anonymous user interactions with a PhpBB 2.0 forum .... 113
6.5 Related work comparisons ....................................................................... 114

7.1 Example AllPhpPages coverage database table, tracking page coverage ...... 124
7.2 Example AllHttpVars coverage database table, tracking server environment variables coverage .............................................................. 125
7.3 Example AllSqlsources coverage database table, tracking SQL statement coverage .......................................................................................... 125
7.4 Coverage metrics results for pages, server environment variables and SQL statements at the application level for a sample test case ....................... 135
7.5 Coverage metrics results for server environment variables and SQL statements at the page level for a sample test case ................................... 135
7.6 Coverage metrics results for pages and server environment variables at the SQL statement level for a sample test case ................................ 136
7.7 Performance penalty of DWASTIC instrumentation on dynamic pages of PhpBB 2.0 ................................................................. 136

9.1 Characteristics of the PhPBB2.0 application ........................................... 163
9.2 Experiment statistics for the number of navigated and filtered pages ........ 164
9.3 Experiment coverage information for SQL statements, pages and server environment variables ................................................................. 164
9.4 More detailed coverage information for SQL statements, and server environment variables ................................................................. 165
9.5 Unauthorized pages access for a guest user attempting to access administrator’s links

9.6 Unauthorized pages access with parameters for a guest user attempting to access administrator’s links. \textit{PhpBB react code: access redirected(1), error message(2), access allowed(3)}

9.7 Unauthorized server’s environment variables access for a guest user attempting to access administrator’s links

9.8 Unauthorized SQL statement access for a guest user attempting to access an administrator’s links

9.9 List of pages and actions that permits a user to access other user’s Email address
List of Figures

2.1 Web Application Components (from [151]) ................................. 11
2.2 Interaction Behavior Modeling Methods: the Di Lucca and Di Penta (Luc-

caP03) [65], Graunke et al. (GFKF03) [88] and Licata and Krishnamurthi
(LK04) [120] models .......................................................... 26
2.3 Interaction Behavior Modeling Methods (cont’d): the Bordbar and Anas-
tasakis (BA05) [33] and Chen and Zhao (CZ04) [45] models ............ 28
2.4 UML-Based Models: the Tonella and Ricca (TR02) [163] and Conallen (Con99)
[54] models, and the Knapp and Zhang (KZ06) [111] UWE basis model .... 32
2.5 Graph-Based Models: the MCWEB [60], Ricca and Tonella (RT00) [150] and
Di Sciascio et al. (SMP02, SMP03) [72, 71] models ....................... 34
2.6 Graph-Based Models (cont’d): the Di Sciascio et al. (SDM+05) [73] and
Castelluccia et al. (CMRT06) [40] methods ............................... 37
2.7 Modeling Web Applications Using a Single Model: the VeriWeb (BFG02)
[32], Haydar et al. (HPS04) [100], and FSMWeb (AOA05) [24] methods .. 41

3.1 The Proposed Framework ................................................ 54

4.1 The TXL Transformation Technique as Applied in our Tool ............ 66
4.2 Grammatical specification and main transformation rule (function) ..... 66
4.3 The GenerateERDElements function, which transforms each SQL table to
its representation in XMI .................................................. 67
4.4 The createEntityAttribute function, which translates SQL table columns to
attributes in XMI ......................................................... 70
4.5 The IsPKAttribute function, which identifies and stereotypes the XMI rep-
resentation of columns which are primary keys ........................... 71
4.6 A part of the PhpBB 2.0 MySQL schema ................................ 73
4.7 Sample of the generated XMI 2.1 file .................................. 75
4.8 RSA visualization for a part of the generated diagram ................... 76

5.1 PHP2XMI tool Architecture ............................................ 83
5.2 The main TXL transformation rule .................................... 85
5.3 The instrumentHTTPVar transform. rule ............................ 86
5.4 The Conv_func_GET TXL transform. rule ............................ 87
5.5 Result of instrumenting the main index.php server page of PhpBB 2.0 .. 88
5.6 UML sequence diagram meta-model elements ........................ 90
5.7 Sample of a database view of generated execution traces .......................... 90
5.8 Fine grained information collected by PHP2XMI ................................. 91
5.9 An example of a generated sequence diagram ................................. 92

6.1 WAFA Architecture ................................................................. 100
6.2 The WAFA Dynamic Analysis database model ............................................... 101
6.3 Results of instrumenting server environment variables in a snippet of code in PhbBB 2.0 application .......................................................... 104
6.4 Result of instrumenting cookie management functions in a snippet of code in the PhbBB 2.0 application .......................................................... 104
6.5 PHP grammar extension to recognize guest patterns (SQL statements) ... 105
6.6 Instrumented snippet of code for PhpBB2.0 application - 1 ..................... 107
6.7 Instrumented snippet of code for PhpBB2.0 application - 2 ..................... 108
6.8 The `instrumentQueriesSource` transformation rule ......................... 110
6.9 The `instrumentSQL` transformation rule ............................................. 110

7.1 DWASTIC Tool Architecture ............................................................... 122
7.2 Dynamic Analysis database model ............................................................ 123
7.3 The TXL `instrumentPage` rule adds page coverage instrumentation to the top of each processed page ......................................................... 127
7.4 Coverage instrumentation added by DWASTIC to the `search.php` dynamic page of the PhpBB 2.0 application .......................................................... 129
7.5 DWASTIC instrumentation for the database interaction points of the `mysql4.php` function of PhpBB 2.0 ............................................................ 130
7.6 TXL program to identify and extract the coverage aspect of an instrumented PHP program ............................................................. 131
7.7 Part of the extracted instrumentation slice for PhpBB 2.0 augmented with database insertion code ............................................................. 133

8.1 Tool Architecture .............................................................. 142
8.2 Mappings between Database trace model and UML SD meta-model .......... 143
8.3 Interaction scenarios and messages encoding ........................................... 144
8.4 A sample UML2.0 Entity-level Sequence Diagram (SD) as generated by WAFA and PHP2XMI ............................................................. 145
8.5 UML2.0 Structural and behavioural meta-models and their mappings to SecureUML meta-model ............................................................. 149
8.6 TXL main rule for PHP2SecureUML .................................................. 150
8.7 RSA project explorer shows the SecurUML elements as recovered by our tool ... 153
8.8 A sample of a generated SecureUML model instance ............................ 154
8.9 TXL main rule for SecureUML2Prolog .................................................. 155
8.10 A sample TXL rule for generating permission facts from SecureUML meta-model elements ............................................................. 156
8.11 Prolog Facts generated for SecureUML model analysis ......................... 157
9.1 The **CollectFormInputs** algorithm for collecting the application’s forms input elements ......................................................... 163
9.2 Prolog rules to check for unauthorized access on the application’ entities . . 165
9.3 Prolog rules to check for Guest access on other registered users’ profiles . . 170
9.4 *PageStart.php* file in PhpBB 2.0, code that control the access on administra-
tion management pages is highlighted ........................................... 172
Chapter 1

Introduction

1.1 Motivation

Current technologies such as anti-virus software programs and network firewalls provide reasonably secure protection at the host and network levels, but not at the application level. When network and host-level entry points are comparatively secure, public interfaces of web applications become the focus of attacks [170].

This thesis focuses on one of the most serious web application vulnerabilities, broken access control. Access control, sometimes called authorization, governs how web applications grant access to functions and content to some users and not to others [146]. Depending on the access control model, sets of users can be grouped into roles, where privileges are assigned to roles rather than users. This kind of access control model facilitates the administration of user management and is called a Role-Based Access Control model (RBAC) [158].

Broken access control in web applications is considered one of the top ten web application security vulnerabilities [146]. Most web applications try to implement access control policies using obscurity, where links to pages are not presented to unauthorized users. This method of protection is not sufficient because attackers can attempt to access hidden URLs, knowing that sensitive information and functions lie behind these URLs. Attackers also try to access
unauthorized objects and resources other than URL pages in an indirect way, for instance, indirect access to back-end resources such as databases.

The consequences of allowing unprotected flows to crafted requests could be very destructive, especially when the web application allows administrators to remotely manage users and contents over the web. In such cases the attackers are not only able to view unauthorized content, but also to take over site administration.

Broken access control is usually caused by an unreliable implementation of access control techniques. In many current web applications, access control polices are spread over the code, which makes the process of understanding and maintaining such rules a difficult if not impossible task [146].

To protect against this attack, access control polices should be based on a strong model that is implemented at all levels of the web application, including both the presentation level and the business level as well. Checking for authorization should be done on every attempt to access secure information, and access control mechanisms should be extensively tested to ensure that there is no way to bypass them [146].

Most of the previous efforts in web application testing and verification are forward engineering approaches [107, 16, 58, 6, 7], while the real need is for a reverse engineering approach that is not only able to model access control polices, but also able to check them in real applications. There is a critical need for an approach that is able to test or model check web applications to ensure that they are protected from broken access control attacks, and this is the goal of our work.

1.2 Research Statement

This thesis presents a security analysis framework for dynamic web applications. The framework is aimed at testing the conformance of dynamic web applications with role-based access control security policies. A reverse engineering process is performed over a
dynamic web application to extract a role based access control security model. A formal analysis is applied on the recovered model to check access control security properties. This framework can be used to verify that a dynamic web application conforms to access control polices specified by a security engineer, either with a correctness check, or with a counter example if any access control violation is encountered in the code.

1.3 Contributions

- The proposed approach is a novel one in web application security verification. Besides being the first approach to tackle the issue of access control verification, the proposed framework is flexible enough to allow for different server side technologies and databases in plug and play fashion.

- Our approach also yields the potential for application in systems other than web applications. The static and dynamic reverse-engineering front-end of the framework can be reused for other kinds of analysis, and the framework could be used to discover other kinds of security attacks, such as cross-site scripting and SQL injection.

- In the first part of this thesis, we conducted a comprehensive survey on different modeling methods used in web site verification and testing. Based on a short catalogue of desirable properties of web applications that require analysis, two different views of the methods are presented: a general categorization by modeling level, and a detailed comparison based on property coverage.

- In the second part of this thesis we designed and implemented an automated transformation from an SQL (DDL) schema to an open XMI 2.1 UML-adapted class model. The adapted model is a tailored UML class model to represent the basic ER diagram components, including entities, attributes, relations, and primary keys. Our transformation technique with its tool, SQL2XMI, is a novel one in that it is open, non-vendor
CHAPTER 1. INTRODUCTION

specific, and targeted at the standard UML 2.1 exchange format, XMI 2.1. Although comparable commercial transformations exist, they are closed technologies targeted at formats tightly coupled to the vendor’s tools, hindering portability and preventing users from choosing their preferred tools in the development process.

• In the third part we designed and implemented an approach to automatically instrument dynamic web applications using source transformation technology, and to recover a sequence diagram from execution traces generated by the resulting instrumentation. Using an SQL database to store generated execution traces, our approach automatically filters traces to reduce redundant information that may complicate program understanding.

• We supported the dynamic analysis phase by an automated instrumentation coverage approach to decrease the percentage of false positives. A set of new coverage metrics, specialized for dynamic web applications, is also proposed and implemented. In addition, we performed a great deal of analysis on the embedded database interaction in the host application. This includes automated distilling of the SQL embedded system, analyzing it, and modeling it as a part of the whole system. Also, three tools have been developed in support of this part, PHP2XMI, WAFA, and DWASTIC.

• In the fourth part, making use of Model Driven Engineering, we automatically constructed a Role-Based Access Control security model from the recovered structural and behavioral models. We used TXL to implement the automatic model to model transformation and composition. The generated model is also represented in the UML 2.1 exchange format, XMI 2.1.

• In the last part, we developed, based on model-to-model transformation approach, a novel tool to transform the semi-formal UML 2.1 security model into a formal model to ease the process of verifying the system against security properties. Two tools have
been developed in support of this phase, PHP2SecureUML, and SecureUML2Prolog.

- In our first experiment, the framework is being evaluated on one of the most popular PHP web applications, PhpBB, to check that the application is free from any remaining access control vulnerabilities.

1.4 Structure of the dissertation

In this chapter we have presented and motivated the primary research problem of RBAC security model recovery and analysis for dynamic web applications, the thesis statement, and the contributions of the thesis research. The remaining chapters of the thesis are organized as follows:

- **Chapter 2**: presents our proposed categorization criteria for web application verification and testing modeling methods, and provides a comprehensive state of the art study of methods in the field.

- **Chapter 3**: provides an outlines for our Role-Based Access Control security analysis framework and its components.

- **Chapter 4**: describes an approach for automated transformation from an SQL (DDL) schema to an open XMI 2.1 UML-adapted class model.

- **Chapter 5**: presents an approach and tool to automatically instrument dynamic web applications using source transformation technology and to recover a sequence diagram from execution traces generated by the resulting instrumentation.

- **Chapter 6**: presents an approach for fine-grained analysis of dynamic web applications.

- **Chapter 7**: presents an automated instrumentation coverage approach to support dynamic analysis and to decrease the percentage of false positives.
• Chapter 8: describes our approach for automatically constructing a Role-based Access Control security model from the recovered structural and behavioral models.

• Chapter 9: presents a set of experiments for our framework and analyses the results.

• Chapter 10: presents a summary of the thesis, a list of contributions, future work and a statement of conclusions.
Chapter 2

Related Work - Analysis Models and Methods for Web Application Verification and Testing

Models are considered an essential step in capturing different system behaviors and simplifying the analysis required to check or improve the quality of software. Verification and testing of web software requires effective modeling techniques that address the specific challenges of web applications. In this study we survey 24 different modeling methods used in website verification and testing. Based on a short catalogue of desirable properties of web applications that require analysis, two different views of the methods are presented: a general categorization by modeling level, and a detailed comparison based on property coverage.

The rest of this chapter is organized as follows. Section 2.1 motivates our work. Section 2.2 gives a brief introduction to web applications and web services and the challenges that affect the analysis and modeling of web applications. Section 2.3 describes the set of comparison and categorization criteria used in our study. In Section 2.4 we give a brief
CHAPTER 2. RELATED WORK

summary and a comparative analysis of the 24 modeling methods. Sections 2.5 and 2.6 add descriptions of some other related methods. Finally, we conclude and suggest some of the open problems in the area in Section 2.7.

2.1 Motivation

Like many software domains, web applications are becoming more complex. This complexity arises due to several factors, such as a larger number of hyperlinks, more complex interaction, and the increased use of distributed servers. Modeling can help to understand these complex systems, and several papers in the literature have studied the specific problem of modeling web applications. In some cases, new models have been proposed, while in other cases, existing modeling techniques have been adapted from other software domains. Modeling can help designers during the design phases by formally defining the requirements, providing multiple levels of detail, and providing support for testing prior to implementation. Support from modeling can also be used in later phases to support validation and verification.

Most of the early literature concentrates on the process of modeling the design of web applications. It proposes forward engineering-based methods designed to simplify the process of building highly interactive web applications [84, 152, 62, 43]. Other research uses reverse engineering methods to extract models from existing web applications in order to support their maintenance and evolution [99, 25, 66]. This chapter surveys a range of different analysis models that are currently applied in the field of verification and testing of web applications. Design modeling methodologies such as those reviewed in [56, 112, 76] are outside the scope of our study. Our survey focuses on the modeling methods used. Thus testing and verification methods as a whole, such as user session-data testing [75] and bypass testing [140] are also outside the scope of this survey.

While reviewing different analysis methods in our scope, we found that some of the
CHAPTER 2. RELATED WORK

literature focuses on modeling the navigational aspects of web applications [30, 150, 73, 96], while others concentrate on solving problems arising from user interaction with the browser in a way that affects the underlying business process [33, 120]. Still others are interested in validating the correctness and completeness of web page contents [18, 20, 17, 19]. The surveyed models are interested in modeling and verifying either the static or the dynamic behaviors and features of web applications [24, 100].

Previous surveys have compared methods proposed for web application development [112, 76]. Most concentrate on design methods, methods that focus on the preliminary phases of the software life cycle. Cuaresma et al. [56] on the other hand concentrate on comparing methods dedicated to requirements engineering. The work most similar to our study is the survey done by Di Lucca and Fasolino [69], but their focus is on the functional testing of web applications, while ours is on the analysis models underlying web application verification and testing. We are interested in those methods that propose models to capture different properties related to the structure (navigation), behavior, and content of web applications, and whether these properties are static, dynamic, or interactive.

To date, no one has analyzed modeling methods devoted to verification and testing of web applications taking into consideration the capabilities of those methods in capturing, verifying or testing the set of desired web application properties that we discuss in this chapter. This study is undertaken to investigate the current state of the art in the field of web application verification and testing, and to distill what is being done and what still needs to be done to help researchers interested in this field to fill the gaps. For web engineers interested in modeling, this study may provide an insight to the different models and notations used to capture the different features in web applications, and encourage them to propose new improved models. For those interested in web application verification and testing, the analysis provided here may suggest new directions that need to be investigated to improve the quality of web applications. In addition, it provides an overview of the range of analyses that are being used to support web application verification and testing which
may help in the integration of different methods to derive new, improved techniques.

2.2 Web Application Modeling

In this section we set our work in the context of the web environment, web applications and services, introduce the major challenges in the analysis and modeling of web applications for verification and testing, and outline the desirable properties of web application models that form the basis of our study.

2.2.1 Web Applications

For the purpose of this survey, a web application is a software application that is accessible via a thin client (i.e., web browser) over a network such as the Internet or an intranet. A web application is often structured as a three-tiered application. As shown in Figure 2.1, the web browser represents the first tier. The web server that implements CGI, PHP, Java Servlets or Active Server Pages (ASP), along with the application server that interacts with the database and other web objects is considered the middle tier. Finally, the database along with the DBMS server forms the third tier.

Web applications generate web pages, comprising different kinds of information such as text, images and forms. These web pages can be either static or dynamic. Static pages reside on a web server and contain only HTML and client-side executable code (e.g., JavaScript) and are served by the web server. Dynamic pages are generated as the result of the execution of various scripts and components on the server. These pages contain a mixture of HTML source and executable code, and are served by the application server.

In our study we treat the terms web application and website as synonymous. Some researchers consider a website to be simply a set of related web pages grouped together by some means on a server, or in a folder on a server. Such pages are static pages that don’t use dynamic features and thus need not be processed by the application servers.
We are interested in all methods that propose models to capture different properties related to the structure (navigation), behavior, and content of web applications; and whether these properties are static, dynamic, or interactive. Another level that could be considered is the presentation level, which describes how information is to be presented to users. Issues related to this level include attributes such as color and font as well as cascading style sheets. In this study we concentrate on the semantics of web applications rather than presentation issues.

### 2.2.2 Web Services

Web services are a standardized way of integrating web-based applications using separate service communication interfaces that can be used by other web applications. They are
primarily used as a means for businesses to communicate with clients and each other without exposing detailed knowledge of each other’s IT systems. Communication is usually in XML, and not tied to any particular operating system or programming language. Web services don’t require the use of browsers or HTML, and don’t provide the user with a GUI. Web services are outside the scope of the present study, but may be a future direction for our work.

2.2.3 Challenges in Analysis and Modeling of Websites

Web applications are evolving rapidly, as many new technologies, languages, and programming models are used to increase the interactivity and the usability of web applications. This inherent complexity brings challenges to modeling, analysis, testing, and verification of this kind of software. Some of these challenges are:

- The diversity and complexity of the web application environment increases the risk of non-interoperability and the complexity of integration. Web applications interact with many components that run on diverse hardware and software platforms. They are written in diverse languages and they are based on different programming approaches such as procedural, OO, interpreted, and hybrid languages such as Java Server Pages (JSPs). The client-side includes browsers, HTML, embedded scripting languages and applets. The server-side includes CGI, JSPs, Java Servlets, and .NET technologies. They all interact with diverse back-end engines and other components that are found on the web server or other servers. The integration of such components and the web system in general is extremely loose and dynamically coupled, which provides powerful abstraction capabilities to the developers, but makes analysis for testing and verification extremely difficult.

- Another major challenge comes from the dynamic behavior, including dynamically generated client components, dynamic interaction among clients and servers, and the
continual changes in the system context and web technologies.

- Web applications may have several entry points, and users can engage in complicated interactions that the web application cannot prevent. Web applications often contain database components and may provide the same data to different users. In these cases, applying access control mechanisms becomes an important requirement for safe and secure access to web application resources, and the process of implementing and applying such rules is considered a great challenge.

- Web applications have the property of low observability, due to the difficulty of tracking some outputs. Usually the output that is observed and analyzed consists of the HTML documents sent back to the user. But there are also other kinds of output, such as the changed state of the server or the database, messages sent to other web applications and services, and so on. It is considered a challenge to perform a precise analysis of web applications that takes into account all of this information.

### 2.2.4 Desirable Properties For Website Modeling

We can view web applications from three orthogonal perspectives (levels of modeling): web navigation, web content and web behavior. We first present an initial categorization of the desirable properties of web applications based on the level of modeling, and in Section 2.3 we further categorize the properties according to the static, dynamic, or interaction aspects as applicable:

1. Web Navigation

   - *Static navigation properties:* Most of the early literature on web analysis and modeling concentrates on dealing with static links, treating web applications as hypermedia applications. It addresses the checking of properties such as broken links, reachability (e.g., return to the home page), consistency of frame structure,
### Feature modeled or property checked

<table>
<thead>
<tr>
<th>Feature modeled or property checked</th>
<th>Example feature or property description</th>
</tr>
</thead>
</table>
| **Static Navigation Properties**    | Broken links Verify the absence of broken links in the web site. An example formula in computational tree logic (CTL), [SDMP02]:  
Φ₁ = AG(link → EX page). For each link in the web site, a page exists that is attached to it in the next state. |
| **Reachability**                    | Check if there is at least one navigation path from the start page (StartPage) to the target page (TargetPage). An example formula in computational tree logic (CTL), [HH06]:  
EF (StartPage = TargetPage). |
| **Dead End**                       | Check that it is not possible to reach Target Page along any path from the start point. (TargetPage is not reachable from any other page). An example formula in computational tree logic (CTL), [HH06]:  
Not(AG EF (StartPage = TargetPage)). |
| **Frames consistency**             | Example situations that lead to frames inconsistencies:  
- Duplicated frame names (a name l that occurs in more than one frame tag).  
- Frame trees deeper than a fixed threshold.  
- Non-existent link targets (anchors tag <a, l > such that l does not appear in any frame tag). [DA01]  
An example formula in linear temporal logic (LTL), [HP04]:  
[(∀p, where p = duplicateFrames_mainW = 0)  
duplicateFrames_mainW is a Boolean variable that is set to True if two frames having same name are active simultaneously. This property requires the absence of a frames error where frames having same names are active simultaneously. |
| **Form filling**                   | The ability of modeling form based pages, and to populate those forms with different values automatically or semi-automatically. |
| **Longest path**                   | The length of a path consists of the number of bytes, or the number of links, that must be downloaded in order to follow it. An example formula in Constructive µ – calculus [A01]:  
In MCWEB, there is an extension that enables the computation of the longest and shortest paths in a set of webnodes. To find the all-pair longest path between webnodes of a domain ∆, MCWEB post-processes the output of the formula, a: home page a; Post(x): webnodes reachable by following one edge from x, in_ domain ∆: holds for a webnode W if there is an URLpage S in W such that S contains the substring ∆;  
(∀x, y (Post(x) ∩ in_ domainS), x)  
The computation of the all-pair longest path can provide information about the bottlenecks in the navigation of a site. |
| **Dynamic Navigation System input**| Modeling using input provided by the user or system to generate a different target for the same navigation link. For example, links that are available only if the user has given access rights; search engines such as Google, which depend on user’s keywords to generate a document containing dynamically generated links representing the result of the search. |
| **User input**                     | Modeling and checking the user interactions with the browser that may affect the business logic of web application; this could include modeling the back button, the forward button, and URL rewriting. The following sequence of steps generates the Amazon bug [18], a well known bug caused by ignoring user interactions with the browser.  
Step 1: The shopping cart of the user is empty and the user browses the web site.  
Step 2: The user adds an item Item1 to the shopping cart.  
Step 3: The user decides that he does not want to buy Item1 after all, but instead of deleting it from the shopping cart he uses the “back” button to return to the previous shopping cart which is empty.  
An example formula using Alloy, [RA05]:  
all x: State | x.browser.display.cHasItems = x.browser.bl.scHasItems  
A major requirement of the model is to guarantee the integrity of the system by ensuring that the list of items that are displayed on the browses current web page (cHasItems) is identical to the contents of the shopping cart (scHasItems); bl (business logic) is an abstract class that relates the browser to its data content. Using Alloy Analyzer one can see that the assertion fails. |

Table 2.1: Desirable Properties for Website modeling.
### Static content properties

**Incomplete WP**

The model should enforce that a given web page contains some information, links between pages exist and that the web page exists. An example formula using Rewriting-based specification language, [ABF05]:

\[ L \rightarrow \text{tr} : \text{If } L \text{ is recognized in some web page of } W \text{ then } r \text{ must be recognized in some web page of } W \text{ which contain the marked part of } r. \]

\[ \text{member}(\text{name}(X), \text{surname}(Y)) \rightarrow \text{hpage} (\text{name}(X), \text{surname}(Y), \text{status}(i)) \]

If there is a Web page containing a member list, then for each member a home page exists containing (at least) the name, the surname and the status of this member.

---

**Incorrect WP**

The model should enforce that the information provided on a web page is valid based on the application requirements. An example formula using Rewriting-based specification language, [ABF05]:

\[ L \rightarrow \text{error} | C , \text{If } L \text{ is recognized in some web page of } W \text{ and all the expressions represented in } C \text{ are evaluated to True (or } C \text{ is empty), the web page is incorrect.} \]

\[ \text{project}(\text{year}(X)) \rightarrow \text{error} | \text{X in}[0-9] \ast, X < 1990 \]

If there is a Web page containing a project year, where the year is numeric and less than 1990, it should be replaced with error.

---

### Dynamic content properties

**Incomplete WP**

Check that the syntax and semantics (specifically the incomplete property) of dynamically generated content that results from the execution of scripts on the application server.

*(None of the examined modeling methods use this kind of checking.)*

**Incorrect WP**

Check that the syntax and semantics (specifically the incorrect property) of dynamically generated content that results from the execution of scripts on the application server.

*(None of the examined modeling methods use this kind of checking.)*

**New connection**

Model connections whose source and target is determined by the system at run time. For example:

- Link an electronic book which has 200 chapters; linking each one individually in the content list is time consuming; time could be saved by using an algorithm that can use user selected text (in the content list) to automatically links the chapter with a title corresponding to the selected text.

**New content**

Model new generated components that the user can’t determine until run time.

---

### Instruction processing

**Server-side execution**

If the method provides a model for code that is executed on the client or the server, can the method specify the location of such execution (i.e. is it executing on the client or server side).

**Client-side execution**

---

### Security properties

**Access control**

Check if the access control rules specified in the application requirements are violated in the web application. An example in computational tree logic (CTL), [CMRT06]:

A member cannot have administrator functions and an anonymous user cannot view pages belonging to a member

\[ \text{AG} (\text{member} \rightarrow \text{all}); \text{AG} (\text{noLog} \rightarrow (\text{partialA} \text{all})) \]

All: administrator functions; partial: member functions.

---

**Session/cookie**

Check if the inactive period of the current session is over a time limit (e.g. 5 minutes). If the inactive period is greater than the time limit, the HTTP request must be redirected to an authentication page to re-authenticate the user. An example in first-order logic [KLH00]:

\[ (\exists s \in \text{Session}) (s = \text{this.Session()} \land s.\text{Inactive()} > S) \rightarrow \text{this.redirect(Auth)} \]

The this.Session() is a function that returns a session object; s.Inactive() is a function that returns the inactive period for the session object s. The this.redirect (Auth) specifies that the HTTP request is redirected to the Auth server-page.

---

<table>
<thead>
<tr>
<th>Feature modeled or property checked</th>
<th>Example feature or property description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static content properties</td>
<td>The model should enforce that a given web page contains some information, links between pages exist and that the web page exists. An example formula using Rewriting-based specification language, [ABF05]:</td>
</tr>
<tr>
<td>Incomplete WP</td>
<td>[ L \rightarrow \text{tr} : \text{If } L \text{ is recognized in some web page of } W \text{ then } r \text{ must be recognized in some web page of } W \text{ which contain the marked part of } r. ]</td>
</tr>
<tr>
<td></td>
<td>[ \text{member}(\text{name}(X), \text{surname}(Y)) \rightarrow \text{hpage} (\text{name}(X), \text{surname}(Y), \text{status}(i)) ]</td>
</tr>
<tr>
<td></td>
<td>If there is a Web page containing a member list, then for each member a home page exists containing (at least) the name, the surname and the status of this member.</td>
</tr>
<tr>
<td>Incorrect WP</td>
<td>The model should enforce that the information provided on a web page is valid based on the application requirements. An example formula using Rewriting-based specification language, [ABF05]:</td>
</tr>
<tr>
<td></td>
<td>[ L \rightarrow \text{error}</td>
</tr>
<tr>
<td></td>
<td>[ \text{project}(\text{year}(X)) \rightarrow \text{error}</td>
</tr>
<tr>
<td></td>
<td>If there is a Web page containing a project year, where the year is numeric and less than 1990, it should be replaced with error.</td>
</tr>
<tr>
<td>Dynamic content properties</td>
<td>Check that the syntax and semantics (specifically the incomplete property) of dynamically generated content that results from the execution of scripts on the application server.</td>
</tr>
<tr>
<td>Incomplete WP</td>
<td><em>(None of the examined modeling methods use this kind of checking.)</em></td>
</tr>
<tr>
<td>Incorrect WP</td>
<td>Check that the syntax and semantics (specifically the incorrect property) of dynamically generated content that results from the execution of scripts on the application server.</td>
</tr>
<tr>
<td></td>
<td><em>(None of the examined modeling methods use this kind of checking.)</em></td>
</tr>
<tr>
<td>New connection</td>
<td>Model connections whose source and target is determined by the system at run time. For example:</td>
</tr>
<tr>
<td></td>
<td>- Link an electronic book which has 200 chapters; linking each one individually in the content list is time consuming; time could be saved by using an algorithm that can use user selected text (in the content list) to automatically links the chapter with a title corresponding to the selected text.</td>
</tr>
<tr>
<td>New content</td>
<td>Model new generated components that the user can’t determine until run time.</td>
</tr>
<tr>
<td>Instruction processing</td>
<td>If the method provides a model for code that is executed on the client or the server, can the method specify the location of such execution (i.e. is it executing on the client or server side).</td>
</tr>
<tr>
<td>Server-side execution</td>
<td><em>(None of the examined modeling methods use this kind of checking.)</em></td>
</tr>
<tr>
<td>Client-side execution</td>
<td><em>(None of the examined modeling methods use this kind of checking.)</em></td>
</tr>
<tr>
<td>Security properties</td>
<td>Check if the access control rules specified in the application requirements are violated in the web application. An example in computational tree logic (CTL), [CMRT06]:</td>
</tr>
<tr>
<td>Access control</td>
<td>A member cannot have administrator functions and an anonymous user cannot view pages belonging to a member</td>
</tr>
<tr>
<td></td>
<td>[ \text{AG} (\text{member} \rightarrow \text{all}); \text{AG} (\text{noLog} \rightarrow (\text{partialA} \text{all})) ]</td>
</tr>
<tr>
<td></td>
<td>All: administrator functions; partial: member functions.</td>
</tr>
<tr>
<td>Session/cookie</td>
<td>Check if the inactive period of the current session is over a time limit (e.g. 5 minutes). If the inactive period is greater than the time limit, the HTTP request must be redirected to an authentication page to re-authenticate the user. An example in first-order logic [KLH00]:</td>
</tr>
<tr>
<td></td>
<td>[ (\exists s \in \text{Session}) (s = \text{this.Session()} \land s.\text{Inactive()} &gt; S) \rightarrow \text{this.redirect(Auth)} ]</td>
</tr>
<tr>
<td></td>
<td>The this.Session() is a function that returns a session object; s.Inactive() is a function that returns the inactive period for the session object s. The this.redirect (Auth) specifies that the HTTP request is redirected to the Auth server-page.</td>
</tr>
</tbody>
</table>

---

Table 2.2: Desirable Properties for Website modeling (Cont.)
and other features related to estimating the cost of navigation, such as longest path analysis.

- **Dynamic navigation properties**: This analysis focuses on aspects that make the navigation dynamic. That is, the same link may lead to different pages depending on given inputs. The inputs could be user inputs transferred via forms, or system inputs depending on some state in the server such as date, time, session information, access control information or information in hidden fields.

- **Interaction navigation properties**: This analysis focuses on properties that are related to user navigation that happens outside the control of the web application, such as user interaction with the browser. This includes features such as use of the back or forward buttons.

2. Web Content

- **Static content properties**: Consistency of the original web page content with respect to syntax and semantics. Two properties are explored in this category, completeness and correctness. When verifying the completeness of a web application the model should enforce that a given web page contains some information, links between web page exist and sometimes even check that the web pages exist (broken links). Correctness implies that the information provided on a web page is valid based on the application requirements.

- **Dynamic content properties**: Consistency of the syntax and semantics of dynamic content. This analysis requires the ability to check the dynamically generated content that results from the execution of script code by the application server. Some technologies are also able to generate new connections, some of which may be to a different server. New web components could be generated at run time, and these components must also be analyzed.
3. Web Behavior

- **Security properties:** This issue is related to access control mechanisms that are employed on the web content or web links. This issue could also be employed on the back-end, as the database may contain data reserved to specific users. Non-authorized users must not be able to access such data. These properties are also tied to session control mechanisms.

- **Instruction processing properties:** These issues include both server and client-side execution. We define client-side execution as any process changing the state of the application without communication with the web server. Server-side execution is defined by all instructions processed on a web server in response to a client’s request. A modeling method should be able to model these features and to recognize whether execution is done on the server or on the client.

Tables 2.1 and 2.2 provide descriptions and examples of these properties. The list is not complete, but it provides a summary of the set of properties that the reviewed methods are interested in modeling, checking or testing. We use symbolic keys in the tables (e.g., SDMP02) to refer to the surveyed methods. Please refer to Table 2.3 in Section 2.4 for details on the symbolic keys.

### 2.3 Comparison and Categorization Criteria

In our study we reviewed 24 different modeling methods that are applied in the field of testing and verification of web applications. Following is a brief description of the main comparison criteria that are used in our review:

1. **Feature Type:** The web application features that are being captured by the proposed models, and the properties that the modeling methods are capable of checking. These features are categorized first in Section 2.2.4 based on the level of web application...
modeling into features related to web application navigation, content, or behavior. In this section, we add another categorization dimension, static, dynamic, or interactive. This additional information can help us to identify the improvements that the modeling methods are trying to achieve at each level of web application modeling. We relate each category to the properties described in Section 2.2.4 at the end of its description.

- **Static Features:** These include the static properties of web applications, such as links that connect an HTML page with other HTML pages. When the user clicks on a static button or a static link, a request is sent to the server in order to fetch a page. The server responds to the request by retrieving the required page from its storage and sends it back to the client. In this category properties from Section 2.2.4 related to static navigation and static content can be checked.

- **Dynamic Features:** These features include dynamic links and dynamic content properties. Dynamic links describe the connection between HTML pages and code that must to be executed on the server in order to generate the required information, build it into an HTML page, and return it to the client. The processing done by the server may depend on input that is provided by the user or the system. User inputs are usually sent by filling a form or by hidden fields in the HTTP request. System inputs depend on the server state, such as server time, or on some kind of interaction with other resources, such as database servers or web objects. The output could be constructed as new content, or a link in a new HTML page. Properties from Section 2.2.4 that fall into this category are those related to dynamic navigation, dynamic content, security, and instruction processing properties.

- **Interaction Features:** The browser’s influence on the navigation behavior of the web application should be taken into consideration when modeling or analyzing
web applications, since user interaction with the browser can interactively modify navigation paths. This category includes properties from Section 2.2.4 related to user interaction with the browser.

2. Notation: Modeling methods use different notations; some of them are formal, while others are either semi-formal or informal. The main notations used by the reviewed methods are:

- Statecharts [97].
- UML and OCL [2] [1]
- UML-based Web Engineering (UWE) [113]
- Alloy [107].
- Finite State Machines (FSM) [167]
- Directed Graphs and Control Flow Graphs (CFG) [89] [129]
- Specification and Description Language (SDL) [80].
- Term Rewriting Systems (TRS) [110]

3. Levels of modeling: Web application modeling can be viewed from different perspectives. We compare the modeling methods here according to three basic levels: content, structure (navigation), and behavior. These three levels in turn could have a static or a dynamic flavor.

4. Application of the model: In our study we focus on methods that are concerned with modeling web applications for the purpose of testing or verification; this also could include design verification.

5. Source code required?: Modeling methods may apply a white-box analysis, which requires source code, or a black-box analysis which does not require source code.
6. **Model optimization:** Complex systems in general have a state explosion problem or they generate a large complex model. In all cases such models need some sort of optimization. In web applications, this problem becomes a major challenge to the success of any method that attempts to analyze and model a scalable web system.

7. **Tool support:** We note whether the method is supported by a proposed or existing tool.

### 2.4 Comparative Analysis.

Our study produced two different views of the surveyed methods: a general categorization by modeling level, and a detailed comparison by property coverage. To minimize space in tables and text, we identify each modeling method with a key based on the last name of authors and the date of publication. Table 2.3 gives these keys along with the full name in text and the citation for the method.

Table 2.4 summarizes the first view, categorizing each of the methods based on the level of modeling: as interaction behavior modeling methods, navigation modeling methods, content modeling methods or hybrid modeling methods (methods that model more than one level). In each category, methods are sorted according to the notation used by the method. At the same time, comparison between the methods was also done based on the other criteria presented in section 4.

The second comparison, shown in Table 2.5, compares some of the more specific details of methods in the same category, in particular, and with other methods in other categories in general. The comparison is based on a combination of feature type and the level of web application modeling, using the comparison criteria outlined in Section 2.2.4 as desirable properties for web site modeling.

In the remainder of this section we discuss and compare the characteristics of the methods summarized in these tables. Our presentation is organized by the levels in Table 2.4,
### Table 2.3: Reference linking summary tables with text

<table>
<thead>
<tr>
<th>Shortcut Keys</th>
<th>Full name in Text</th>
<th>Reference No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LuccaP03</td>
<td>Di Lucca and Di Penta 2003</td>
<td>[68]</td>
</tr>
<tr>
<td>GFKF03</td>
<td>Graunke et al. 2003</td>
<td>[90]</td>
</tr>
<tr>
<td>LK04</td>
<td>Licata and Krishnamurthi 2004</td>
<td>[121]</td>
</tr>
<tr>
<td>CZ04</td>
<td>Chen and Zhao 2004</td>
<td>[48]</td>
</tr>
<tr>
<td>BA05</td>
<td>Bordbar and Anastasakis 2005</td>
<td>[37]</td>
</tr>
<tr>
<td>ABGR07</td>
<td>Anastasakis et al. 2007</td>
<td>[27]</td>
</tr>
<tr>
<td>ABF+07</td>
<td>Alpuente et al. 2006</td>
<td>[22,24,21,23]</td>
</tr>
<tr>
<td>ABF06</td>
<td>Alpuente et al. 2007</td>
<td></td>
</tr>
<tr>
<td>CF07</td>
<td>Coelho and Florido 2007</td>
<td>[55,56]</td>
</tr>
<tr>
<td>CF06</td>
<td>Coelho and Florido 2006</td>
<td></td>
</tr>
<tr>
<td>Con99</td>
<td>Conallen, 1999</td>
<td>[57]</td>
</tr>
<tr>
<td>BMT04</td>
<td>Bellettini et al. 2004</td>
<td>[34]</td>
</tr>
<tr>
<td>RT00</td>
<td>Ricca and Tonella 2000</td>
<td>[150]</td>
</tr>
<tr>
<td>dA01</td>
<td>MCWEB, de Alfaro 2001</td>
<td>[62,63]</td>
</tr>
<tr>
<td>dAHM01</td>
<td>de Alfaro et al. 2001</td>
<td></td>
</tr>
<tr>
<td>SDMP02</td>
<td>Sciascio et al. 2002</td>
<td>[74,75]</td>
</tr>
<tr>
<td>SDMP03</td>
<td>Sciascio et al. 2003</td>
<td></td>
</tr>
<tr>
<td>SDM+05</td>
<td>Sciascio et al. 2005</td>
<td>[76]</td>
</tr>
<tr>
<td>CMRT06</td>
<td>Waver, Castelluccia et al. 2006</td>
<td>[44]</td>
</tr>
<tr>
<td>WP03</td>
<td>Winckler and Palanque 2003</td>
<td>[172]</td>
</tr>
<tr>
<td>HIf06</td>
<td>FARNav, Han and Hofmeister 2006</td>
<td>[98]</td>
</tr>
<tr>
<td>SM03</td>
<td>Syriani and Mansour 2003</td>
<td>[161]</td>
</tr>
<tr>
<td>KLH00</td>
<td>Web Test Model WTM, Kung et al. 2000</td>
<td>[118]</td>
</tr>
<tr>
<td>BFG02</td>
<td>VeriWeb, Benedikt et al. 2002</td>
<td>[36]</td>
</tr>
<tr>
<td>HPS04</td>
<td>Haydar et al. 2004</td>
<td>[102]</td>
</tr>
<tr>
<td>AOAA05</td>
<td>FSMWEB, Andrews et al. 2005</td>
<td>[28]</td>
</tr>
<tr>
<td>WO02</td>
<td>Wu and Offutt 2002</td>
<td>[173]</td>
</tr>
<tr>
<td>TR04</td>
<td>Two-layer-model, Tonella and Ricca 2004</td>
<td>[164]</td>
</tr>
<tr>
<td>TR02</td>
<td>Tonella and Ricca 2002</td>
<td>[163]</td>
</tr>
<tr>
<td>KZ06</td>
<td>Knapp and Zhang 2006</td>
<td>[113]</td>
</tr>
<tr>
<td>GSDA07</td>
<td>Guerra et al. 2007</td>
<td>[93]</td>
</tr>
<tr>
<td>Method Name</td>
<td>Feature type</td>
<td>Notation</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>LuceoP03</td>
<td>Interaction</td>
<td>StateCharts</td>
</tr>
<tr>
<td>GKF03</td>
<td>Interaction</td>
<td>Abstract model, use lambda calculus</td>
</tr>
<tr>
<td>LK04</td>
<td>Interaction</td>
<td>WebCFG</td>
</tr>
<tr>
<td>CZ04</td>
<td>Interaction</td>
<td>Labeled transition</td>
</tr>
<tr>
<td>BAS5</td>
<td>Interaction</td>
<td>UML (Web application structure) (OCL; behavior of the model)</td>
</tr>
<tr>
<td>AAB+07</td>
<td>Static</td>
<td>Partial rewriting</td>
</tr>
<tr>
<td>CF07</td>
<td>Static</td>
<td>Logic FL- Prolog extension (XCentric)</td>
</tr>
<tr>
<td>Con09</td>
<td>Static</td>
<td>Extended UML</td>
</tr>
<tr>
<td>BMT04</td>
<td>Static + dynamic</td>
<td>UML-meta Model + UML state diagram</td>
</tr>
<tr>
<td>R100</td>
<td>Static</td>
<td>Directed graph</td>
</tr>
<tr>
<td>AAB+01</td>
<td>Static + dynamic</td>
<td>Directed graph</td>
</tr>
<tr>
<td>SDM01</td>
<td>Static + dynamic</td>
<td>Web graph</td>
</tr>
<tr>
<td>SDM1-05</td>
<td>Static + dynamic</td>
<td>(WAG)/Web application graph + extension to Kripke structure</td>
</tr>
<tr>
<td>WP03</td>
<td>Static + dynamic</td>
<td>Extended StateCharts</td>
</tr>
<tr>
<td>HFB06</td>
<td>Static + dynamic</td>
<td>StateCharts</td>
</tr>
<tr>
<td>SM03</td>
<td>Static + dynamic</td>
<td>SDL</td>
</tr>
<tr>
<td>KLM00</td>
<td>Static + dynamic</td>
<td>Control-flow graph, data flow graph, and finite state machines (OSM; object state diagram)</td>
</tr>
<tr>
<td>BFG02</td>
<td>static + dynamic</td>
<td>Directed graph</td>
</tr>
<tr>
<td>HPS04</td>
<td>Static + dynamic</td>
<td>System of communicating automata</td>
</tr>
<tr>
<td>JAA05</td>
<td>static + dynamic</td>
<td>Hierarchies of Finite State Machines (FSM)</td>
</tr>
<tr>
<td>WOD2</td>
<td>Interaction + static + dynamic</td>
<td>Regular expression</td>
</tr>
<tr>
<td>TR04</td>
<td>Static + dynamic</td>
<td>(model navigation layer) + CFG (client &amp; server code)</td>
</tr>
<tr>
<td>KZ06</td>
<td>Static + dynamic</td>
<td>Extended UML (UWE)</td>
</tr>
<tr>
<td>GSDA07</td>
<td>Static + dynamic</td>
<td>Architecture Development Method (ADM)</td>
</tr>
</tbody>
</table>

Table 2.4: Summary of Methods Categorized by Modeling Level
### Table 2.5: Detailed Comparison of Methods by Properties Covered

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LuccaP03</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>GFKF03</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LK04</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CZ04</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CF07</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CF06</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Con99</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>BMT04</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>BA05</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ABGR07</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ABF+07</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ABF06</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SDMP02</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SDM+05</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>WP03</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>WP03</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>HH06</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>KLH00</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>BFG02</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>HPS04</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>AOA05</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>WO02</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>TR04</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>KZ06</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>GSDA07</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 2.5: Detailed Comparison of Methods by Properties Covered
that is, we first discuss interaction modeling methods in section 2.4.1, then content modeling methods in section 2.4.2 followed by navigation modeling methods in section 2.4.3, and finally hybrid methods in section 2.4.4. The categories are not disjoint; some methods are discussed more than once since they have aspects that address multiple levels, but we try to make the presentation consistent with Table 2.4 otherwise. For example methods in each category are discussed based on the notation employed by those methods in order to identify how specific notation can affect the capability of the modeling methods to capture different features.

2.4.1 Interaction Behavior Modeling Methods

Dealing with user operations (interactions) is very important. Such interactions are problematic, for example: clicking the back button forces the computation to resume at a prior interaction point; submitting multiple forms then clicking the back button causes computations at the same interaction point to resume many times. These operations happen in the browser and are not reported to the web application. Consequently, the browser interacts with the web application in an unexpected manner. Modeling methods that do not take into account this kind of behavior are incomplete and unrealistic. The methods discussed here refer to the first section of Tables 2.4 and 2.5. The presentation follows the chronological order of the methods unless specific relationships between methods need to be identified.

Di Lucca and Di Penta (LuccaP03) [65] model the browser loading a page as a Statechart with four basic states: Back Disabled, Forward Disabled (BDFD); Back Enabled, Forward Disabled (BEFD); Back Enabled, Forward Enabled (BEFE); Back Disabled, Forward Enabled (BDFE), as shown in Figure 2.2(a). The user navigation is modeled as transitions between those basic states and the transition has four different labels: forward, backward, reload and link, to indicate if the transition is activated by clicking on regular links or by clicking browser navigational buttons. The state transitions are also labeled with guard conditions to specify the navigation sequence restrictions. Possible interactions with the
browser are generated using test cases to satisfy defined coverage criteria, such as all states, all transitions, all transition k-tuples, and all round-trip paths. Potential inconsistencies are collected by executing the browser test cases and comparing the results with an oracle, taking into account the verification of the chosen coverage criteria. The authors propose a way to integrate their browser model with other web application testing models to make them browser interaction aware methods. Unlike the following methods, this method is applied in web application testing rather than verification.

Graunke et al. (GFKF03) [88] detect data inconsistency problems such as the “Orbitz” bug [120] and bugs caused by form input. Problems are detected dynamically by modifying the server run-time system. An abstract model encodes user interactions with either the application or the browser using its navigation buttons (e.g., forward, backward) in terms of three rewriting rules (pattern/replacement pairs describing changes in state): fillform, switch, and submit. The model is focused on sequential web interaction and thus is limited to a single server and a single client (Figure 2.2(b)). Dynamic features are limited to client-side forms with arbitrary client-side navigation (such as back and forward buttons) represented using the rewrite rules, allowing for detection of navigation bugs such as the Orbitz problem.

Licata and Krishnamurthi (LK04) [120] have built a model checker that uses the Graunke et al. model to reduce user operations to two main rewriting rules: submit and switch. Their method differs from Graunke et al. in that it is a static method that can provide guarantees about all possible execution sequences using a control flow graph (CFG) to model the web application. User operations are added to extend the graph to a WebCFG constructed automatically from the source of the application. The WebCFG is built using a standard CFG construction technique followed by a graph traversal to add web interaction nodes and edges which model the user interactions with the browser. The resulting model is checkable using language containment, implemented as constraint automata optimized by automatically generating constraints to rule out redundant forward paths.
CHAPTER 2. RELATED WORK

Figure 2.2: Interaction Behavior Modeling Methods: the Di Lucca and Di Penta (LuccaP03) [65], Graunke et al. (GFKF03) [88] and Licata and Krishnamurthi (LK04) [120] models.

Chen and Zhao (CZ04) [45] model user interactions with web browsers using a much more complete model. As well as modeling the back button, forward button and URL rewriting functionalities, their method is distinguished from other methods in its ability to represent the history stack and its impact on navigation, the local cache and its influence on the freshness of web pages, and authentication sessions. While this method builds a navigational model taking into account interaction with the browser, dynamic links are not represented in the assumed page navigation diagram, and in the functionality provided by session/cookie techniques of the application under test, Chen and Zhao have chosen to model only session control. The proposed model (Figure 2.3(d)) is a labeled transition
system (LTS) consisting of a set of states $S$, a set of labels $L$ and a set of transition rules mapping between states. States maintain a page id to denote the current page and an error page for invalid accesses, a history stack of current URLs in the session history, a set of page ids for locally cached pages, and a boolean to represent the authentication status of the session. Labels encode user actions as entry (a manually entered URL), back, forward, err (navigation redirected to a special error page), or one of a number of specific user actions such as signin or signout. A fresh/cache flag indicates whether the resulting page is from the server or the local cache.

In this model, the example rule shown in the Figure 2.3(D) can be translated as: “If the user can sign-in from page $p$ into page $q$ and $q$ is in the cache, then there is a transition from the current state $p$ to the one with page $q$, where $q$ is put into the history stack”. In the new state, the guard is set true to indicate that the session for authentication is now open, and the label on the transition indicates that this is a sign-in action.

Bordbar and Anastasakis (BA05) [33] create an abstract model, called Abstract Description of Interactions (ADI) to depict the interactions between the browser and business logic. This model consists of four classes: the browser with its functionality, the business logic that relates to the browser and it’s data content, the data that are exchanged between the server and the browser, and the generic functionality of the web page that contains data which could be altered from the user interface. Figure 2.3(e) shows their proposed model.

While Licata and Krishnamurthi (LK04) [120] built their own model checker, Anastasakis et al. [23] use the Alloy [107] model checker to find interaction bugs. The main difficulty of this method is the process of building the ADI model from the Platform Independent Models (PIM) for web applications which are large and complex. The construction process requires a projection of the PIM and deletion of the unrelated model elements, which is currently done manually.

To summarize, the authors in this section are able to model the interactions of web applications with the browser by using abstract models represented in different notations. All
Figure 2.3: Interaction Behavior Modeling Methods (cont’d): the Bordbar and Anastasakis (BA05) [33] and Chen and Zhao (CZ04) [45] models.

of the surveyed methods are able to model the basic browser back and forward operations. Some of the methods also model other browser features such as the history stack, the page cache and user sessions. These methods manually integrate the interaction model with a static navigational model. Some detect bugs in the interaction between the web application and the browser by implementing their own model checker or by using existing testing and model checking techniques. None of the models demonstrate integration with dynamic web applications, or how dynamic features affect the interaction models.
2.4.2 Content Modeling Methods

These methods check the completeness and the correctness properties of web application content. As the examples in Table 2.2 demonstrate, these methods must be able to enforce and check that certain information is available on a given web page, links between pages exist, or even the existence of the web pages themselves (completeness property). Furthermore, web application content may need to be checked against semantic conditions to see if they are met by the web document (correctness property). This kind of checking must handle both static and dynamic content. The methods discussed here refer to the second section of Tables 2.4 and 2.5. The presentation follows the chronological order of the methods unless specific relationships between methods need to be identified.

Alpuente et al. in (ABF+06,ABF07) [18, 20, 17, 19] propose a method for verifying static web applications for both syntactic and semantic properties using partial rewriting. In this method, web pages are modeled as the ground terms (constant formulae) of a term algebra, and the entire web site is represented as a set of such ground terms [110]. Rewriting rules specify pattern/replacement pairs for modifications to the formulae. A checking specification is a triple $(R, IN, IM)$, where $R$ is a set of global function definitions used in the rules, $IN$ is the set of correctness constraints encoded as partial rewriting rules, and $IM$ is the set of completeness constraints encoded as partial rewriting rules. Table 2.2 shows an example of completeness and correctness rules using this method.

The Alpuente et al. method is implemented in GVerdi, a graphical evolution of the VERDI verification system, and improved in a new prototype WebVerdi – M (Web Verification and Rewriting for Debugging Internet sites with Maude) (ABF+07) [19], which implements a more scalable, efficient and usable verification system that can be used as a web service from anywhere by any user. GVerdi – R is an improved GVerdi system able to repair faulty web pages semi-automatically [20]. Ballis and Romero in [27] have improved the level of automation of the GVerdi-R system by decreasing the amount of information.
to be changed and the number of repair actions to be made to correct a faulty web site.

In place of the partial rewriting applied in the Alpuente et al.’s approach, which uses tree simulation for recognizing patterns inside semi-structured documents (HTML/XML), Coelho and Florido (CF06,CF07) [52, 53] use an extension of Prolog called XCentric to check and repair the syntactic and semantic properties for the content of XML-based web sites. The XML web document is translated into a temporary document which is composed of logical terms corresponding to the XML tags in the original document, then, a sequence of checking and repairing rules is applied on the translated document to verify the semantic of its content. The verified document is then translated back to its original representation, XML. The framework was first implemented in Coelho and Florido (CF06) VeriFLog [52], then improved in Coelho and Florido (CF07) [53].

While the focus of the above methods is on verifying the static content of web applications, up until now none has studied the verification of dynamic content for correctness and completeness. Such a study will be required to help with the increasing dynamism of web applications.

2.4.3 Navigation Modeling Methods

The methods discussed here refer to the third section of Tables 2.4 and 2.5. In this group, some methods share the same underlying modeling notation, so we discuss methods based first on the notation then take into consideration the chronological order. The discussion begins with the UML-based models, continues with graph-based models, then Statechart-based models and finally an SDL-based model.

**UML Navigation Models.** Conallen (Con99) [54] extends UML notation to represent web application components with both static and dynamic features. These extensions include stereotypes, tagged values and constraints. Stereotypes in UML allow the definition of new semantics for a modeling element. In this method, this idea is used to define two kinds of web pages, client pages and server pages, and a web page is modeled as a class with the semantics
of either a client or server page as defined by the stereotype. The relation between server
and client pages is defined using the stereotype \((\textit{build})\) and relations between web pages
are expressed using the stereotype \((\textit{link})\). Other HTML elements, such as JavaScript,
Java applets, ActiveX controls, forms, and frames, are similarly represented as stereotyped
classes. Tagged values, which represent new properties that can be associated with model
elements, are used here to define the parameters that are passed along with a link request,
for example the \texttt{link} association tagged value \texttt{Parameters} is a list of parameter names
(and optional values) that are expected and used by the server page that processes the
request. Finally, constraints specify new conditions under which a model can be considered
"well-formed".

While Conallen does not himself present a modeling method for any of the web appli-
cation development phases, his UML extensions form the basis of many modeling methods
applied in different phases of web development. The main benefit of this method is that it
allows representation of all components of a web application using standard UML notation.

Like Conallen, Tonella and Ricca (TR02) [163] propose a UML meta-model for modeling
web applications, specifically to represent static navigation. The main difference between
the two approaches is that Conallen’s model describes web applications from a design point
of view, without proposing a method for design or for designing the navigation aspects of
a web application. Tonella and Ricca on the other hand use their model in a reverse engi-
neering method, in order to extract a model of the web application to aid in maintenance
and evolution. Their model is therefore aimed more at analysis rather than design, and
specifically at modeling and analyzing navigation features. The Tonella and Ricca method
is semi-automatic; it requires user interaction to complete the model extraction process.
Since user input from POST access methods is not normally logged by most web servers,
the web server or the web application must be augmented with extra tracing. Also, server
responses may be cached (by the client or by a proxy), so some values must be recon-
structed heuristically. The input values used during model extraction are not generated
Figure 2.4: UML-Based Models: the Tonella and Ricca (TR02) [163] and Conallen (Con99) [54] models, and the Knapp and Zhang (KZ06) [111] UWE basis model.

automatically, but instead are provided by extensive user interaction. Source code running on the browser, such as Javascript and applets, that are executed by the web server are not currently analyzed. However, their analysis allows treatment of these components as white-boxes. Figure 2.4(b) shows the Tonella-Ricca model.

Bellettini et al. (BMT04) [30] use a model similar to the Conallen [54] and Tonella and Ricca [163] models to extract a model instance (in particular class and state diagrams) from the analyzed web application. Their method differs from Tonella and Ricca in that Bellettini’s WebUML requires minimal user interaction. Class diagrams are used to describe the structure and the components of the web application (forms, frames, Java applets, input fields, cookies, scripts, and so on), while state diagrams are used to represent the behavior
and navigational structures (client-server pages, navigation links, frames sets, inputs, and scripting code flow control). WebUML employs a mix of techniques based on source code static and dynamic analysis. Static analysis is performed using simple parsers based on a pattern matching scanner. Dynamic analysis is performed through source code mutational techniques combined with simulated web application execution. This technique avoids heavy language analysis, but requires the implementation of a simple map of mutant operators.

Castelluccia et al. (CMRT06) [40] and Di Sciascio et al. (SDM+05) [73] use the Conallen model in order to build a diagram for the web application, where the aim is to verify the design of the application. In order to apply model checking techniques to any model, the models must be formal. Di Sciascio et al. implement a component, XMI2SMV, that converts UML diagrams in XMI format into a Web Application Graph (WAG). The WAG can be translated, in turn, into a Symbolic Model Verifier (SMV) model which is given as input to the NuSMV model checker [50].

A similar conversion idea was applied by Bordbar and Anastasakis (BA05) [33] which is described in section 2.4.1. They also designed a model translation tool, UML2Alloy, that in their case maps from a UML diagram to an Alloy model, which can then be model checked using Alloy.

> Graph Navigation Models. UML diagrams provide a valid support to verify web applications requirements; however, they need to be turned into a formal model; so other researchers prefer to start with a formal model rather than doing the conversion.

In MCWEB [60, 59] de Alfaro et al. model web applications using a webgraph, an extension to the simple flat directed graph model in which web pages are modeled as nodes, and links, anchors and frame (sub-frame) tags are modeled as edges. The model supports natural connectivity analysis of the web, where web graph nodes (webnodes) form a hierarchical frame structure, generated by the grammar

\[
\text{webnode ::= URLpage (name webnode)*}
\]

A URLpage is the result of fetching a given URL from the web application using a GET
Figure 2.5: Graph-Based Models: the MCWEB [60], Ricca and Tonella (RT00) [150] and Di Sciascio et al. (SMP02, SMP03) [72, 71] models.

method, and each \((\text{name webnode})\) pair consists of the name of a subframe and the subframe content. The edges of the graph correspond to links between web pages, where the destination webnode is obtained by updating the frame structure as specified in the HTML standard. Based on this model, de Alfaro verifies properties expressed in constructive \(\mu-calculus\) against static web applications. The MCWEB tool downloads a web site from a given URL and builds an abstract representation of it in the form of a graph. Figure 2.5(a) shows an example of the structure of a web site in this model.

Like de Alfaro, Ricca and Tonella (RT00) [150] address the issue of hierarchical frame structure of web pages but in a different way. In their model, nodes and edges of the graph representation are partitioned into different subsets. The nodes are split into the set of
all web pages, the set of frames for one web page, and the set of all frames. The edges are split into three subsets according to the type of target node. This includes a set of hyper-links between pages or a relation showing the composition of web page into frames (E1); a set of the relations between frames and pages as they show which page in which frame is loaded (E2); and a set of relations showing the loading of a page into a particular frame (E3) as shown in Figure 2.5(b). The name of the frame is given as a label next to the link. This model is implemented in ReWeb. ReWeb can download and analyze a web site, and also provides a graphical user interface for searching and navigating the analysis results. ReWeb’s purpose is understanding web applications, but it is also used to generate a UML model that can be used by TestWeb, a tool implemented by Ricca for the purpose of web application testing [149]. ReWeb can be applied to static web pages with, or without, frame structure.

Di Sciascio et al. (SMP02,SMP03) [72, 71] model the frame structure of web pages by proposing a new state window that corresponds to a page that could be divided into one or more frames that, in turn, can load one or more web pages. Each node can be window, page or link, as shown in Figure 2.5(C). In this work, client-side scripts are modeled as static pages. For server-side scripts the dynamic redirection actions depend on user input from forms. Other dynamic features that require white-box analysis for the scripts, such as server contact with the database and other resources, are not considered. Such pages are considered static pages.

Di Sciascio et al. (SDM+05) [73] extend their previous model by adding actions to the set of states. Web applications are modeled as a finite state machine, where pages, links, windows and actions are states. Their method, AnWeb, is shown in Figure 2.6(C).

Castelluccia et al.(CMRT06) extend the web application graph in the verification tool WAver [40] by adding some important features related to web application access policies. The extension was made by assigning some resources to two categories of users:
• **Authorized users:** They can view specific areas of the web application not accessible to anonymous users.

• **Administrators:** They can insert or cancel a new user, view the list of authorized users and access all the resources of the web application.

By introducing this extension, Castelluccia et al. are able to model important features related to access control, and are able to verify properties related to this feature using axioms formulated in CTL (computational tree logic). The main advantage of this method is its ability to perform a priori verification of web application design by applying the verification process to the UML-design of web application in a single automated process using the verification tool WAver. Figure 2.6(e) shows the proposed model.

To summarize, graph-based models can be used to verify page reachability, dominators of the navigation path, navigation path length, strongly connected components, broken links and frame errors. It is also possible to do pattern matching to find out if the navigation model contains a diamond structure, tree structure or index structure.

**Statechart Navigation Models.** Winckler and Palanque (WP03) [172] have created an extension of Statecharts [98] called StateWebCharts (SWCs) which is similar to the Conallen model in that it creates an extension to an existing notation (Statecharts, instead of UML in the Conallen case). Thus they can help designers in building a formal model of their web application that can be directly model-checked. Currently SWCs are used to describe the navigation between documents rather than the interaction between objects. They have created a tool, SWCEditor, that supports their proposed notation and helps designers create, edit, visualize and simulate SWC models.

Han and Hofmeister (HH06) [96] also use a formal model for navigation. Their method, FARNav, uses Statecharts [97] to model adaptive navigation - web applications that can semi-automatically improve their organization and presentation by learning from visitor access patterns. In this model, the authors use parallel (ANDed) sub-states to represent
learned navigation patterns. The main sub-state contains a state machine with one state per web page, and transitions between pages for the navigation links. When a web application has only simple (non-adaptive) navigation, this sub-state comprises the entire navigation model. The model is created by observing the behavior of the web application and treating screens that provide similar kinds of content as one web page. They attempt to scale their model by making use of the hierarchical features of Statecharts. Like Di Sciascio et al., their model is converted into the SMV modeling language CTL to be verified. An existing approach is used for translation of the Statechart model to CTL. Since FARNav uses Statecharts, the limitations of state machines’ modeling capabilities make it difficult
to verify certain properties that are easy to verify with a graph-based model such as the de Alfaro, Di Sciascio et al. and Ricca and Tonella methods. For example, it is difficult to count the length of the navigation path, which can provide information about bottlenecks in the navigation of a site, using this method. In addition, none of these models supports adaptive navigation.

**SDL Navigation Models.** Syriani and Mansour (SM03) [161] use the Specification and Description Language (SDL) [80] to model web applications. SDL is a modeling language used to describe real-time systems. SDL is used to model the details of a system, which can then be simulated and proven, whereas UML is used to model at a higher level of abstraction [80]. Using SDL, Syriani and Mansour are able to model pages, hyperlinks, the behavior of the web page on both the client-side and the server-side, and client-server and distributed-server communication. In this method, each web page is represented by an agent, and hyperlinks between pages are represented by signals. A hyperlink in a web application represents a navigational path through the system, and this relationship is represented in the SDL model by a signal sent between agents using a channel association. Signals may carry parameters such as user name and password that are sent with the signals to login to a server. SDL tools are used to do the testing of their model and to help them in verifying the consistency of a web application implementation with its specification.

The approaches described in this section all use either a UML or graph based model to represent the navigation level of the web. While UML is the modeling standard for many applications, including the web, it may not be the appropriate choice for testing and verification. In order for the UML based models to apply the testing and verification techniques, the models should be translated into formal ones. The alternative choice is to use graph based models that can be directly tested or model checked. All the proposed UML-based models are able to capture the static features of navigation, and to represent the specific details of the web pages including the frame structure. In graph based models, nodes represent different modeling semantics in the different methods, from simple pages
in the flat model to pages with frame structure. Nodes are used also to represent windows, links, actions, and in some methods nodes are categorized into classes to reflect secure resources.

2.4.4 Hybrid Modeling Methods

Some researchers model the web application as a whole, using a single model for multiple levels of the application. After the model is expressed, they then attempt to solve the state explosion problem. Other methods analyze web applications at more than one level by using separate models. This section begins with the single model methods, followed by a discussion of methods using separate models for different web application levels. After both are discussed we give a general comparison of all the methods. In general, the methods discussed here refer to the fourth section of Tables 2.4 and 2.5. The presentation follows the chronological order of the methods unless we are identifying a specific relationship between methods.

Single Model Methods. VeriWeb (BFG02) [32], is a dynamic navigation testing tool for web applications. In this tool a systematic website exploration is performed under the control of VeriSoft, an existing tool for systematically exploring the state spaces of concurrent, reactive software systems. In VeriSoft the state space of the system is defined as a directed graph that represents the combined behavior of all the components of the system being tested. Paths in this graph correspond to sequences of operations (scenarios) that can be observed during executions of the system. In web applications the state space is the set of web pages (statically or dynamically generated) in the site that can be reached from some initial page. Reachable pages are the states of the website state space, while the set of possible actions from a given page defines the set of transitions from a given state. The size of the graphs is controlled using a pruning process. VeriWeb is able to deal with static pages, forms and client-side scripts. Figure 2.7(a) shows the VeriWeb method.

Haydar et al. (HPS04) [100] propose a method where an automaton is generated to
model observed run time behaviors of both static and dynamic pages with form filling (using the GET and POST methods). The authors call these observed behaviors *browsing sessions*. Frames and frameset behavior, multiple windows, and their concurrent behavior are also observed as portions of browsing sessions, called *local browsing sessions*. Those partitions are modeled as communicating automata to represent the concurrent interaction between local browsing sessions and to assist in reducing the state space of the underlying system.

The method is implemented using a framework that includes the following five steps: First, the user defines desired attributes using a graphical user interface prior to the analysis process. These attributes are used in formulating the formal properties to verify. Second, a monitoring tool intercepts HTTP requests and responses during the navigation of the Web Application Under Test (WAUT). Third, the intercepted data are fed to an analysis tool that either continuously analyzes the data in real time (online mode), or incrementally builds an internal data structure of the automata model of the browsing session, and translates it into XML-Promela. Fourth, The XML-Promela file is then imported into aSpin, an extension of the Spin model checker. Finally, the aSpin checker verifies the model against the properties, yielding counter-examples that facilitate error tracking. While outwardly extensive, this work lacks for completeness, as other dynamic features, user operations, and security properties are not captured. Figure 2.7(b) shows the Haydar’s et al. framework.

Rather than building a flat graph model like the VeriWeb model, or using communicating automata like Haydar’s model, FSMWEB (AOA05) [24] uses the idea of clustering related web pages into a logical web page. Hierarchies of finite state machines are then built for the resulting logical web pages. The FSMWeb model is able to capture many static and dynamic features, but is not able to cope with all the required features such as user interactions, and security properties. The logical web pages are currently generated by hand. The hierarchies of FSMs reduce the state space size an alternative to the graph pruning used by VeriWeb. The communicating automata in Haydar’s model represents another kind of reduction of
FIGURE 2.7: Modeling Web Applications Using a Single Model: the VeriWeb (BFG02) [32], Haydar et al. (HPS04) [100], and FSMWeb (AOA05) [24] methods.

the state space. Figure 2.7 shows all three methods.

Wu and Offutt (WO02) [173] present a modeling technique for web applications based on regular expressions. They model the behavior of web applications consisting only of dynamically generated pages for the purpose of functional testing. The technique identifies atomic elements, defined as a static HTML files or sections of a server programs that print HTML and have an all-or-nothing property (i.e., either the entire section is sent to clients or none of it is sent). An atomic element may be a constant HTML section, or it may be an HTML section that has a static structure but may contain variable content. These elements are dynamically combined to create composite web pages using sequence, selection,
aggregation, and regular expressions. This work is different from FSMWeb, Haydar et al. and VeriWeb in its ability to deal with user operations, and in its use of source code in the analysis.

**Multiple Model Methods.** Kung et al. developed their method, the Web Test Model (WTM) (KLH00) [117], based on multiple models of the applications under test. The models include Object Relation Diagrams (ORD), Object State Diagrams (OSD), a Function Cluster Diagram (FCD), and a Page Navigation Diagram (PND). The web application is represented using object relation diagrams (ORD) expressed in terms of objects (web pages and components) and their relationships. An \( ORD = (V, L, E) \) is a directed graph, where \( V \) is a set of nodes representing the objects, \( L \) is a set of labels representing the relationship types, and \( (E \subseteq V \times V \times L) \) is a set of edges representing the relations between the objects. There are three kinds of objects in WTM: client pages, server pages, and components. The relations navigation, request, response, and redirect are used to model navigation, HTTP request/response, and redirect respectively in web applications. Navigation behavior of the web application is represented using a page navigation diagram (PND), a finite state machine with states to represent client pages, and transitions between the states to represent hyperlinks. Object state diagrams (OSDs), which are similar to Statecharts, are used to describe the state behavior of interacting objects. To capture control and data flow information, a Block Branch Diagram (BBD, similar to a control flow graph) and a Function Cluster Diagram (FCD, a graph representation of dynamic function calls) are used.

The FSMWeb (AOA05) [24] and Haydar et al. (HPS04) [100] methods differ from Kung’s work in that those methods do not require source code to be available; their models are built depending on logical web pages rather than physical web pages, and they use an enhanced single FSM model instead of multiple models. Kung et al. differs from FSMWeb in that it can not deal with dynamically generated web pages, and from Haydar et al. in not handling the concurrent behavior of multiple windows.

Tonella and Ricca propose a two-layer model (TR04) [164]. The first layer is a UML
model of the web application for high level abstraction. This model is based entirely on static HTML links and does not incorporate any dynamic aspects of the software. The second layer is represented using a multicolored control flow graph (CFG) obtained by white-box analysis supported with information extracted from the access log of the server while the application is under executing. This work is different from FSMWeb and Haydar et al. in that it performs white-box analysis and uses multiple models. It also differs in not handling the concurrent behavior of frames and multiple windows.

Even though WTM, Tonella and Ricca, and Wu and Offutt’s (WO02) [173] methods all use a white-box approach in the analysis of the web application, the navigational model obtained by Tonella and Ricca is static, whereas in WTM and Wu and Offutt the model is dynamic. While WTM and Tonella and Ricca both try to model web applications using more than one model, the integration of the models and the validation of their interaction is not clearly described.

In contrast, Knapp and Zhang (KZ06) [111] propose a systematic approach to integrate a complete model for web applications from separate models. This is done using graph transformation rules on the UML-based web engineering meta-model [113] to generate a UML state machine that includes static navigation in addition to dynamic behavior. The final model can then be validated formally, though this model still lacks for checking of many dynamic, security and interaction properties.

Another integration method is proposed by Díaz et al. [74], the Ariadne Development Method, which is able to specify and evaluate hypermedia and web applications in a systematic, flexible, integrative and platform-independent way. The Ariadne Development Method provides a set of meta-models to specify information structure, navigation paths, interaction mechanisms, presentation features and access control policies.

Guerra et al. (GSDA07) [91] propose a verification framework dedicated to security policies in web design. Their approach is based on graph transformation, using a source model based on a strong design model, the Ariadne Development Method of Díaz et al. [74].
The Ariadne Development Method is able to capture many static and dynamic navigation behaviors, including security policies. In Guerra et al., the focus is on verifying properties related to access control policies. They generate an equivalent Petri net graph from the Ariadne design model using the triple graph transformation system (TGTS) [77]. This is composed of three graphs: the source graph, the target graph, and a correspondence graph that relates the elements in the source and the target graphs. Using this transformation, the authors are not only able to verify many static and dynamic properties, but also to relate the results of the analysis back to the original model.

To summarize, the approaches described here either use a single model for representing more than one level of the web application, or an integration of different models where each represents a single level. In the single model methods, to control the large state space caused by the complexity of web applications, the authors either use pruning in the graph-based models or clustering and communicating automata for FSM-based models, using the concept of logical web pages rather than physical web pages as a basis. For the integrated models, the authors use a variety of notations to represent the different levels, but mostly they use UML-based models to represent static navigation, and state-based models to represent dynamic behavior. The methods discussed here do not provide a clear description on how the integration is done, and none of them is able to model or check the desirable properties at all web application levels.

2.5 Proposed Methods That Do Not Fit Our Comparison Criteria.

Other work has been proposed to check the correctness of web application design specifications [63, 64], and yet others try to verify consistency between design specifications and the implementation of web applications [127].

Deutsch et al. propose a framework, WAVE, to help designers verify properties expressed
in temporal logic against web application specifications expressed in a rule-based textual format. The checking is done statically at design time [63, 64, 34]. The output is expressed by either true if the property is satisfied or false with counter examples if the property is not satisfied. The framework also is able to generate code based on the verified web application specifications. The set of properties that WAVE is able to verify is quite different from those that we reviewed in this survey. Besides being able to verify reachability properties like all other methods, WAVE focuses on checking the semantic properties of the business process underlying the web application such as, “the user cannot cancel an order that has already been shipped”.

Based on the verification engine provided by WAVE, Brambilla et al. [34] provide a front end taking advantage of Model Driven Architecture (MDA). Instead of writing the text-based specification to be fed into WAVE along with the properties to be checked at design time, Brambilla et al.’s framework enables web developers to verify models built by WebML, a high-level notation for data-, service-, and process-centric web applications, using a set of transformations to translate WebML models into WAVE specifications. Again, both frameworks are different from the methods that we are interested in, as they are focused on different kind of properties - business process properties. They could however be classified according to our taxonomy as navigational modeling methods.

Miao and Zeng [127] propose an approach to check the consistency between two models: the design model of the web application and its implementation model. They use Object Relation Diagram (ORD) proposed by Kung et al. [117] to build their design model. Unlike most of the methods that we reviewed, properties to be checked are derived from the design model rather than being specified in advance. The automatically generated properties along with the implementation model that is extracted manually from the code are fed to the SMV model checker. The properties to be checked are generated based on a consistency theory proposed by the authors. The consistency theory is mainly concerned with the coverage of all the nodes (any web page or web component) and relations (any navigational link between
the nodes) specified in the design model. It also checks that any other relation or node not specified in the design model is not covered. The authors also use the same approach to automatically generate a sequence of test cases based on the consistency between the design and the implementation models [176]. This approach could fit in our classification under navigational modeling methods. However, we choose not to include it because the set of properties to be checked is not specified in advance, even though they are mainly reachability properties.

The Choi and Watanabe [48] propose an approach to check consistency between different design models of web applications. They check consistency between the page flow diagram and the class model, which is composed of the object oriented-web application class specifications along with their methods. They also check consistency of the behavior of the designed web application by checking the consistency of the class model vs. the activity diagram. They generate a formal model by representing all the design models as labeled transition systems that are fed into the model checker UPPAAL[29]. Choi and Watanabe method is quite different from the methods that we reviewed in that they focus on consistency issues between the different design models rather than properties that other models are interested in verifying.

2.6 Models Proposed But Not Yet Used for Verification and Testing.

Dargham and Nasrawi [57] propose a new approach for modeling hypermedia web applications using an extended Finite State Machine (FSM). The authors first propose a classification for a web application’s pages and links to capture most of the web application behavior such as the static, dynamic, and interaction behaviors, then they represent the web application using an extended FSM by adding types to its states and transitions which map to their proposed classification. Their model can be used for testing and verification, because it is
based on a formal notation that has been used for this purpose in many previous methods.

In place of using a FSM, Qian et al. [154] use a labeled transition system (LTS) to model hyper-media web applications using a very similar classification to Dargham and Nasrawi. Then they extend the LTS to add types to its constructs that correspond to their classification. Again their model can be used for verification and testing, and is able to capture different behaviors of web applications such as static, dynamic and interaction behaviors. However, both models still lack the ability to capture properties related to security, sessions and cookies.

2.7 Conclusions and Open Problems

Little work has been done to compare different modeling methods used in web application validation. Even though there has been small-scale comparison, to the best of our knowledge this is the first study, other than our previous short summary [9], that provides a comprehensive review and comparative study of modeling methods that are currently applied in the field of web application verification and testing. All previous work has focused on the development process in general, and on the design phase in particular. Comprehensive reviews and comparative studies such as ours can help in highlighting the areas that need further research, and may help new researchers who are interested in the area to quickly get an idea of what has been done, and what could be done. This is especially so if the study is able to provide them with the strong and the weak points for each method, which may give them ideas on how to combine the strong points in a unified improved new modeling method.

2.7.1 The State of the Art

Our study shows two different views of the methods we surveyed, a general categorization by modeling level, and a detailed comparison by property coverage. Table 2.4 summarizes
the first one, where the 24 methods are categorized according to the level of web application modeling, as interaction behavior modeling methods, navigation modeling methods, content modeling methods and hybrid modeling methods (methods that model more than one level). In each category, methods are sorted according to the notation used by the method. At the same time, comparison between the methods was also done based on other criteria such as application for the method (analysis, testing, verification or some combination); whether the source code is required for the analysis or not; the way the method solves the state space explosion problem; and finally, whether there is tool support for the method. The second comparison, shown in Table 2.5, aims at a comparison of the more specific details between methods in the same category in particular, and with other methods in other categories in general. The comparison is based on a combination of feature type and the level of web application modeling, using the comparison criteria outlined in Section 2.2.4 as desirable properties for web site modeling. Based on our analysis in this review we want to highlight the following ideas and results.

First, in Section 2.4.1, we saw that interaction modeling approaches are able to model the interaction of web applications with the browser by proposing abstract models represented in different notations. All of the proposed methods are able to model the basic browser back and forward operations, and some are more mature, with the ability to model other browser features such as the history stack, page caches, and user sessions. The authors discuss the ability to integrate their interaction models with the static navigational model and try to do the integration manually, some try to detect web application-browser interaction bugs by implementing their own model checker or by using existing testing and model checking techniques. None of the models discuss integration with dynamic web applications, or how the dynamic features affect their interaction models.

For content modeling methods, the focus of the discussed methods is on verifying the static content of web applications. Up until now none has studied the verification of dynamic content for the same features of correctness and completeness. This kind of study will be
required to help with the increasing dynamism of web applications.

In navigation modeling methods, the authors use either UML-based models, Graph-based, Statechart-based models or SDL-based models to represent the navigation level of the web. While UML is the modeling standard for many applications including the web, it may not be the appropriate choice for testing and verification. In order for the UML-based models to apply the testing and verification techniques, the models should be translated into formal ones. The alternative choice is to use graph based models that can be directly tested or model checked.

All the proposed UML-based models are able to capture the static features of the navigation, and to represent the specific details of the web pages including the frame structure. In Graph-based models, nodes represent different modeling semantics in the different methods, from simple page in the flat model into page with frame structure. Nodes are used also to represent windows, links, actions, and in some methods nodes are categorized into classes to reflect secure resources.

For hybrid modeling methods, some researchers model the web application as a whole, taking into account all the modeling levels of the application, and then attempt to solve the problem of the state space explosion in some way. Other methods model web applications at more than one level by using separate models. Using separate models for the different levels of the web application help in reducing the complexity of the model as well as decreasing its state space size which will have its effect in the accuracy of the testing and the verification process. However, the integration between those models should be declared explicitly and carefully by the modeling methods, whereas most of the discussed methods fails to satisfy, and none is able to completely check or test web applications from all its modeling levels.

2.7.2 Challenges for the Future

Ideally, we are looking for a model that is able to capture all the desirable features of web applications at all modeling levels, as well as being able to validate the model using model
checking. To the best of our knowledge no such model yet exists, but perhaps it may be obtained by integrating some of the existing modeling techniques.

In addition, web applications have the property of low observability, due to the difficulty of tracking some outputs. Usually the output that is sent back to the user as HTML documents are being analyzed, but there are also other kinds of output, such as changing the state of the server or the database, and sending messages to other web applications and services. Up until now it appears that there is no research which can address this issue.

Finally, there is also a need for work on security modeling techniques that are able to deal with the complex, distributed structure of web applications, taking into account the concurrent access to web servers and the other resources that are attached to them.

In the remainder of this thesis, we present a new framework for addressing these issues. The framework provides a set of novel techniques for the analysis and modeling of web applications for the purpose of security verification and validation. It is largely language independent, and based on adaptable model recovery which can support a wide range of security analysis tasks.
Chapter 3

A Verification Framework for Access Control in Dynamic Web Applications

In the previous chapter we found that many methods propose static models and tools to check static properties of web applications, and some of them try to model and check dynamic features, but none of them is able to check or even model the access control features of web applications. To target this challenge we present in this chapter a security analysis framework for dynamic web applications. Our framework is aimed at testing the conformance of dynamic web applications with role-based access control security policies. A role-based access control (RBAC) security model is recovered from the dynamic web application using a combination of static and dynamic analysis techniques. A formal analysis is applied on the recovered model to check access control security polices specified by a security engineer, either with a correctness check, or with a counter example if any access control violation is encountered in the code.

This chapter is organized as follows: Section 3.1 motivates our work. Section 3.2 outlines
our framework. Section 3.3 presents our plans for evaluating it, and Section 3.4 concludes and summarizes the chapter.

3.1 Motivation

Many methods and tools have been proposed to check for attack vulnerabilities in web applications such as SQL injection and cross site scripting [102, 103], but none of them attempts to detect broken access control attacks, either by testing or by model checking. In our previous work [9, 13], and as shown in Chapter 2, we found that many methods propose static models and tools to check static properties of web applications, and some of them try to model and check dynamic features, but none of them is able to check or even model the access control features of web applications.

In general there is little work [107, 16, 58, 6] on UML-based security modeling. The focus of UMLsec [107] is on modeling security issues such as data confidentiality and integrity rather than access control. Basin et al. propose Model Driven Security (MDS) and their tool SecureUML [58] to integrate security models into system models. The authors first specify a secure modeling language for modeling access control requirements as a generalization for RBAC, after which, they embed this language within an extension of UML Class diagrams. The authors of authUML [16] take a step back and focus on analyzing access control requirements before proceeding to the design modeling to ensure consistent, conflict-free and complete requirements. The Ahn and Hu method [6] differs from the above approaches in using standard UML to represent the access control features of the security model. They provide a policy validation based on Object Constraint Language (OCL) and Role-based Constraints Language 2000(RCL2000) [7], and then translate the security model to enforcement code.

All of these are forward engineering approaches, while the real need is for a reverse engineering approach that is not only able to model access control policies, but also able
to check them in real applications. There is a critical need for an approach that is able to test or model check web applications to ensure that they are protected from broken access control attacks, and this is the goal of our work.

3.2 Research Approach

Our proposed framework (Figure 3.1) is aimed at recovering an RBAC security model from dynamic web applications. Based on a formal version of this model, the framework can be used to verify whether a dynamic web application conforms to the access control policies specified by a security engineer, either with a correctness check, or with a counter example if an access control violation is encountered in the code. The framework involves two main phases:

1. Static and dynamic reverse engineering of the web application structure and behavior.
2. Security model construction and analysis.

In the following subsections we will outline all of the framework components and the flow of data between them.

3.2.1 Web Application Reverse Engineering

In the first phase, static and dynamic analysis of the dynamic web application is used to recover the basic elements of a RBAC model [158]. We need to specify the set of users, roles, resources and their hierarchies, as well as the relations and access policies between them. Extracting static models such as class diagrams and behavioral models such as sequence diagrams helps us in this regard.
CHAPTER 3. A VERIF. FRAMEWORK FOR ACCESS CONTROL IN WEB APP

Static Analysis

The static analysis shown in Figure 3.1(B) extracts class diagrams that help in identifying the set of users, roles, resources and any relations between them. In Chapter 4, we present [8] an automated transformation from an SQL (DDL) schema to an open XMI 2.1 UML-adapted class model. The adapted model is a tailored UML class model to represent the basic ER diagram components, including entities, attributes, relations, and primary keys. Our transformation technique is a novel one in that it is open, non-vendor specific,
and targeted at the standard UML 2.1 exchange format, XMI 2.1. Although comparable commercial transformations exist, they are closed technologies targeted at formats tightly coupled to the vendor’s tools, hindering portability and preventing users from choosing their preferred tools in the development process. This analysis is supported by a dynamic analysis that may refine the class diagram, as well as recover behavioral models.

**Dynamic Analysis**

Static analysis is not adequate because it does not take into account the runtime behavior of web applications. Dynamic analysis is required to perform a full security analysis, including tracking user sessions, cookies, and user inputs. To recover the implicit permissions from dynamic web applications, we have proposed and implemented an approach and tool [39][14] to automatically instrument dynamic web applications using source transformation technology [55], and to recover a sequence diagram from execution traces generated by the resulting instrumentation, Figure 3.1(A).

Using an SQL database to store generated execution traces, our approach automatically filters traces to reduce redundant information that may complicate program understanding. The elements in the sequence diagram are the interactive user and browser session, the Application Server, and the application pages and entities. The messages between these elements represent page transitions and how they affect the application entities, either with read or write operations. While our current implementation supports all versions of the PHP scripting language, the framework is not tied to any particular language and can be easily adapted to other scripting languages. This aspect of the framework is discussed in Chapter 5.

Our proposed framework addresses code coverage by augmenting the dynamic analysis with instrumentation for code coverage, as presented in Chapter 6. This will decrease the percentage of false positives due to an analysis that results in a model that only partially covers the code (leading to verifications of properties that may in fact not hold).
CHAPTER 3. A VERIF. FRAMEWORK FOR ACCESS CONTROL IN WEB APP

Even using code and flow coverage methods, enumerating all execution paths is difficult. Ideally our framework should be able to identify all execution paths, but in some cases the human factor may be unavoidable, for instance when valid or critical information is needed in forms, user names or passwords. Like web security scanning tools such as VeriWeb [128] and AppScan [106], we have adopted a profile-based solution which requires administrators to manually supply valid values for form fields.

3.2.2 Model Construction and Analysis

In this phase a UML-based security model is constructed based on the Basin et al. [58] security meta-model (SecureUML). A transformation from this model to a state-based formal analysis model is then performed to ease the process of security analysis and verification.

RBAC-Model Construction

The core part of the proposed framework is the security model. In order to be able to check the web application’s access-control security properties, the framework must be based on a strong security model, and be able to extract it from the source code. We construct our security model using a Role-Based Access Control (RBAC) approach, Figure 3.1(C). Since users are not assigned permissions directly, but rather acquire them through their role (or roles), management of individual user rights is simplified. In a role-based model, permissions for common operations such as adding a user or changing a user’s department become obvious.

Our RBAC model is constructed by binding the recovered application ER model [8] with the recovered dynamic behavioral model (sequence diagram). The recovered sequence diagram is generated based on execution traces collected from the dynamic analysis part of our framework [39]. Web crawling tools that mimic user interactions with web applications, such as clicking links, filling in forms and pressing buttons [39, 169] are used to automate collecting traces, while the application roles themselves are recovered manually by studying
the software documentation. Roles can be identified from the HTTP session variable and by recovering the way the web application classifies users into roles. (Complete automation of this part is currently a work in progress). The generated sequence diagrams are combined with the application XMI 2.1 form of the ER model recovered by the static analysis part of our framework [8], using Model Driven Security (MDS) [58] to automatically generate a SecureUML model for the web application. Chapter 8 elaborates on this part in more detail.

Model Transformation and Formal Verification

Once the SecureUML model is constructed, we need to analyze it against the security properties (Figure 3.1(D)). While UML models provide good support for verifying web application requirements, they need to be converted into a formal state model in order to be automatically checked [9, 13]. Several methods in the literature propose tools for the translation from UML diagrams to formal state models that can be checked using existing formal verification tools. Examples are UML2Alloy [33] and XMI2SMV [40].

In Chapter 8, we convert our SecureUML model to a formal state model using a similar conversion process. The formal model along with the desired security properties is fed to a formal verification tool such as Alloy, yielding either confirmation that the properties hold, or a counter example. When a counter example is generated, the problem is mapped back to the code at the function point level by tracing back to the violated dynamic page. In some cases it may be possible to go deeper, for example using the parameters provided in the URL to identify the block of code causing the violation.

3.3 Evaluation and Preliminary Results

In our first experiment, we applied the proposed approach to the PhpBB [90] web application. PhpBB is the world’s leading open source forum software. It has a powerful
permission system and a number of other key features such as private messaging, search functions, a customizable template and language system, and support for multiple database technologies.

Each phase of our framework has been evaluated on a partial trace of the whole application, and results have been described along with approach discussion. A thorough experiment for the whole framework is also provided in Chapter 9.

3.4 Conclusion

The proposed approach is a novel one in web application security verification. Besides being the first approach to tackle the issue of access control verification, the proposed framework is flexible enough to allow for different server side technologies and databases.

Our approach also yields the potential for application in systems other than web applications. The static and dynamic reverse-engineering front-end of the framework can be reused for other kinds of analysis, and the framework could be used to discover other kinds of security attacks, such as cross-site scripting and SQL injection.

In our first experiment, the framework is evaluated on one of the most popular PHP web applications, PhpBB, to check that the application is free from any remaining access control vulnerabilities.
Chapter 4

Lightweight Transformation of Data Models from SQL Schemas to UML-ER

Data modeling is an essential part of the software development process, and together with application modeling forms the core of the model-driven approach to software engineering. While UML is considered the standard for application modeling, there is really no corresponding open standard for data modeling. In this chapter, we propose an approach and a tool to help bridge the gap between application and data modeling based on source transformation technology. The approach, SQL2XMI, was initially created to support our security analysis framework. Its role is to recover the application resources from an SQL schema and to represent them as a UML-ER model expressed in XML Meta Interchange (XMI) 2.1. On the other hand the approach we have used can be generalized to recover a rich UML-ER model from any SQL schemas to any XMI 2.x format, and can easily be extended to support code engineering by automatically generating SQL schemas from an XMI 2.x file.
This chapter is structured as follows. Section 4.1 motivates our work, and Section 4.2 places it in the context of related work. Section 4.3 introduces the SQL2XMI tool and briefly discusses its implementation using source transformations. Section 4.4 presents a small case study, using the tool to recover a UML ER diagram from the SQL DDL database schema of a popular open source production web application. Finally, Section 4.5 concludes the chapter and outlines possible directions for future work.

4.1 Motivation

Model-driven software development generally begins with either application modeling or data modeling. The two are closely related to, and complement, one another. In the application modeling domain, several Object Oriented modeling notations were combined in the early 1990s to produce the Unified Modeling Language (UML [138]). It has become the open standard notation for describing multiple aspects of the specification and design of large software systems. On the other hand, Entity-Relationship diagrams (ER diagrams [46]) are usually used for data modeling, and ER is the most commonly used method to build data models for relational databases.

However, ER modeling does not define a standard graphical syntax for the representation of ER diagrams, and there is no open standard for representing data objects in ER. Rather, each practical modeling methodology uses its own notation. The original notation used by Chen [46] is widely used in academic texts and journals, but is rarely seen in either CASE tools or publications by non-academics. Currently, a number of notations are used; among the more common are Bachman [26], crow’s foot [162], and IDEFIX [79]. This diversity in the underlying notations leads to data modeling that is tightly coupled to the specific data modeling tool of a specific vendor. As a result there is neither a generally accepted open standard nor interoperability of ER data models. From a model-driven engineering (MDE) perspective there is no general way to define transformations that either generate
or consume such data models.

As researchers try to find a standardized graphical representation for data modeling using ER diagrams, the obvious choice is to use UML. Using UML as a standard for ER data modeling could bring many benefits. First, because UML it is a widely accepted language used by analysts and software developers, it can be an excellent fit for graphical representation of ER diagrams. Second, UML’s generality can assist in the unification of all areas of expertise into a unified platform. It does not matter how different the technologies are, all of them can be described using the same language. This strength is brought to UML by profiles, a standardized set of extensions and constraints that tailors UML to particular uses [139]. Finally, recent empirical research comparing ER with UML class diagrams indicates that UML can significantly improve the comprehension level of programmers [61].

Through UML, development teams for application modeling and data modeling can more easily communicate. In addition they can gain the benefits of easy integration into repositories using meta-models, use of a standardized input/output format (XMI), and a unified software development process from analysis to implementation to deployment. In the long run, other benefits related to code engineering may be gained, for example by using the UML model to generate Data Definition Language (DDL) scripts, which can be executed on database systems to create the specified tables. In the shorter term, reverse engineering of independently created SQL database definitions (DDL scripts) to UML models can yield some benefits for reasoning about and maintaining deployed production systems.

Unfortunately, up until now there is no standardized UML data modeling profile. The need for such a profile has already forced some UML vendors and users to define their own UML profiles, but each has made their own interpretation and trade-offs, and all are UML 1.x based profiles.

This chapter is an extended version of our previous short paper [9] in which we proposed a transformation technique to bridge the gap between data modeling and application modeling using UML. An automated transformation from a SQL (DDL) schema to an open
CHAPTER 4. TRANSFORMATION FROM SQL SCHEMAS TO UML-ER

XMI 2.1 UML-adapted class model is presented. The adapted model is a tailored UML class model to represent the basic ER diagram components, including entities, attributes, relations, and primary keys. Our transformation technique is a novel one in that it is open, non-vendor specific, and targeted at the standard UML 2.1 exchange format, XMI 2.1. Although comparable commercial transformations exist, they are closed technologies targeted at formats tightly coupled to the vendor’s tools, hindering portability and preventing users from choosing their preferred tools in the development process.

While so far a prototype, our tool can recover a UML-based ER diagram from any SQL DDL schema and visualize it using any UML tool that supports the import of open standard XMI 2.1 exchange format. The tool can be easily adapted to be compatible with many different implementations of the SQL DDL notation, as well as different XMI 2.x versions. Using our method an open reverse transformation from XMI 2.x to an SQL DDL schema can also be easily implemented.

4.2 Previous Work

Several UML data modeling profiles (e.g., Ambler [21][22], Gorp [86], Gronik [85], and Silingas and Kaukenas [155]) have been proposed to answer the need for a UML data modeling profile to support the entire development process and help integrate with application modeling. In the absence of a formal UML data modeling profile, the proposed profiles represent a partial solution at best, since they support only UML 1.x, do not address the interoperability issue, and do not support the need for valid transformations to generate and consume these models.

Many commercial tools, such as Rational Data Architect (RDA) [104], Rational Software Architect (RSA) [105] and MagicDraw [136] provide their users with a transformation facility to import SQL DDL schemas to ER diagrams. However, the transformations used in these systems involve specific proprietary file formats that are tied to their own tools,
making portability difficult. For example, IBM tightly integrates RSA, which is used for application modeling, with RDA, used for data modeling. They have a transformation from an SQL DDL schema to an RDA logical data model (LDM) which can then be transformed into an RSA UML model and vice versa, but they do not provide a facility for direct transformation from an SQL DDL schema to an RSA UML model [44] and do not provide open interoperation. By contrast, our tool supports direct transformation from SQL DDL to the open standard XMI 2.1 portable interchange format for UML 2.1, thus supporting direct semantic model interchange between the range of different UML tools that are able to import the XMI 2.1 standard, including RSA. In addition, because our tool is open rather than proprietary, it can be quickly and easily adapted to support other XMI 2.x versions that may be used by other UML tools.

We use the TXL source transformation system [55] to implement our tool. This technology has previously been used by Abu-Hamdeh et al. [4] to reverse engineer SQL schemas to Prolog-style textual factbases. While Abu-Hamdeh et al.’s transformation recovers more information than our tool, for example entity types (weak, strong, etc.), cardinalities and some constraints, our target representation is the XMI 2.1 model interchange standard, which can be easily imported, visualized and manipulated by UML-based tools. The additional information recovered by Abu-Hamdeh et al. can be handled by our tool in the future using additional transformation rules, but is not required in our current program comprehension application.

Another related system is Chung and Hartford’s XMI2SQL [49], which can transform an XMI file exported by a UML tool to an SQL implementation. An ER model based on the Ambler profile [21] is built in UML first, and then exported to XMI by the UML tool. XMI2SQL then transforms this Ambler-based model in XMI to an SQL DDL schema. XMI2SQL is implemented as a web service in C#. This tool complements our work on the forward engineering and code generation side. However, its adaptation to cope with different variations of XMI files and SQL schemas could be challenging, since it is implemented as a
traditional custom C# program. Our use of source transformation rules makes it easy to rapidly adapt to variations.

There is a long history of reverse engineering of ER diagrams from databases, such as Premerlani and Blaha [145]. These approaches are more mature and handle more features by utilizing more input artifacts. Di Lucca et al. [70], Yang et al. [174] and Canfora et al. [38] all recover data models from the source code of data intensive applications. The purpose of our work is different, providing a lightweight translation of SQL DDL schemas to standard UML, a problem for which the OMG called for proposals in 2005 [137].

4.3 SQL2XMI

SQL2XMI is a process for automatically transforming an SQL DDL database schema to a UML ER diagram in XMI 2.1 model exchange format. In this first implementation, we have targeted MySQL, although extending to any other SQL variants is straightforward.

The SQL grammar used in our project is tuned for MySQL version 3.23.44. MySQL supports multiple storage engines. The InnoDB engine supports the checking of foreign key constraints and enforcement of referential integrity. This support requires the explicit use of foreign key and reference attributes during table creation. Other engines ignore these attributes. When provided, SQL2XMI uses these attributes to recover relations between entities. Without the attributes, SQL2XMI recovers relations based on naming conventions. Identifying foreign keys that are not explicitly defined is a problem that has been researched intensively in the past [145]. However, these approaches (e.g. [70, 174, 38]) require analysis of source code, which is difficult in dynamic dispatch systems such as PHP. Naming convention has proved sufficient for our purposes.

Our present prototype reverse engineers all the basic elements of the ER diagram, that is, the set of entities and their attributes, the primary key set, the foreign key set, and the relationships between them. In this initial version we do not infer entity types or
cardinalities, although they can be easily inferred if required. Our initial application is the complex software comprehension of web applications for security analysis, and in this application entity types and cardinalities are not a concern.

### 4.3.1 Implementation

Our implementation is based on source transformation technology, in which the target program, an XMI file, is viewed as a syntactic modification of the source program, the SQL DDL schema. This modification involves adding the target language features as extensions to the grammar of the source language, and then transforming each input language grammatical form to corresponding target language forms. For this purpose we use TXL [55], a programming language designed for manipulating and experimenting with programming language notations and features. TXL has been used in many production applications with transformations involving billions of lines of source code.

The TXL transformation process normally consists of three parts: a context-free “base” grammar for the source language to be manipulated, a set of context-free grammatical “overrides” (extensions or changes) to the base grammar, and a rooted set of source transformation rules to implement transformation of the extensions to the base language [55]. The TXL processor parses the source program and converts it into a parse tree, then recursively applies the set of transformation rules, beginning with a “main” rule, until there is no match encountered in the parse tree. TXL finalizes the process by unparsing the transformed parse tree into the target program. Figure 4.1 shows the TXL transformation process as applied in our tool. In our application we began by defining a grammar for the input MySQL DDL, and a second (override) grammar for our target XMI output forms. The main grammatical form (called [program] in TXL) allows both, but expects the XMI part to be empty on input and full on output, and vice-versa for the SQL part, Figure 4.2. We used the TXL producer-consumer translation paradigm to make a set of transformation rules to “produce” the XMI output while “consuming” the SQL input. The transformation
if the Col is PK in T1, then check the occurrence of this column in all other tables (T2..Tn). If a match happened with a column in Ti, it will add an XMI ownedAttribute element of type uml:Property to refer to a relation of type Ti between the two tables.

e. The function GenerateERDElements then identify table relations based on foreign keys (relation target) by applying the function constructRelations on each table Ti, which will check for each non primary key column the occurrence of this column in the set of PK_T list that is collected in step a, if there is a match with PKj_Ti, add an XMI ownedAttribute element of type uml:Property to refer to a relation of type Ti between the two tables.

f. The function GenerateERDElements then creates the relations links and for each relation between table Ti and Tj, detected in step d and e, create an XMI packagedElement of type uml:Association, this element should be generated once for each PK_Ti set and FK_Tj set.

g. The function GenerateERDElements finally concatenate the results of steps 1, 2, 3 and 4.

TXL is a programming language specifically designed for manipulating and experimenting with programming language notations and features using source to source transformation. The motivating paradigm of TXL consists of beginning with a grammar for an existing language, specifying syntactic modifications to the grammar representing new language features or extensions, and rapidly prototyping these new features by source transformation to the original language.\[13\]

TXL has grown into a powerful general purpose source transformation programming system. It has been used in a wide range of applications, including industrial transformations involving billions of lines of source code.

TXL programs (Fig. 1) normally consist of three parts, a context-free “base” grammar for the language to be manipulated, a set of context-free grammatical “overrides” (extensions or changes) to the base grammar, and a rooted set of source transformation rules to implement transformation of the extensions to the base language.

Figure 4.1: The TXL Transformation Technique as Applied in our Tool

```
% Base grammar for SQL
include "MySQLGrammar.grm"

% Output override grammar for XMI
include "XMI.grm"

% Allow for both input and output forms
define program
  [repeat MySQLStatement]
  [repeat XMItoken]
end define

% Main transformation rule
function main
    XMIFile % Initially empty
    [createXMIIH] % Make XMI header
    [GenerateERDElements MySQLS each MySQLS]
    % Generate XMI elements
    [createXMIE] % Make XMI trailer
end function
```

Figure 4.2: Grammatical specification and main transformation rule (function)

is started by the main rule, shown in Figure 4.2. TXL begins by parsing the MySQL input schema into its basic DDL statements (pattern variable MySQLs), with an initially empty XMI output (pattern variable XMIFile). The main rule replaces this entire input by constructing the XMI output beginning with the empty XMIFile and transforming it into the XMI representation of the input MySQLs using three transformation subrules, createXMIIH, GenerateERDElements, and createXMIE.

The functions createXMIIH and createXMIE simply create the XMI header and trailer elements which bracket the body of the translation. The function GenerateERDElements, shown in Figure 4.3, is responsible for the bulk of the transformation, transforming each
% Generate the UML ER diagram XMI element for each table column

\begin{verbatim}
function GenerateERDElements
  AllTableStructures [repeat MySQLStatement]
  TableStructure [MySQLStatement]

deconstruct TableStructure
  CREATE TABLE TN [id] { ColList [list createDefinition] };

construct Tname [stringlit]
  _ [quote TN]

construct SetOfAllPK [repeat XMItoken]
  _ [findAllPKL each AllTableStructures]

construct SetOfFK [list index_col_name]
  _ [findFKL each ColList]

construct eAnnotationsID [stringlit]
  _ [quote TN ] [EAnnotation]

construct detailsID [stringlit]
  _ [quote TN ] [Entity]

construct XMI_ERD_Entity [repeat XMItoken]
  <packagedElement
    xml:tipo="uml:Class" xml:id=Tname name>Tname>
    <xmi:Extension
      extender="http://www.eclipse.org/emf/2002/Ecore">
      <eAnnotations xmi:tipo="ecore:EAnnotation"
        xml:id= eAnnotationsID
        source="http://www.eclipse.org/uml2/2.0.0/UML">
        <details xmi:tipo="ecore:EStringToStringMapEntry"
          xml:id= detailsID key="Entity"/>
      </eAnnotations>
    </xmi:Extension>

    construct packagedElementCloseT [repeat XMItoken]
  </packagedElement>

    construct XMI_ERD [repeat XMItoken]
      XMI_ERD_Entity
      [createEntityAttrib Tname SetOfAllPK each ColList]
      [createFKside_Relation TableStructure each SetOfAllPK]
      [constructRelations TN AllTableStructures each SetOfFK]
      [. packagedElementCloseT]
      [constructAss TN SetOfPK each AllTableStructures]

    replace * [repeat XMItoken]
      % tail of output
      by
      XMI_ERD

end function
\end{verbatim}

Figure 4.3: The GenerateERDElements function, which transforms each SQL table to its
representation in XMI
a. Scan the MySql schema
b. Collect the set of all PK_T (Primary key, table name), based on the explicit primary key constraint statement in each table definition.
c. For each table create the correspondance XMI packagedElement of type uml:Class element that is annoted to be of Entity type, using stereotyping.
d. For each table column generate an XMI ownedAttribute of type uml:Property, and map the MySql data type for that column into the most approperiat one in the predefined types in the class diagram.
e. Identify table primary keys: Check if the table column (Col) and its table (T_name) are located in the set of all PK_T, which is collected in step b, if so add an XMI eAnnotations element for the XMI attribute element that is corresponds to the mached table column.
f. Identify table relations based on foreign keys (relation source): If the Col is PK in T1, then check the occurrence of this column in all other tables (T2..Tn), If a match happens with a column in Ti , add an XMI ownedAttribute element of type uml:Property to refer to a relation of type Ti  between the two tables.
g. Identify table relations based on foreign keys (relation target): For each non primary key column check the occurrence of this column in the set of PK_T list that is collected in step b, if there is a mach with PKj_Ti, add an XMI ownedAttribute element of type uml:Property to refer to a relation of type Ti  between the two tables.
h. (Create relation link): For each relation between table Ti and Tj , detected in step I and g, creat an XMI packagedElement of type uml:Association, this element should be generated once for each PK_Ti set and FK_Tj set.

<table>
<thead>
<tr>
<th>Table</th>
<th>&lt;&lt;Entity&gt;&gt;</th>
<th>packagedElement</th>
<th>uml:Class</th>
<th>Table name</th>
<th>Table name</th>
<th>Element eAnnotations</th>
<th>XMI Name</th>
<th>XMI Extension or additional attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>Attribute</td>
<td>ownedAttribute</td>
<td>uml:Property</td>
<td>Attribute Name</td>
<td>Attribute Name</td>
<td>Additional attributes</td>
<td>visibility</td>
<td>private</td>
</tr>
<tr>
<td>Primary Key</td>
<td>Attribute With extension (stereotype annotation) &lt;&lt;PK&gt;&gt;</td>
<td>ownedAttribute</td>
<td>uml:Property</td>
<td>Attribute Name</td>
<td>Attribute Name</td>
<td>Additional attributes</td>
<td>visibility</td>
<td>private</td>
</tr>
<tr>
<td>Foreign Key</td>
<td>Attribute</td>
<td>ownedAttribute</td>
<td>uml:Property</td>
<td>Source Table Name Target Table Name(PK)</td>
<td>Target Table Name</td>
<td>Additional attributes</td>
<td>visibility</td>
<td>private</td>
</tr>
<tr>
<td>Association</td>
<td>Relation</td>
<td>packagedElement</td>
<td>uml:Association</td>
<td>AssociationID</td>
<td>NULL</td>
<td>Additional attributes</td>
<td>memberEnd</td>
<td>String type containing the id of entities member in this relation</td>
</tr>
</tbody>
</table>

Table 4.1: Mappings between MySQL schema elements, ERD elements, and XMI 2.1 elements
SQL table definition, including all of its attributes, relations, and primary keys, to the corresponding XMI 2.1 elements. The correspondence between SQL DDL schema elements and XMI 2.1 elements (i.e., UML 2.1 ER diagram elements) implemented by the transformation is shown in Table 4.1. The GenerateERDElements function uses pattern matching to recognize each of the DDL table elements and map them to the corresponding XMI output.

4.3.2 Transforming Table Definitions to Entities, Attributes and Relationships

For the transformation of each SQL table in the input, the GenerateERDElements function is passed both the particular table to be transformed (each MySQLs) and the entire set of all of the input tables (MySQLs). This is because SQL tables do not have a simple one-to-one relationship with the corresponding UML 2.1 ER model - rather, the transformation of each table definition depends on information that can only be derived from other tables to which it is related, for example the set of all primary keys in the schema. Thus it is a context-dependent source transformation.

The function begins by deconstructing the create table statement to break it into its syntactic parts so we can access and manipulate them separately. By doing so, we are able to extract the table name to map to the entity name, the table columns to map to the entity attributes, and to identify the table’s primary keys and relations shared between the entities. Our implementation involves 16 separate transformation rules. Here we discuss three of the critical transformations in detail, and briefly outline the overall transformation process. Other rules in our transformation are similar.

The TXL constructor SetOfAllPK collects a list of all (primary key, table name) pairs based on the explicit primary key constraint statements in all of the table definitions in the database. This information is required later in the rule for inferring entity relationships for the table being processed. The constructor SetOfPK then collects another list of all the primary keys defined in the particular table being processed.
% Generate the UML ER diagram XMI element
% for each table column

function createEntityAttrib STableName [stringlit] PKL [repeat XMItoken] Colm [createDefinition]
  deconstruct Colm
  ColName [colm_name] ColDef [col_def]
  deconstruct * [dataType] ColDef
  DT [dataType]
  % isPKAttrib checks if the column is primary key
  % and annotates it with PK stereotype if so
  construct ModefidColName [repeat XMItoken]
  _ [isPKAttrib STableName ColName each PKL]
  construct AttribName [stringlit]
  _ [quote ColName]
  construct AttribDT [stringlit]
  _ [quote DT]
  construct AttribDef [repeat XMItoken]
  <ownedAttribute xmi:type="uml:Property"
  xmi:id= AttribName name= AttribName
  visibility="private">
  construct Closingtag [repeat XMItoken]
  </ownedAttribute>
  replace * [repeat XMItoken]
  % tail of output
  by AttribDef [.ModefidColName] [.Closingtag]
end function

Figure 4.4: The createEntityAttribute function, which translates SQL table columns to attributes in XMI

The constructor XMI_ERD_Entity creates the XMI representation of the entity for the table, consisting of an XMI packagedElement element of type uml:Class that is annotated to be an Entity using stereotyping. The constructor XMI_ERD then uses a number of subrules to flesh out this initial ER representation of the table by adding the translation of attributes, foreign keys and relationships to yield the entire translation of the SQL table definition.

4.3.3 Transforming Table Columns to Attributes

For each table column, the GenerateERDElements function uses the transformation subrule createEntityAttribute, shown in Figure 5.3. This transformation function generates an XMI ownedAttribute of type uml:Property to represent the table column. It begins by using
% Annotate XMI attribute element with PK stereotype
% if primary key

function IsPKAttrib STableName [stringlit]
    ColN [colm_name] SetOfAllPK [XMItoken]
    construct ColN_string [stringlit]
        _ [ quote ColN]
    % Is this column of this table in the primary keys?
    deconstruct SetOfAllPK
        <ownedAttribute _ [opt xmi_colon]
            _ [id_or_key] = STableName]
            _ [id_or_key] = ColN_string >
        replace * [repeat XMItoken]
    % tail of output
    by
        <xmi:Extension
            extender="http://www.eclipse.org/emf/2002/Ecore">
            <eAnnotations xmi:type="ecore:EAnnotation" xmi:id="_Ovi-
                uPdEdy8F" source="http://www.eclipse.org/uml2/2.0.0/UML">
                <details
                    xmi:type="ecore:EStringToStringMapEntry"
                    xmi:id="_Ovi-V- 8rGg0Zw" key="PK"/>
            </eAnnotations>
        </xmi:Extension>
    end function

Figure 4.5: The IsPKAttrib function, which identifies and stereotypes the XMI representation of columns which are primary keys

TXL deconstructors pattern to capture the column’s name (ColName), definition (ColDef), and data type (DT). The TXL constructor uses the transformation subrule IsPKAttrib to determine if the column is a primary key of the table and to annotate it with the XMI primary key stereotype if so. The remainder of the function creates the ownedAttribute representation of the column in XMI, embeds the primary key stereotyping and appends the result to the XMI entity representation of the table.

The transformation subrule IsPKAttrib, shown in Figure 4.5, identifies table primary keys by checking if the table column ColN and its table STableName are present in the set of all primary keys that was collected in GenerateERDElements and passed as parameter to its subrules. If so, an XMI primary key stereotype eAnnotation element that corresponds to the matched table column is returned by the function (otherwise it returns no annotation). The annotation is added to the XMI representation of the column by GenerateERDElements in its translation.
4.3.4 Translating Foreign Keys and Relations to Relationships and Associations

When foreign key and references attributes exist in the schema, they can be used by SQL2XMI to identify relations. When absent, SQL2XMI uses naming convention and column data type to infer foreign key relations. This can be augmented in the future with stemming operations to enhance naming conventions. Otherwise, foreign key inference is a difficult problem that requires additional data and code analysis such as the work by Di Lucca et al. [70], Yang et al. [174] and Canfora et al. [38], which require a completed running application and is outside the scope of our work. The following are the steps used to recover relations between tables based on naming convention and the data type similarity, as the other case is straightforward:

First, following the transformation of table columns, the GenerateERDElements function uses similar transformation subrules to handle foreign keys and relations. The subrule constructFKside_Relation identifies foreign keys as relation target by checking for an occurrence of each KEY column in the set of all primary keys \(\text{SetOfAllPK}\), which is passed as a parameter to the subrule. For each primary key of another table \(T_i\) that matches, an XMI ownedAttribute element of type \(\text{uml:Property}\) to refer to a relation of type \(T_i\) between the two tables is generated, where \(T_i\) is the name of the other table.

Next, GenerateERDElements uses the transformation subrule construct Relations to identify foreign keys as relation source by checking for columns that are primary keys in this table and also occur as a KEY column in another table. For each such foreign key, an XMI ownedAttribute element of type \(\text{uml:Property}\) is generated to refer to a relation of type \(T_i\) between the two tables.

Finally, GenerateERDElements uses the transformation subrule constructAss to create an XMI packageElement of type \(\text{uml:Association}\) between the two tables involved in each relation generated by the previous two subrules.
CREATE TABLE phpbb_forums ( 
    forum_id smallint(5) unsigned not null, 
    cat_id mediumint(8) unsigned not null, 
    forum_name varchar(150), 
    forum_desc text, 
    forum_status tinyint(4) default 0 not null, 
    forum_order mediumint(8) unsigned default 1 not null, 
    forum_posts mediumint(8) unsigned not null, 
    forum_topics mediumint(8) unsigned default 0 not null, 
    forum_last_post_id mediumint(8) unsigned default 0 not null, 
    prune_next int(1), 
    prune_enable tinyint(1) default 0 not null, 
    auth_view tinyint(2) default 0 not null, 
    auth_read tinyint(2) default 0 not null, 
    auth_reply tinyint(2) default 0 not null, 
    auth_edit tinyint(2) default 0 not null, 
    auth_delete tinyint(2) default 0 not null, 
    auth_sticky tinyint(2) default 0 not null, 
    auth_pollcreate tinyint(2) default 0 not null, 
    auth_reply tinyint(2) default 0 not null, 
    PRIMARY KEY (forum_id) );

CREATE TABLE phpbb_posts ( 
    post_id mediumint(8) unsigned auto_increment, 
    post_subject char(60), 
    bbcode_uid char(10) default '' not null, 
    user_id mediumint(8) unsigned not null default 0, 
    user_name varchar(25), 
    user_email varchar(255), 
    user_pass varchar(32) not null, 
    user_regdate int(11) default 0, 
    user_posts mediumint(8) unsigned not null, 
    user_status tinyint(2) default 0 not null, 
    PRIMARY KEY (post_id) );

CREATE TABLE phpbb_groups ( 
    group_id mediumint(8) not null auto_increment, 
    group_name varchar(100), 
    group_description varchar(255), 
    group_order mediumint(8) unsigned not null, 
    group_type tinyint(4) default 1 not null, 
    PRIMARY KEY (group_id) );

CREATE TABLE phpbb_users ( 
    user_id mediumint(8) not null, 
    user_name varchar(25), 
    user_email varchar(255), 
    user_pass varchar(32) not null, 
    user_regdate int(11) default 0, 
    user_posts mediumint(8) unsigned not null, 
    user_status tinyint(4) default 1 not null, 
    user_allow_viewonline tinyint(1) default 1 not null, 
    user_allow_pm tinyint(1) default 1 not null, 
    user_allow_bbcode tinyint(1) default 1 not null, 
    user_allow_html tinyint(1) default 1 not null, 
    user_allow_viewemail tinyint(1) default 1 not null, 
    user_allow_viewonline tinyint(1) default 1 not null, 
    user_allow_bbcode tinyint(1) default 1 not null, 
    user_allow_html tinyint(1) default 1 not null, 
    user_allow_viewemail tinyint(1) default 1 not null, 
    user_allow_viewonline tinyint(1) default 1 not null, 
    PRIMARY KEY (user_id) );

CREATE TABLE phpbb_categories ( 
    cat_id mediumint(8) unsigned not null auto_increment, 
    cat_title varchar(100), 
    PRIMARY KEY (cat_id) );

CREATE TABLE phpbb_forums ( 
    forum_id smallint(5) unsigned not null auto_increment, 
    forum_name varchar(150), 
    forum_desc text, 
    forum_status tinyint(4) default 0 not null, 
    forum_order mediumint(8) unsigned default 1 not null, 
    forum_posts mediumint(8) unsigned not null, 
    forum_topics mediumint(8) unsigned default 0 not null, 
    forum_last_post_id mediumint(8) unsigned default 0 not null, 
    PRIMARY KEY (forum_id) );
The concatenation of the results of the entire set of transformation subrules of GenerateERDElements forms the complete result transformation of the column to XMI, and the concatenation of the transformations of the whole set of columns forms the result of the main transformation function, yielding the complete XMI 2.1 representation of the UML 2.1 ER diagram for the original SQL schema.

4.4 An Example: PhpBB

SQL2XMI was originally designed to serve an ongoing project in web application security analysis, in which reverse engineering based on both static and dynamic analysis is being used to identify the application resources, permissions, and subjects that constitute the basic elements of a security system. Data models constitute one of the main sources of such information, and visualizing data models facilitates the process of understanding the structure of the system, its basic entities and their relationships. The output of the understanding phase is mainly represented using UML models, and we are using the XMI 2.1 model interchange notation to help unify the results of the understanding phases of our project.

In this context, we have evaluated SQL2XMI on the popular web bulletin board system PhpBB versions 2.0 and 3.0 [90]. Using the source transformation system described in Section 4.3.1, SQL2XMI has been able to automatically recover ER diagrams for the data models of both versions. The result has helped us both to understand the complex data model of this system and to recognize that the data model of PhpBB 3.0 has been completely restructured from the previous version.

PhpBB 2.0 has 30 tables that generate a 30 page XMI file representing a UML2 ER diagram much too large to fit in this chapter. So as an example we show only a part of the schema including ten tables in Figure 4.6, one page of the generated XMI file, showing one entity and all of its relations in Figure 4.7, and a snapshot of part of the visualization
Figure 4.7: Sample of the generated XMI 2.1 file
Figure 4.8: RSA visualization for a part of the generated diagram
of the ER diagram for the ten tables in Figure 4.8. We used Rational Software Architect (RSA) [105] to import and visualize the UML 2.1 ER diagram represented by the XMI 2.1 file automatically generated by our tool.

To get a better feeling for the process, we can look at a table from the schema presented in Figure 4.6 and see how it is mapped into the elements of the XMI file in Figure 4.7, and on to the ER diagram in Figure 4.8.

The table `phpbb_groups`, which is the last table defined in the schema of Figure 4.6(A) has one primary key, `group_id`, and six columns. The table does not define a foreign key statement, so it hard to tell from a first look at the schema which other tables are involved in relations with this table. In Figure 4.7, we can see that, based on the mappings defined on Table 4.1, the `phpbb_groups` table has been mapped into a UML `ownedAttribute` element of type "uml:Class" and annotated as an "Entity" using the XMI tag "eAnnotation".

The primary key, `group_id`, has been mapped to an XMI "ownedAttribute" element with type `uml:property`, and annotated with the XMI "eAnnotation" element to be of type "PK". Each of the other five columns is mapped to an XMI "ownedAttribute" element of type "uml:property", but without any annotation. In the lower middle part of Figure 4.8 we can see the visualization of the XMI element and how it is represented as << Entity >>, with the primary key represented as << PK >> `group_id`, and the other five attributes listed without any annotation.

We can recognize easily in Figure 4.8 that the entity `phpbb_groups`, marked with (A), is involved in two relations based on its primary key, one with `phpbb_auth_access` entity, marked with (B), while the other with `phpbb_user_group`, marked with (C). This fact is difficult to see in the original schema because the foreign key constraint is not defined explicitly. Relations for the table have been recovered as described in section II, and constructed in the XMI file as a `packageElement` of type "uml:Association" whose `memberEnds` are the entities evolved in the relation. We can recognize two of these in the XMI file, reflecting the two relations that the `phpbb_groups` entity is involved in. XMI `ownedAttribute` elements of
XMI type "uml:Property" has been recovered for both entities that share a relation, such that each one refers to the other in the type attribute of the element, and both of them refer to the same association element that joins them.

Even in this first simple example we can see that important information such as relations can often not be easily understood directly from the SQL schema. The process of comprehending the application at this level can be tedious work, especially for more complicated and larger schemas. Similarly, even after automatic transformation to ER form, the XMI file structure is itself much too complicated to follow. Visualizing the XMI file as an ER diagram as shown in Figure 4.8, however, presents the database schema in format that can be easily and quickly understood by all members of the development team.

4.5 Conclusions and Future Work

In this chapter we have presented a source transformation technique to bridge the gap between data modeling and application modeling that can assist in the process of complex software comprehension and evolution. Our new open tool, SQL2XMI, automatically transforms an SQL DDL schema to a UML 2.1 ER diagram which can be visualized by any UML tool that supports XMI 2.1. Unlike other tools that reverse engineer to proprietary formats, SQL2XMI explicitly aims at open and flexible portability, requiring only the SQL DDL schema and targeting the official OMG XMI 2.1 UML representation. We have presented the details of our lightweight source transformation-based approach and an example of the application of our tool to recover an ER diagram for the popular internet bulletin board system PhpBB. The approach and mapping are unique to our work.
Chapter 5

Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications

In Chapter 4, we used static analysis to recover application entities and the relations between them as a UML-based ER model. In this chapter, we use dynamic analysis, implemented via PHP2XMI, to recover the permissions associated with each user role and express them as a behavioral model represented by a UML sequence diagram.

This chapter presents an approach and tool to automatically instrument dynamic web applications using source transformation technology, and to reverse engineer a UML 2.1 sequence diagram from the execution traces generated by the resulting instrumentation. The result can be directly imported and visualized in a UML toolset such as Rational Software Architect. Our approach dynamically filters traces to reduce redundant information that may complicate program understanding. While our current implementation works on PHP-based applications, the framework is easily extended to other scripting languages in plug-and-play fashion. In addition to supporting web application understanding, our tool is
being used to recover traces from dynamic web applications in support of web application security analysis and testing. We demonstrate our method on the analysis of the popular internet bulletin board system PhpBB 2.0.

The rest of this chapter is structured as follows. Section 5.1 motivates our work. Section 5.2 presents the details of our approach, and Section 5.3 presents an example that demonstrates our method on a real system. Section 5.4 relates our efforts to previous work. Finally, Section 5.5 outlines our conclusions and plans for future work.

5.1 Motivation

Program comprehension, analysis and evolution is often based on reverse engineering of the structure and behavior of software to visual models such as UML diagrams, and much recent research has been focussed on recovering and presenting the structure of programs as UML class diagrams. However, the recovery of dynamic behavior, and particularly interaction behavior, to models such as sequence diagrams presents many challenges that have yet to be addressed.

The problem of recovering execution traces to sequence diagrams for object oriented systems, written in languages such as C++ or Java, has already been extensively studied. Hamou-Lhdaj and Lethbridge [95], and Briand et al. [35] provide surveys of tools that have been applied in the domain of object oriented languages that deal with interaction behaviors, and Merdes and Dorsch [126] have presented the major challenges in building a scalable and efficient tool to understand the interaction behavior of such software. These surveys raise four main issues: First, how do methods model the execution traces? Second, how do they solve the execution trace explosion problem? Third, is the method able to represent technical details such as loops and conditions in the sequence diagrams? And fourth, how do the methods represent the final diagram for visualization purposes?

Reverse engineering of sequence diagrams from web applications implemented using scripting languages such as PHP faces all of these problems, and presents a number of
additional challenges that are not addressed by these object-oriented analysis methods:

- Identification of the interaction elements. In general, web applications are not built based on object-oriented concepts, so it can be difficult to identify the application entities in the source code.
- Identification of loops and conditions. Web applications often have multiple entry points, and exhibit behavior that is difficult to detect until run time. This behavior usually depends on user inputs that can not be inferred by static analysis.
- Recognition of similar execution trace patterns from static or run time information.
- Representation of the complete set of behavioral changes from state to state in sequence diagram terminology. Web applications often have several components that may be affected by a single page execution, such as the database and session and cookie variables.
- Analysis of multilingual documents.

In this work we present an approach and a tool that faces these additional challenges, automatically generating interaction sequence diagrams from dynamic web applications. Our method is not a perfect solution for all of these issues, but is an improvement to the extent that it delivers accurate results and supports the process of web application comprehension, analysis and evolution.

5.1.1 Web Application Testing

PHP2XMI is an essential part of a framework aimed at testing the conformance of dynamic web applications with role-based access control security policies. A role-based access control (RBAC) security model is recovered from the dynamic web application using a combination of static and dynamic analysis techniques. The static analysis is used to recover application entities and the relations between them as a UML-based ER model [8]. The dynamic analysis, implemented using PHP2XMI, recovers the permissions associated with each user role and expresses them as a behavioral model expressed as a UML sequence diagram.
Visualizing execution traces as a sequence diagram facilitates the process of understanding the interaction behavior of the system, and helps us deduce the permissions for each user role. The XMI 2.1 textual representation of the sequence model is analyzed and combined with the XMI 2.1 representation of the ER model to construct a UML-based RBAC model, which can be converted into a formal model to be checked for access control vulnerabilities using a standard model checker.

This chapter explains how PHP2XMI is used to recover role permissions at the level of page access, and we are currently also evaluating it at the entity level. The behavioral model recovery technique implemented in PHP2XMI can be used to test for other web application security vulnerabilities, such as SQL injection, by tracking SQL sources and their relation to user inputs captured as HTTP variables. For security analysis we want to preserve all the possible paths the user may follow to reach target pages. Hence, the model behind the execution traces is a complete graph. A filtering process is used to ensure that each observed path is stored just once in the database. While our current version of the tool does not model loops explicitly, our filtering does not prevent the handling of cycles and they can be easily detected by analyzing the database and representing them using the appropriate UML 2.1 meta-model elements.

5.2 PHP2XMI

PHP2XMI is a new reverse engineering tool aimed at recovering UML 2.1 sequence diagrams from PHP-based dynamic web applications. The approach used in PHP2XMI involves three steps, as shown in Figure 5.1:

1. Parsing and Dynamic Instrumentation: The core of our method, which automatically inserts probes into the source code to collect dynamic information such as page URLs, http variables, sessions and cookies.
2. Filtering and Storing: During interactive browser sessions, execution traces generated
Dynamic PHP Web Application

Instrumented PHP Application

Web crawling Tool

Instrumenting Function Library

Browser Session

SQL Trace Database

Trace to XMI Transformation

XMI2.1 Sequence Model

Rational Software Architect

UML 2.1 Sequence Diagram

Figure 5.1: PHP2XMI tool Architecture

by the probes are filtered to ignore redundant information and stored in an SQL database for further analysis.

3. Database Analysis and Model Generation: The execution traces stored in the database are transformed into UML2.1 sequence meta-model elements.

In the following subsections we present the details of each of these main steps.

5.2.1 Parsing and Dynamic Instrumentation

Static analysis alone is not sufficient for architectural recovery of heterogeneous and highly dynamic software such as web applications, and therefore it must be complemented by dynamic analysis [153]. Instrumentation is one of the techniques used to observe and extract dynamic information from systems during execution [109]. Instrumentation does not modify the system structure and behavior. It may add new variables, insert new code, invoke the original program methods, or replace part of the code by an invocation of a new method that substitutes for the omitted code while performing additional tasks related to the instrumentation process. However, the functionality of the original program should not be affected by instrumentation, and the instrumented program must deliver the same results as the original uninstrumented version. The only side effect caused by instrumentation is
the additional overhead of recompiling the source code, executing the instrumentation code and generating the execution traces. In our method we use a source transformation technique to add source code instrumentation to dynamic web applications. For this purpose we use TXL [55], a programming language for manipulating and experimenting with programming language notations and features. TXL is a powerful source transformation system that has been used in industrial applications involving millions of lines of source code. The TXL transformation process consists of three parts: a context-free “base” grammar for the language to be manipulated, a set of context-free grammatical “overrides” (extensions or modifications) to the base grammar, and a rooted set of source transformation rules to implement transformation of the extensions to the base language. The TXL processor parses the source program into a parse tree, then recursively applies the set of transformation rules, beginning with a main rule, until there are no remaining matches in the parse tree. The transformation is completed by unparsing the transformed tree to the new target source program.

The source transformation approach brings two benefits to the instrumentation process. First, the process can be adapted in a plug and play fashion to deal with any scripting language as a source for instrumentation. This can be done by writing a set of context-free grammatical overrides to the base grammar (in our case at present PHP versions 3,4,5) to add the grammars of the additional languages, along with additional transformation rules to take into account the code of the newly added scripting languages.

Second, source transformation easily adapts to documents that include a mixture of languages and technologies, usually by applying island grammars [166], for example in the approach used by Synytskyy et al. [160] to handle mixed-language web pages. Island grammars divide the input into interesting input forms, called “islands”, and uninteresting sequences of other input items, called “water”. In our case, the islands are the PHP script statements that we want to instrument, and the water is the surrounding static HTML code and text. The main benefit of island grammars is that interesting parts can be identified
include "php.grm"

function main
  replace [program]
    P [program]
  by
    P [instrumentPage]
    [instrumentCookie]
    [instrumentHttpVar]
  end function

Figure 5.2: The main TXL transformation rule

The main rule simply matches the entire input PHP server page source document and applies the transformation subrules instrumentPage, instrumentCookie and instrumentHttpVar globally to it.

without performing a detailed parse of the entire input [131]. In addition, no preprocessing is needed to unify the source code of the web application as in the approach used by WANDA [25].

The instrumentation process begins with the main rule, shown in Figure 5.2, by parsing each web application server page into a parse tree based on the context-free grammar definitions in the grammar file php.grm.

TXL begins by applying the main rule to this tree, and then recursively applies the transformation rules until the entire set of pages is instrumented. The main transformation rules we used for instrumentation are: instrumentPage, which inserts a call to a PHP function that is responsible for tracing page URLs along and their parameters, filtering them and inserting them into an SQL database, instrumentCookie, which inserts a call to a PHP function that captures cookie information and inserts it into the database, and instrumentHttpVar (Figure 5.3), which inserts a call to a PHP function that captures information about HTTP variables and inserts it into the SQL database. As an example to demonstrate the transformation process, we discuss here the details involved in instrumenting HTTP variables.

The rule instrumentHttpVar (Figure 5.3) begins by searching for HTTP variables in its pattern by finding all instances of the grammatical type \[Expr\] (expression) from the PHP grammar, and applying a set of transformation subrules, each of which matches a
CHAPTER 5. AUTOMATED REVERSE ENG. OF UML SD FOR WEB APP.

rule instrumentHttpVar
  replace [Expr]
    E [Expr]
  construct NewE [Expr]
    E [Conv_func_GET]
    [Conv_func_POST]
    [Conv_func_COOKIE]
    [Conv_func_SESSION]
  where not
    NewE [= E]
  by
    NewE
end rule

Figure 5.3: The instrumentHttpVar transform.

The instrumentHttpVar rule finds every PHP expression ([Expr]) and applies four transformation subrules to recognize and instrument instances of HTTP variables for GET, POST, cookies and sessions respectively.

specific HTTP variable. Expressions that are not references to HTTP variables are simply left unchanged since no subrule matches them.

For example, the Conv_func_GET subrule (Figure 5.4) matches any expression of the form \$HTTP_GET_VARS [list of parameters] and replaces it with a call to the instrumenting function HttpVar_track (Param, \$HTTP_GET_VARS [Param], 'GET') . At run time the HttpVar_track () function inserts into the database the parameter names and values and identifies them as GET parameters, returning the value of the HTTP variable to the caller as originally expected without instrumentation. The transformation is supported by a small library of such instrumenting functions that interact with the database. The three other subrules referenced in Figure 5.3 do similar transformations on references to HTTP variables associated with POST, cookies, and sessions.

Figure 5.5 shows the result of transforming the main PHP server page of the PhpBB 2.0 web application. Sections shown in boldface are instrumentation function calls automatically added by our source transformation.
The Conv_func_GET rule transforms each HTTP_GET_VARS expression to a call to the instrumenting function HttpVar_track which tracks the GET parameters in the database and returns the original result.

5.2.2 Filtering and Storing

Once the dynamic web application has been instrumented, execution traces are collected as the application is executed in a web browser. The instrumentation function calls inserted by the source transformations dynamically populate a database with the collected trace information. In our security work, we are primarily interested in collecting unique traces based on user roles. Thus at present we are automatically collecting traces and analyzing them one role at a time. Web crawling tools that mimic user interactions with web applications, such as clicking links, filling in forms and pressing buttons [39, 169] are used to automate collecting traces, while the application roles themselves are recovered manually by studying the software documentation. Roles can be identified from the HTTP session variable and by recovering the way the web application classifies users into roles. (Complete automation of this part is currently a work in progress.)

A user in a specific role can visit a web page more than once, following either the same path or different paths. Capturing each visit and storing it in the database leads to a huge amount of redundant information that can complicate the analysis process. To address this issue, we dynamically optimize the recovered traces by filtering such that we store only unique traces. We recognize unique traces as those that lead to the generation of a new client page, or that generate a previously visited client page using a different path.

New and previously visited client pages are recognized by tracking server pages executed
Figure 5.5: Result of instrumenting the main index.php server page of PhpBB 2.0
Sections in boldface have been added by our instrumenting transformation.

due to user visits. Each server page can generate one or more client pages depending on
the parameters passed to the page. We consider that the same client page is regenerated
if the server page is re-executed without any parameters, or with the same parameters.
In such cases we do not insert the new page into the database unless a different path is
followed in its generation. Our database is constructed to reject any insertion that violates
these conditions. This approach also detects loops in traces, including revisits to pages from
themselves.
5.2.3 Database Analysis and Model Generation

In this phase a sequence diagram is built as a UML 2.1 sequence model based on the execution traces stored in the database. We implement a PHP program to transform the execution traces in the database to the sequence diagram elements shown in Figure 5.6, in the form specified by the Object Management Group (OMG) [138].

Many web applications, including the example presented in this chapter, are not built with object-oriented concepts in mind. Therefore, the interaction elements in our method are the browser session, the generated pages, and the page transitions including their parameters. The instrumentation process generates a record of fine-grained information including such details as http variables, cookies, and sessions. This information is too large to be included in its entirety in the generated sequence diagram, so we have chosen to include only those parameters that are passed in page transitions and shown in the URL address bar.

Execution trace elements, which constitute a database row for a user in a specific role, represent a page ID, a page URL, page parameters, and a page access time. User roles and page URLs for a specific page ID are mapped into sequence diagram lifelines. The transitions between pages and the set of parameters that accompany these transitions are mapped into sequence diagram messages. Page access times in the database are used to determine the order of page transmissions and in the sequence diagram appear as two sets of MessageOccurrenceSpecification events, one for sending the message and the other for receiving it. The message receipt event begins the BehaviorExecutionSpecification fragment (the rectangle bar in the figure), and the message sending event, ExecutionOccurrenceSpecification, ends the same BehaviorExecutionSpecification fragment in that lifeline.
5.3 An Example Application

We have applied PHP2XMI to the analysis of the popular internet bulletin board system PhpBB 2.0 [90]. Using the method described in the previous section, PHP2XMI was able to automatically recover sequence diagrams from user interaction with this application. The system under test was first uploaded to a test server, where it was automatically instrumented by PHP2XMI and executed in a controlled environment. Test scripts were
Figure 5.8: Fine grained information collected by PHP2XMI used to drive the web browser in a manner similar to a user of the application. The test scripts were implemented using *Watir* [169], a library that interfaces Ruby to Microsoft Internet Explorer. Although our framework collects coverage information as part of its instrumentation, in this example we did not attempt to cover all possible execution traces for any specific user role, as that would generate a result much too large to fit in this chapter. Instead we collected a representative set of 15 separate execution records, which generated the database shown in Figure 5.8. From this database PHP2XMI automatically generated a UML sequence diagram model that we imported and visualized using Rational Software Architect (RSA) [105] (Figure 5.9).

Figure 5.7 shows a sample of the execution traces collected by PHP2XMI for a browser session with PhpBB 2.0. The sample represents the scenario of an anonymous user visiting a forum main page and exploring one of the active forums, then trying to do a reply on one of its topics, which requires registered user permission. The user then logs in as an Administrator, replies to the post and switches to the Administrator panel.

Each server page is represented by one or more unique *Page IDs* based on whether the page receives parameters when executed. For example, the server page *posting.php* has four entries in the table, each with a different *Page ID*. This is done to reflect the fact that the server page *posting.php*, in this particular example, generates four different client pages based on the parameters it receives. As another example, the server page *login.php* has four entries. Two of them receive different parameters, and thus PHP2XMI gives them different *Page IDs*. On the other hand, the other two entries do not receive any parameters,
but are from different paths. PHP2XMI inserts both these visits, but gives them the same page ID, in order preserve all the paths that may lead to a specific page. In all cases,
filtering insures that the Pages table does not include any duplicate rows with the same combination of Page Name, Page Parameter and Prev. Page ID. Figure 5.8 shows a view of the join of three tables of the collected database, illustrating the fine-grained information collected by PHP2XMI, including page information, HTTP variables, cookies, and sessions.

In order to map page transitions from rows of Figure 5.7 to sequence diagram elements of Figure 5.6, a lifeline is assigned to the interactive browser session and to each server page. For instance, the Page Name of the second row in Figure 5.7 is mapped into an UML 2.1 element of type \texttt{uml:Lifeline}. The \textit{covered by} property of this lifeline lists the \texttt{xmi:id}'s of the set of \texttt{MessageOccurrenceSpecification} and \texttt{BehaviorExecutionSpecification} events that this lifeline is engaged in, and that to represent the event of sending and receiving the first message between the two lifelines, \texttt{index.php} and \texttt{viewforum.php}. This message is represented using an UML 2.1 element of type \texttt{uml:message}, and its name is the composition of the server page name and the parameter it receives. Finally, UML 2.1 elements of type \texttt{uml:class} represent each lifeline and the set of messages it receives as a class with a set of operations.

Figure 5.9 shows a visualization for a part of the generated model using Rational Software Architect. Once imported into RSA, the diagram can be explored and connected to other UML models to develop a unified understanding of the entire web application.

5.4 Related Work

In CPP2XMI [114], Korshunova et al. describe a reverse engineering tool to extract class, sequence, and activity diagrams from C++ source to XMI 1.1 format. The authors use Columbus/CAN [82] as a fact extractor, which parses the C++ code and generates output in XMI. The authors then analyze the XMI file to extract the information needed for each diagram, using Dot [115] to visualize the output.

Briand et al. [36] propose a method to reverse engineer sequence diagrams from C++ applications, using Perl to implement the automatic instrumentation and Java to transform
traces into sequence diagrams. They propose two meta-models, one to represent the recovered traces and the other to represent sequence diagrams, transforming one to the other based on OCL constraints. While they recover the technical details such as conditions and loops, they do not address visualization of the resulting sequence diagrams.

Jiang et al. [108] propose a method to reverse engineer a sequence diagram in UML 2.0 format from the runtime communication between sample applications and an API. Their method works by monitoring API usage in the sample applications, then filtering the generated execution traces and merging them into a state machine. The analysis of the state machine leads to the recognition of common, optional and alternative parts. A combined sequence diagram is then built and illustrated as a UML 2.0 sequence diagram.

Our approach differs from the above methods in its ability to handle applications with multilingual source code documents, such as web applications. Unlike the Korshunova et al. and Briand et al. methods, PHP2XMI automatically generates XMI 2.1 sequence diagram files which can be visualized directly in any UML 2.1 toolset. While both methods use filtration, ours is focussed on minimizing the database to optimize extraction of sequence diagrams, whereas CPP2XMI gathers a much richer initial XMI representation and then filters to extract sequence elements.

Many methods have been proposed to extract behavioral models from web applications for the purpose of testing. Ricca and Tonella [148], for example, construct a UML object model of a web application’s pages, frames, forms and the links between them from the dynamic HTML output of the application. While their object model is aimed at generating test cases, our approach uses automated test cases implemented using Watir [169] and a coverage metric to recover a model for the purpose of testing security properties of dynamic web applications. A major difference between our recovered behavioral models, the Ricca and Tonella custom models, and almost all other previous work in this field, is that ours are the only recovered behavioral models represented using standard UML2.1 sequence diagrams. This is an important advantage from a practical point of view, since it allows
import, analysis and manipulation using standard UML tools such as Rational Software Architect. A detailed analysis of the state of the art in the field of web applications modeling for analysis and testing can be found in our survey[13].

Although it is not designed for web applications testing, the work most similar to our approach is WANDA (Web ApplicatioNs Dynamic Analyzer) [25], which collects execution traces in a similar way. In WANDA execution traces are not filtered, which leads to a huge amount of redundant information stored in the database, and consequently tends to yield cluttered sequence diagrams that are difficult to comprehend. Di Lucca and Di Penta [67] have proposed an approach to filter the execution traces generated by WANDA by using the WARE tool [123] to identify groups of equivalent Built Client Pages (i.e., client pages dynamically built by server pages which share common features). A filtration of the execution traces collected by WANDA is then performed based on page clustering. This method has been evaluated on a PHP web application.

PHP2XMI differs from WANDA in using source transformation technology for the parsing and the instrumentation phases, which aids in eliminating the overhead caused by any preprocessing needed to deal with multilingual documents. The filtration proposed by Di Lucca and Di Penta is based on the static analysis provided by WARE, which may identify the web application pages, but does not preserve all the possible paths that the application may allow its users. For this reason their method is not sufficient for security testing purposes, whereas PHP2XMI’s filtering is tailored to the task. WANDA’s unfiltered model can also have scalability problems, whereas PHP2XMI’s dynamic filtering bounds the number of lifelines and messages in the model to the number of server pages and paths between them. The output format generated by PHP2XMI is in the XMI 2.1 standard for model interchange between the UML tools, whereas WANDA uses a custom local format.
5.5 Conclusion and Future Work

We have presented an automated approach and practical tool to instrument dynamic web applications using source transformation technology to recover a dynamic behavior model from observed interaction. The tool is able to automatically reverse engineer UML 2.1 sequence diagrams from PHP-based web applications. The result can be imported and visualized in any UML 2.1 toolset. The approach we use filters execution traces directly on insertion into the database, automatically eliminating redundant information that may complicate the understanding process.

In this chapter the interaction elements in the resulting sequence diagram are the user and the dynamic pages of a browser session, represented as lifelines, and the dynamic transitions between the pages along with their parameters, represented as messages. In the next chapter we present an approach which supports the process of raising the sequence diagram to the entity level from the page level.

At this stage, we don’t try to enumerate all the possible executions for each role, but we will see in Chapter 7 how we use an instrumentation coverage technique to handle this issue and to provide a measure for the completeness of the generated sequence diagram.
Chapter 6

WAFA: Fine-grained Dynamic Analysis of Web Applications

In Chapter 5 we recovered sequence diagrams for test sessions at the page level. We have used the work described in this chapter to extend the recovered sequence diagrams to include interactions with the database. The extended sequence diagrams are then used as part of our RBAC model-based analysis framework.

Database interactions are a vital source of information in the analysis of highly dynamic systems such as web applications. Most web application security vulnerabilities, such as SQL injection and broken access control, can be traced to problems in database interactions, which are implemented as a set of embedded or constructed SQL statements. The identification and analysis of these embedded statements as an integral component of the host application requires complex analysis including robust parsing, pattern matching, control flow and data flow analysis.

In this chapter, we propose an approach to this problem using source transformation technology. A rich model of fine-grained information is extracted from dynamic web applications, allowing us to reason not only about the SQL embedded system, but also about page
access, server environment variables, cookies and session management functions. We evaluate our system on the popular bulletin board web application PhpBB, a PHP / MySQL-based dynamic web application.

This chapter is organized as follows: Section 6.1 provides the motivation for our approach. Section 6.2 presents WAFA, our fine-grained analysis approach. Section 6.3 presents the instrumentation technique used in WAFA. Section 6.4 evaluates WAFA on some test cases from PhpBB. Section 6.5 relates our approach to previous work. Finally, Section 6.6 concludes the chapter.

### 6.1 Motivation

Web applications are one of many kinds of systems with multiple components that dynamically interact to deliver a specific business process. Sophisticated static and dynamic analysis is needed when reverse engineering web applications to extract all relevant information from the various components, and to correlate the extracted information to model the actual application behavior.

Several approaches for reverse engineering web applications have been proposed, most of which have focused on extracting the structural levels of the application, such as pages, frames, forms, and hyperlinks [123, 148]. Others have addressed particular aspects of application behavior such as interaction with the browser [65], and still others have aimed at extracting a higher level abstract behavioral model which describes the basic application elements, but does not combine the results or extract the details of database interaction [68, 25, 67].

Most current dynamic web application business processes depend on the support of a database back end. A great deal of information is stored in the database, including critical knowledge such as session management and access permissions. Dynamically identifying and extracting database interactions alone can be misleading, as they do not reflect the actual intended behavior of the application business process as a whole. Analyzing database
interactions in the context of the entire web application may clarify aspects of the business process hidden behind the web application presentation level. Since many web application vulnerabilities rely on modifying the database interaction statements at runtime, there is a need to analyze not only the values of the SQL statements constructed at runtime, but also the original source of the host language statements used to construct the SQL statements.

Database interactions are often implemented in web applications using a combination of string concatenation expressions and host language statements that work together to construct an SQL statement. These expressions and statements are composed of constant strings and application variables. Identifying, extracting, and analyzing these dynamically constructed statements in the context of the overall system is not a trivial process.

In this chapter we propose an approach to analyze dynamic web applications, extracting a fine-grained model aimed at understanding the interaction between the web application and the database. Our approach is different from other methods in the following aspects:

- An automated instrumentation methodology that handles mixed languages, and can be easily extended to other technologies and host languages.
- Extraction of a model that relates information about pages, server environment variables, database interactions and host language source statements. This model is stored in a database to facilitate accessibility and future analysis.
- Extraction of a web application’s embedded SQL components, which is comprised of the source of the original SQL statements, and the corresponding execution instances. The SQL components also include both static host application variables and dynamic server environment variables.

This work takes place in the context of a larger project on web application security, specifically the analysis of role based access control [11]. In the class of web applications we are analyzing, user roles and access permissions are stored in the application database with the other application data. Providing the context of the database interactions allows us to
better analyze these interactions and recover the details of user roles and permissions. In the previous chapter we recovered sequence diagrams for test sessions at the page level. We have used the work described in this chapter to extend the recovered sequence diagrams to include interactions with the database. The extended sequence diagrams are then used as part of a model-based analysis of access control.

Another possible application of our approach is the analysis of SQL injection attacks in a way similar to Halfond and Orso [94]. Differences between the structure of the runtime query and the source version of the query indicate input that may have changed the meaning of the query.

6.2 Approach

Figure 6.1 shows the architecture of our approach, called WAFA (Web Application Fine-grained Analysis). An instrumentation transformation is used to analyze the source code of the application, inserting appropriate calls to an instrumentation library written in PHP, Figure 6.1(A). The instrumented application is deployed in a testing environment, where a web crawling tool uses predefined test cases to exercise the web application. Since test case design and coverage are important issues, they are discussed in detail in Chapter 7. We focus our discussion here on recovering the interaction behavior between the web application
As the tests run, Figure 6.1(B), the instrumentation library inserts information about each query into a separate instrumentation database. A source transformation is performed on each SQL statement at runtime to analyze its structure. This structural information is inserted in the database along with the query, Figure 6.1(C). The schema for our database is shown in Figure 7.2. It is comprised of five tables and one view. The Server Pages table is used to keep track of access to individual pages, while the other four tables contain information about the HTTP variables, environment variables, cookies and database statements associated with each page, linked using the Page_Ins_ID field. We combine the information from the various tables into a single unified trace view in the Dynamic Analysis view. The following section elaborates the approach in more detail.

### 6.3 Instrumentation Methodology

We automatically analyze and add source code instrumentation to web application source using TXL [55], a programming language designed for manipulating and experimenting with programming language notations and features. TXL is a powerful source transformation
system that has been used in industrial applications involving millions of lines of source code. The TXL processor takes as input a context-free grammar for the language to be manipulated, parses the source program into a parse tree, and then recursively applies a set of transformation rules, beginning with a main rule, until there are no remaining matches in the parse tree. The transformation is completed by unparsing the transformed tree to the new target source program.

Our implementation presently instruments web applications written in PHP(3,4,5) and MySQL(5.x). However, our TXL-based approach is easily adapted to deal with other scripting languages and database engines. Documents that include a mixture of languages and technologies are easily handled by employing island grammars [160]. In our implementation, the islands are PHP code, while the HTML source and document text are considered water. Island grammars simplify the transformation as interesting elements can be identified and analyzed without parsing the entire document.

6.3.1 Instrumenting and Collecting Page Information

In our process, page access is tracked by querying the server execution environment information created by the web server when the user interacts with the web application. Pages are instrumented by inserting an instrumenting function call at the head of each PHP dynamic page that performs this query to retrieve the page URL address along with any parameters passed to the page as well as the page access time, and inserts a trace element of this information in the page access table of our instrumenting database.

As part of the insertion of each page access trace element in the database, an analysis is performed to insure that the inserted trace element is unique. We recognize unique trace elements as those that lead to the generation of a new client page, or that generate a previously visited client page using a different path. Each server page can generate one or more client pages depending on the parameters passed to the page. We consider a generated client page to be the same if the same server page is re-executed without any parameters,
or with the same parameters. In such cases we do not insert the new page into the database unless a different path is followed in its generation. Our database is constructed to reject any insertion that violates these conditions.

### 6.3.2 Instrumenting and Collecting Server Information

In PHP, predefined global variables contain information about a script’s environment, such as the client’s web browser, the HTTP host, and the HTTP connection. All those variables along with any function that manipulates them are instrumented. When the application is executed, a trace element on each piece of information will be inserted in a database for later analysis.

In particular, four types of HTTP variables are identified and instrumented, namely, the `GET`, `POST`, `COOKIE`, and `SESSION` variables. References to these variables are replaced with a call to the instrumenting function `HttpVar_track()`, which takes as parameters the HTTP variable name, value, type, name of the PHP variable to which the HTTP variable is assigned (if any), and the page number in which the variable is located. After adding the information to our database, the function returns the value of the HTTP variable so as to preserve the semantics of the code. An example of this instrumentation is shown in Figure 6.3. In the Figure, references to the HTTP parameter `mode` are logged in the database along with the fact that it is assigned to the PHP variable `$mode`.

### 6.3.3 Cookies and session management functions

In the HTTP Protocol, state is preserved between consecutive requests using cookies, a small identifier that is stored in the client browser and sent back to the server on each subsequent request. This identifier is used by the web application to store and retrieve data specific to that session. The predefined global variables `$HTTP_COOKIE_VAR` is used to access the cookies, and this variable is instrumented as described in the previous section. In addition, we instrument changes to the cookie by adding a call to our instrumentation function, `cookie_track()`, after each call to the PHP function `setcookie()`. Our function takes the
CHAPTER 6. FINE-GRAINED DYNAMIC ANALYSIS OF WEB APP.

<?php
...
if ( !empty($HTTP_POST_VARS['mode']) || !empty($HTTP_GET_VARS['mode']) )
  $mode = ( !empty($HTTP_POST_VARS['mode']) ) ?
    $HTTP_POST_VARS['mode'] : $HTTP_GET_VARS['mode'];

Figure 6.3: Results of instrumenting server environment variables in a snippet of code in PhbBB 2.0 application

Each HTTP reference variable is identified and transformed into an instrumenting function call which is passed the variable name, assigned variable name, variable value, unique ids for the variable name and the page name, and the variable type. The instrumenting function returns the variable value as its return value.

<?php
...
if ( !empty($HTTP_POST_VARS['mode']) || !empty($HTTP_GET_VARS['mode']) )
  $mode = ( !empty ($HTTP_POST_VARS ['mode'])) ?
    HttpVar_track ('$mode', 'mode', $HTTP_POST_VARS ['mode'],186,192, POST):
    HttpVar_track ('$mode', 'mode', $HTTP_GET_VARS ['mode'],186,191, GET);
...
?>

Figure 6.4: Result of instrumenting cookie management functions in a snippet of code in the PhbBB 2.0 application

Each cookie management function is identified and an additional instrumenting function call that captures all cookie management function parameters is added

same parameters and stores them in the Cookie table of our instrumentation database. An example of this transformation is shown in Figure 6.4. The line in bold is added to the contents of the if statement directly after the call to setcookie(). If cookies have been disabled, then session information is encoded into URLs using PHP utility routines. We instrument the calls to these functions in a similar manner.

<?php
...
if ($userdata ['session_logged_in']){
  setcookie ($board_config ['cookie_name'] .'_f_all', time (), 0,
    $board_config ['cookie_path'], $board_config ['cookie_domain'],
    $board_config ['cookie_secure']);

  cookie_track ($board_config ['cookie_name'] .'_f_all', time (), 0,
    $board_config ['cookie_path'], $board_config ['cookie_domain'],
    $board_config ['cookie_secure']);
}
...
?>
CHAPTER 6. FINE-GRAINED DYNAMIC ANALYSIS OF WEB APP.

% SqlCommandString is any string or character literal 
%    beginning with an SQL query verb 
tokens
  SqlCommandString  "'SELECT #'*"  
  |  "'INSERT #'*"  
  |  "'DELETE #'*"  
  |  "'UPDATE #'*"  
  |  ""SELECT #""\"\"  
  |  ""INSERT #""\"\"  
  |  ""DELETE #""\"\"  
  |  ""UPDATE #""\"\"  
% and any others needed
end tokens

% SqlExpr is any string concatenation expression 
%    that begins with one of the above magic words 
define SqlExpr
  [SqlCommandString] [CatAddExpr*]
end define

Figure 6.5: PHP grammar extension to recognize guest patterns (SQL statements)

6.3.4 Instrumenting and Collecting Database Interactions

The most complex set of transformations is used to identify and instrument the interactions with the database. We break the transformation into four parts. The first is to identify dynamically constructed SQL statements. Once they are identified, then we can insert code to construct a string value that contains the same SQL statement, but with the names of the application variables that are used instead of the variables. Together the two strings are inserted into the instrumentation database. An analysis of each of the SQL statements is done to identify the type of the statement, and the key elements present in the statement. This information is added to the database entry for the statement.

Identifying Dynamically Constructed SQL Statements

Our approach uses a separate lexical token class (SqlCommandString) to distinguish string literals that begin with the SQL keywords Select, Insert, Update, Delete, Create, Alter and Drop from other strings in the PHP source text. This allows us to use the parser to recognize concatenation expressions built from these strings (Figure 6.5). The transformation then targets assignment statements that use these strings to build larger
strings. Once an assignment is found that uses one of these SQL keyword strings, other assignments using the same PHP variable are also checked and instrumented. The code is normalized prior to the transformations, moving string expressions to separate assignment statements and replacing them with a temporary PHP variable.

Constructing SQL Statement Sources

As the SQL assignment statements identified in the previous section are encountered, our transformation inserts additional assignments into the code to accumulate the source representation of the SQL statement as shown in bold in Figure 6.6. The SQL command that is constructed by the code in the Figure is the string:

"INSERT INTO phpbb_themes(themes_id,template_name,style_name) VALUES('8','test','test_style');"

At the same time, the assignment statements inserted by our instrumentation construct a second string:

"INSERT INTO phpbb_themes($db_fields[$i], $db_fields[$i], 
$db_fields[$i]) VALUES ($db_values[$i], $db_values[$i], $db_values[$i]);"

Just before the SQL command string is sent to the database, both strings are inserted into the instrumentation database. This gives us a runtime snapshot that contains not only the values used in the actual SQL query, but also the PHP variables from which the query was constructed. Figure 6.8 shows the TXL transformation rule instrumentQueriesSource that identifies an assignment containing an SQL Command String and adds the instrumentation assignments.

The instrumented SQL statement is constructed in the TXL variable SQL_E_Sorce. The parts of the SQL expression are passed as a parameter to the collectparameters function which processes them one part a time. It classifies each SQL part it receives into one of three categories:
CHAPTER 6. FINE-GRAINED DYNAMIC ANALYSIS OF WEB APP.

Figure 6.6: Instrumented snippet of code for PhpBB2.0 application - 1

Sections in boldface have been added by our instrumenting transformation. This example demonstrates how the dynamic SQL statement is constructed from its fragments using forward flow analysis.

1. **Constant variables.** Constant variables are the same for all SQL statement instances, so the function concatenates them to the result without quotes, adding the actual values at run time. The first assignment shown in Figure 6.6 references the constant
Figure 6.7: Instrumented snippet of code for PhpBB2.0 application - 2

Sections in boldface have been added by our instrumenting transformation. This example shows how the dynamic SQL statement is expanded from its fragments using backward flow analysis.

variable, THEMES_TABLE. This variable, which contains the name of the table, is concatenated without quotes in the instrumentation assignment on the next line, shown in bold. Note that at this point in time, the assignment statements have yet to be generated. The function is constructing the expressions that will occur on the right hand side of the instrumentation assignment statements.

2. PHP variables. These variables are bound to different values at run time, generating different versions of the SQL statement. The function concatenates a quoted version of the variables to the result, protecting the variable from runtime substitution. In Figure 6.6, the assignments of the PHP variables, $db_fields[$i] and $db_values[$i], are quoted when concatenated to the instrumentation variable $GLOBAL["Sql_source"] [549] [0].

In some situations, an SQL statement is constructed in fragments using multiple PHP variables before assembling the final SQL statement. Currently, we are using naming conventions (ending in _sql ) to identify the part variables, which are collected in a
TXL global variable. Once identified, we instrument each of the variables in a similar manner. In Figure 6.7, the SQL statement assigned to $sql is constructed from parts contained in the $count_sql, $join_sql_table, $join_sql, and $order_sql variables. The Figure shows how each of the part variables are instrumented and collected in the global array SqlParts. The final assembly of the SQL statement is shown at the bottom of the Figure. The instrumentation string is constructed by concatenating the parts that were previously stored in the SqlParts array.

3. String expressions. In PHP, string expressions can have two forms: double quoted and single quoted strings. The difference between the two forms is that embedded PHP variables will be substituted in double quoted strings but not in single quoted strings. Thus the collectparameters function transforms double quoted strings into single quoted ones to protect embedded variables, unless the variables are recognized as SQL fragment variables. In Figure 6.7 the string literals in the SQL fragment assigned to the PHP variable, $join_sql have been changed to single quotes when assigned to the instrumentation array.

Once the expressions have been collected, a unique identifier for the SQL statement is generated. The TXL rule in Figure 6.8 generates the assignment statements to collect the SQL instrumentation strings in the PHP global array SqlSource using the unique identifier. The unique identifier for the page (uniquePageid) is also stored in the array so that it can be stored in the database to link the SQL statement to the page from which it was generated.

The TXL rule instrumSqlStmtParts is called on the remaining source code to search for and mark any concatenation statement that may contribute to the construction of the SQL statement identified in the rule. It takes as parameters the PHP variable from the left hand side of the assignment, and the statement’s unique identifier. Figure 6.6 shows how the SQL statement with the unique id 594 is constructed from its fragments that are
CHAPTER 6. FINE-GRAINED DYNAMIC ANALYSIS OF WEB APP.

rule instrumentQueriesSource
  % Get the page ID for the page containing the SQL query source construction
  import uniqePageid [id]

  % Find the first statement of any SQL query source construction
  replace [TopStatement*]
    Rest [TopStatement*]
    ...
  % Collect and expand SQL statement source
  construct SQLE_Source [Expr]
    SqlE [collectparameters]
  % Create a unique id for the constructed SQL statement source
  construct uniqeid [id]
    _ []
  % Replace the statement with an instrumented version
  by /
    % Original statement
    OCV AsOp Sql;
    % Added instrumentation statements
    '$GLOBALS'["Sql_Source"]['uniqeid']'[0'] = SQLE_Source ;
    '$GLOBALS'["Sql_Source"]['uniqeid']'[1'] = uniqePageid ;
    '$GLOBALS'['Sql_index']' = uniqeid ;
  % And instrument any following SQL query source construction fragments
  Rest [instrumSqlStmParts OCV uniqeid]
end rule

Figure 6.8: The instrumentQueriesSource transformation rule

The instrumentQueriesSource rule captures each SQL statement and transforms it into its source by collecting the statement fragments and manipulating them to keep any embedded variable unsubstituted scattered in conditional statements and loops.

Binding SQL Statement Source with its Runtime Instances

In the previous subsection, we identified and globally instrumented SQL statement construction. We now identify and instrument the actual SQL execution points, combining the instantiated SQL statements with the SQL statement source. The TXL rule instrumentSQL,

rule instrumentSQL
  replace [Expr]
    'mysql_query ( Q [Expr], R [Expr] )
  by
    'mysql_query_track ( 'mysql_query, Q, R,
    '$GLOBALS'["Sql_Source"]')['GLOBALS'["Sql_Source"]'],'GLOBALS'["Sql_index"]')
end rule

Figure 6.9: The instrumentSQL transformation rule

The instrumentSQL rule captures each SQL execution statement and transforms it into a call to an instrumenting function that combines the globally constructed SQL statement source with its execution instance

...
shown in Figure 6.9, identifies calls to the `mysql_query` PHP function and replaces them with the instrumentation function `mysql_query_track()`. This function takes as parameters the original function’s SQL statement Q and database connection R, as well as two additional instrumentation parameters: the SQL statement source, from the global array "Sql_Source" (mapped by the SQL statement unique identifier in the global variable "Sql_index"), and the SQL statement’s unique identifier.

### Analyzing SQL Statement Sources

During runtime execution of the instrumented PHP application, the instrumentation function `mysql_query_track()` is called to insert trace elements into the Database Interaction database table. This function invokes a TXL program that parses the SQL source statement, identifying the statement type, the application variables and the database tables used in the statement. It also identifies any PHP variables embedded in the SQL statement and associates them with the syntactic part in which they are used. For example, the program will identify whether a PHP variable is used in a `WHERE` clause or an `ORDER BY` clause. The results of this analysis are also stored in the Database Interaction table so that PHP variables can be directly linked to the run-time values and database interactions they control.
### Table 6.3: Sample trace elements for the Database Interactions database table

Traces related to PageIDs 15 and 17, viewforum.php and viewtopic.php

#### 6.4 Evaluation

We have evaluated our approach by analyzing a production dynamic web application, *PhpBB 2.0* (a popular internet forum system) [90]. We use Web Application Testing In Ruby (WATIR) [39, 169], a library used to script web browsers, to help automate the collection of usage traces.

Tables 6.1, 6.2, and 6.3 show a subset of the results of one test case (visits to Page_ID 15 and 17) for an anonymous user interacting with a PhpBB forum. The tables also show some of the HTTP variables and database interactions generated by the visits. Based on the Page_ID values, these three tables are joined into a single view for ease of analysis.

In the tables we can see that the viewforum page, with Page_ID 15, passes values to the HTTP Get variable `f` and the HTTP Cookies variable `sid`. Table 6.2, *HTTP Variables*, shows that the `f` Get variable is assigned to the `$forum_id` PHP variable, and `sid` is assigned to the `$session_id` PHP variable. Table 6.3, *Database Interactions*, shows some of the SQL statements generated from this interaction. We can see that SQLStmt_ID 2788 uses the...
CHAPTER 6. FINE-GRAINED DYNAMIC ANALYSIS OF WEB APP.

<table>
<thead>
<tr>
<th>Filtered client pages</th>
<th>Utility pages visited</th>
<th>Client pages generated</th>
<th>All visited pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>180</td>
<td>50</td>
<td>230</td>
</tr>
</tbody>
</table>

Table 6.4: Trace statistics for anonymous user interactions with a PhpBB 2.0 forum

PHP variable $forum_id to retrieve the forum information, while the other database interactions shown for this page visit do not depend on user inputs to perform their transactions.

The Database Interactions table columns are populated as a result of the execution of the instrumented application except the two columns, EntityName and SQL Parts, which come from analysis of the SQL source statements using TXL. The column EntityName shows the name of database tables that the SQL statements are performed on. Column Php Parts shows three pieces of information, the SQL statement type (SelectStm, UpdateStm, and so on), the embedded PHP variable and its syntactic location within the SQL statement (such as the WHERE clause), and the statement’s WHERE and SELECT expressions. This example illustrates the key benefit of our approach, linking the runtime instance of the query to the elements used to assemble it including PHP variables and HTTP request variables.

Table 6.4 shows some statistics for the test scenario of an anonymous user visiting a PhpBB forum. It shows the number and type of SQL statements executed during this interaction, the number of SQL statements that depend on user inputs, the location of the embedded PHP variables in the SQL statement, and the number and the type of the HTTP variables used in the interaction. Table 6.4 shows the total number of pages visited during this interaction as well as the number of filtered client pages stored and analyzed by our tool. In the work described in this chapter, both the SQL query source and the runtime SQL instances are collected at run time. In a new experiment we have shown how the SQL source statements can be extracted statically by slicing the instrumentation aspect of an
instrumented PHP application into a separate PHP program. When executed offline, this program inserts the original SQL statement sources into the database. The runtime SQL commands are then collected dynamically when the application is executed and related to the original SQL statement sources as described in section 6.3.4. The dynamic approach of this chapter adds an average overhead of approximately 70%, while the static optimization reduces the average overhead to 30%.

### 6.5 Related Work

Several techniques have been proposed to support understanding and analysis of web applications using reverse engineering. A detailed study of the state of the art in this field can be found in our recent survey [13]. To the best of our knowledge this is the first approach that dynamically analyzes database interactions in combination with other web application basic elements of information, such as pages, server environment variables, application variables and session and cookie management functions. The combined information gained from all of those sources provides a strong infrastructure that can serve many analysis tasks requiring precise fined-grained information, such as web application security analysis.

The problems of identifying and analyzing database interactions have been previously studied for standard systems. For instance, Cleve and Hainaut [51] use aspect-based tracing to relate and extract the basic components of the dynamic SQL query. This includes the basic dynamic query, the variable parts of the query, the query result, in addition to some

<table>
<thead>
<tr>
<th>Approach</th>
<th>Static\Dynamic Analysis</th>
<th>Host language Application</th>
<th>Extendable</th>
<th>Client Application Source code</th>
<th>Parsing tech.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrumenting transformation (WAF)</td>
<td>S\D</td>
<td>PHP \Web Applications</td>
<td>Yes</td>
<td>Syntactic modification</td>
<td>Island grammar</td>
</tr>
<tr>
<td>Tracing Aspects</td>
<td>D</td>
<td>Java</td>
<td>No</td>
<td>No modification</td>
<td>Independent parsers</td>
</tr>
<tr>
<td>Symbolic execution and inference rules</td>
<td>S</td>
<td>PHP\Web Applications</td>
<td>Yes</td>
<td>No modification</td>
<td>JIForester(Parser)</td>
</tr>
<tr>
<td>Control and flow analysis</td>
<td>S</td>
<td>PL/SQL, COBOL, Visual Basic</td>
<td>Yes</td>
<td>No modification</td>
<td>ANTLR(tokenizer)</td>
</tr>
</tbody>
</table>

Table 6.5: Related work comparisons
environment variable such as the class and the line number in which the query called. The authors provide three kinds of post analysis of the trace elements after the program is executed which include constant/variable identification, value-based dependency analysis and static statement restructuring. However their approach works only with prepared statements and does not handle the case where the SQL statement is constructed in string variables and passed as a string to the database API. Their approach is also yet to be evaluated on a production system.

Brink et al. [165] propose a tool for assessing the quality of database interactions in standard applications. They first extracted embedded SQL statements using control and dataflow analysis. The identification of SQL string literals are done using a standard Java program that tokenizes the source program based on predefined SDF grammars. Then they collected the identified parts in a query object which includes information about the reconstructed query, its location, and name and type of variables required for the reconstructed query. A post analysis is done over the extracted queries for quality assessment purposes. This analysis is done for PL/SQL, COBOL, Visual basic, and Java. The authors’ aim is to extract the queries for quality assessment, while our aim is to reverse engineer a web application to gain a rich infrastructure that can support different kind of analysis including quality assessment of database interactions. While the identification of SQL queries in this work is similar to ours, using source transformation technology we combine the process of parsing, pattern matching and flow analysis into a single coherent step, yielding a faster and more flexible analysis with more accurate results.

Ngo and Tan [135] propose an automatic static technique to extract database interaction points from web applications. The approach first identifies all program paths that include a database interaction and slice them out as an interaction Control Flow Graph (ICFG), then each interaction path is symbolically executed, and all possible interaction types are derived from the generated symbolic expression using inference rules. Evaluating the approach on a case study, the approach is able to extract 80% of the database interactions. The complexity
of the extraction process is high as it is composed of 5 stages, and is affected by factors such as number of the interaction paths (i-paths), and the length and complexity of each i-path. The authors also do not specify how to handle SQL statements constructed from sequences of string concatenations. Table 6.5 summarizes and compares related work.

6.6 Future Work and Conclusions

We have presented WAFA, an automated reverse engineering approach to recover fine-grained interaction behavior of dynamic web applications. To the best of our knowledge, our approach is the first one to extract the web application’s embedded SQL subsystem, which includes both the original SQL statement source as well as corresponding execution instances, and an analysis to attach it to both static host application variables and dynamic server environment variables.

We used the fine-grained analysis described in this chapter to help construct an entity-based sequence diagram. Chapter 8 elaborates on this in more detail. In the next chapter, we will present a new instrumentation coverage approach to provide a measure for the completeness of the dynamic analysis provided in this chapter.
Chapter 7

DWASTIC: Automating Coverage Metrics For Dynamic Web Applications

Building comprehensive test suites for web applications poses new challenges in software testing. Coverage criteria used for traditional systems to assess the quality of test cases are simply not sufficient for complex dynamic applications. As a result, faults in web applications can often be traced to insufficient testing coverage of the complex interactions between the components.

This chapter explains how DWASTIC (Dynamic Web ApplicationS Testing Instrumentation Coverage) is used to augment the dynamic analysis, presented in Chapters 5 and 6, with instrumentation for code coverage in order to decrease the number of false positives due to an analysis that yields a model that only partially covers the code (leading to verification of properties that may in fact not hold). The chapter presents a new set of coverage criteria for web applications, based on page access, use of server variables, and interactions with the database. Following an instrumentation transformation to insert dynamic tracking of these
aspects, a static analysis is used to automatically create a coverage database by extracting and executing only the instrumentation statements of the program. The database is then updated dynamically during execution by the instrumentation calls themselves. We have evaluated the usefulness of our coverage criteria and the feasibility and precision of our approach on the popular bulletin board web application PhpBB.

This chapter is structured as follows. Section 7.1 motivates our approach. Section 7.2 presents our proposed coverage metrics. Section 7.3 presents the details of our approach. Section 7.4 presents an evaluation and an example that demonstrates our method on a real system. Section 7.5 relates our efforts to previous work. Finally, Section 7.6 outlines our conclusions and plans for future work.

7.1 Motivation

Testing is one of the most essential yet complex activities in web application development and maintenance. The dynamic distributed structure of web applications poses new challenges to building comprehensive test suites. Users interact with application pages, providing various inputs that are used to instantiate the server environment variables. These variables are then used to interact with the database back-end, retrieving information used to dynamically construct new client pages to be sent back to the users.

Database interaction is the most critical part of this cycle, and often requires extensive testing. For this reason, several approaches have been proposed to assess the correctness of database interactions in standard database systems, for example Cabal and Tuya [37] and Wilmor and Embury [171].

Coverage metrics have been proposed on different levels of granularity for SQL statements either as an isolated component or as an embedded component in the whole system. However, most of these approaches either do not provide automation for coverage assessment, or do not consider other kinds of interactions. Our approach is specialized for web
applications, handling similar issues to those tailored for conventional database applications while at the same time addressing the new challenges related to the distributed and dynamic structure of web applications. We have implemented our approach in an extendable and precise tool.

The contributions of this chapter are:

- A set of coverage criteria for the testing of dynamic web applications. This can be used to assess thoroughness and the adequacy of test suites applied on different levels, such as the page access level, the server environment variable level, and the database level, as well as interactions between the levels.

- An extendable, automated approach and tool to instrument, collect and analyze the coverage information. The tool also statically extracts and analyzes the embedded SQL subsystems from dynamic web applications.

7.1.1 Web Application Testing

Our tool, called DWASTIC (Dynamic Web ApplicationS Testing Instrumentation Coverage), can be used to support any testing activity for web applications. It focuses the testing efforts on the vulnerable parts of the code, which are most likely the source of web application faults and attacks such as SQL injection. It also provides a direct way to trace the part of the code that is not covered by test cases.

DWASTIC is an essential part of a framework aimed at testing the conformance of dynamic web applications with role-based access control security policies [11]. In the framework, a role-based access control (RBAC) security model is recovered from the dynamic web application using a combination of static and dynamic analysis techniques. This chapter explains how DWASTIC is used to augment the dynamic analysis with instrumentation for code coverage in order to decrease the number of false positives due to an analysis that yields a model that only partially covers the code (leading to verification of properties
that may in fact not hold).

### 7.2 Web Application Coverage Metrics

In this section we propose three test coverage dimensions that are specifically tailored to web applications: web application pages, server environment variables and database interactions. These are not meant to replace traditional code coverage metrics, rather to augment them to provide specific coverage measures for the client and database interaction aspects of dynamic web applications as well. Our criteria subsume the criteria proposed for database systems [171, 37] and include new measures that are specialized for web applications. The concern in web application testing is whether the application as a whole behaves as specified, and this cannot be determined without thorough testing of all three levels of interaction.

While the coverage criteria we propose can be used to support many different testing activities, our specific aim is to provide a completeness measure for extracting an access control security model from a web application under test. This requires that we ensure coverage of all client pages that can be generated from the application, all database interactions applied on application entities, and all user inputs passed to server pages or SQL statements that can influence the dynamically constructed client pages. In the following subsections we elaborate on our proposed criteria and how they can serve our aim.

#### 7.2.1 Page Access Coverage

Page coverage measures the adequacy of test cases for ensuring that all server pages are executed at least once and are running properly, the measure can be expressed as:

\[
\text{Page Coverage} = \frac{\#_{\text{cov.Pages}}}{\text{total}\#_{\text{application pages}}}
\]

The equation measures the ratio of executed server pages to the total number of the application server pages.
7.2.2 SQL Statement Coverage

SQL statement coverage measures the adequacy of test cases to ensure that all possible SQL statements, including dynamically constructed ones, are tested at least once. These statements are different from those which perform API calls to issue commands to the database, such as `mysql_query($SQL Statement, db_connect_id)` in PHP. Using the terminology introduced by the database community, these API calls are called database interaction points. Coverage based on database interaction points is not sufficient, as each specific database interaction point can issue multiple forms of dynamically constructed SQL statement.

Our SQL statement coverage measure is done both on the level of the whole web application and on the level of each individual server page. The measure based on the application level can be expressed as:

\[
\text{SQL Stm Coverage} = \frac{\#\text{of cov.SQL Stms}}{\text{total}\#\text{of the application SQL Stms}}
\]

The measure on the page level can be expressed as:

\[
\text{Page SQL Stm Coverage} = \frac{\#\text{of cov.SQL Stms in Page}}{\text{total}\#\text{of SQL Stms in page}}
\]

7.2.3 Server Environment Variables Coverage

Server environment variables are variables returned by HTTP forms on generated pages using GET or POST. Server environment variables coverage measures the adequacy of test cases to insure the coverage of all server environment variables at the level of the web application, the individual server page level and the SQL statement level. The measure based on the application level can be expressed as:

\[
\text{Server Env Var Cov.} = \frac{\#\text{of populated Server Env Var}}{\text{total}\#\text{of the application Server Env Var}}
\]

The measure for the page level can be expressed as:

\[
\text{Page Server Env Var Cov.} = \frac{\#\text{of cov.Server Env Var in cov. Pages}}{\text{total}\#\text{of Server Env Var in page}}
\]
And the measure for the SQL statement level can be expressed as:

\[
\text{SQL\_Server\_Env\_Var \ Cov.} = \frac{\# \text{of cov. SQL Stm with Server Env Var}}{\text{total \# of SQL Stm with Server Env Var}}
\]

In each case the equation measures the ratio of the number of server environment variables covered to the total number of variables usage on the different levels.

### 7.3 Constructing the Coverage Database

Figure 7.1 shows the architecture of our approach. An instrumentation transformation is used to analyze the source code of the application, identifying and globally marking instances of the coverage criteria and inserting appropriate calls to an instrumentation coverage library developed in PHP. These instrumentation calls will update the coverage database as the program is executed (Figure 7.1).

In order to insure that we have an accurate table of all of the instances to be covered, the initial coverage database itself is automatically derived from the instrumented application. This is done by slicing the instrumentation statements from the instrumented PHP code into a separate PHP program augmented with database calls (Figure 7.1). As the augmented slice is executed, the program builds the initial coverage database by adding a coverage table entry to a global array as each instrumentation statement in the slice is executed.
During insertion, another transformation also analyzes the type of the SQL statement to identify and insert its components into the database. The resulting arrays then become the initial coverage tables in the database.

This slicing method is necessary in order to capture all instances of SQL statements that will be dynamically constructed from string fragments, concatenations and function calls before being passed to the database interface. If we were not to handle these cases, a large fraction of the SQL database interactions would be missed, invalidating our database interaction coverage metrics, and it would not be possible to accurately attach server environment variables to the SQL statements that use them, invalidating our server environment variable coverage metrics. The construction of the SQL SELECT statement from fragments in a dynamic page from PhpBB 2.0 shown in Figure 7.4 is a typical example.
Table 7.1: Example AllPhpPages coverage database table, tracking page coverage

The complete schema for our dynamic analysis database, including the coverage database, is shown in Table 7.2. It is comprised of eight tables and one view. Three of the tables, AllPHPPages, AllHttpVars and AllSQL_Sources, are constructed to hold coverage information and are initialized statically by our approach. During execution of the web application, these tables are updated at each instrumentation call to track coverage of page access, server environment variable access, and database SQL statement forms respectively. Tables 7.1, 7.2 and 7.3 show examples of these coverage tables as initialized by our static instrumentation slice.

The Server Pages table (Figure 7.2) is used to keep track of access to individual pages, and is associated with the AllPHPPages table, which contains information about all of the application pages, while the other tables contain information about the HTTP variables, environment variables, cookies and database statements associated with each page, linked using the Page_Ins_ID field. We combine the information from the various tables into a single unified trace view in the Dynamic Analysis view. The AllHttpVars table is associated with the HttpVars table and holds coverage information related to server environment variables. The AllSQL_Sources is associated with the Database Interactions table and holds coverage information related to database interactions.

7.3.1 Instrumenting Web and SQL Aspects

We automatically analyze and add source code instrumentation to web application source using TXL [55], a programming language designed for manipulating and experimenting with programming language notations and features. TXL is a powerful source transformation
Table 7.2: Example AllHttpVars coverage database table, tracking server environment variables coverage

<table>
<thead>
<tr>
<th>HttpIndex</th>
<th>PageIndex</th>
<th>Assigned_Var</th>
<th>HttpName</th>
<th>HttpType</th>
<th>Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>376</td>
<td>365</td>
<td>$forum_id f</td>
<td>GET</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>377</td>
<td>365</td>
<td>$forum_id f</td>
<td>POST</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>378</td>
<td>365</td>
<td>$forum_id forum</td>
<td>GET</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>379</td>
<td>365</td>
<td>$start start</td>
<td>GET</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>380</td>
<td>365</td>
<td>$mark_read mark</td>
<td>POST</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>381</td>
<td>365</td>
<td>$mark_read mark</td>
<td>GET</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>382</td>
<td>365</td>
<td>$tracking_forums _f COOKIE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>383</td>
<td>365</td>
<td>$tracking_topics _t COOKIE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>384</td>
<td>365</td>
<td>$tracking_topics _t COOKIE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>385</td>
<td>365</td>
<td>$tracking_forums _f COOKIE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>386</td>
<td>365</td>
<td>$tracking_forums _f COOKIE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>387</td>
<td>365</td>
<td>$topic_days topicdays POST</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>388</td>
<td>365</td>
<td>$topic_days topicdays GET</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3: Example AllSqlSources coverage database table, tracking SQL statement coverage

system that has been used in industrial applications involving millions of lines of source code. The TXL processor takes as input a context-free grammar for the language to be manipulated, parses the source program into a parse tree, and then recursively applies a set of transformation rules, beginning with a main rule, until there are no remaining matches in the parse tree. The transformation is completed by unparsing the transformed tree to the new target source program.

While our process is presently targeted at PHP and MySQL, this lightweight TXL-based process is adaptable in plug-and-play fashion to deal with any scripting language and database engine as a source for transformation. Documents that include a mixture of
languages and technologies can be easily handled, usually by employing island grammars [130][160], where the interesting elements, PHP code in our case, are considered islands, and uninteresting elements, HTML code and other text in our case, are considered water. Using island grammars simplifies the parsing process as interesting elements can be identified and analyzed without parsing the whole document.

The instrumenting transformation process is used to serve two major purposes. The first is to globally mark, for further static extraction and processing, the coverage information on the level of page access, server environment variable access and database interactions, and second is to insert appropriate calls to an instrumentation coverage library developed in PHP to dynamically update the coverage database as the application under test is executed. The following subsections provide the details of this process.

**Instrumenting Page Access**

In DWASTIC, the application sources are processed statically, one page at a time. A set of rooted transformation rules is applied on each page to mark, extract and analyze coverage information. When a page is sent to DWASTIC for processing, the TXL main rule imports the page’s full path name and generates a unique page identifier in the TXL global variable `uniqePageid`, which will be used for any further analysis associated with this page. The transformation rule `instrumentPage` (Figure 7.3) is then called to add dynamic instrumentation at the top of the page. The `instrumentPage` rule constructs a block of statements in the `PageIns` variable, which includes adding a new entry to the PHP global array `$GLOBALS["PhpFileName"]` to represent the current processed page name and unique ID. A call to the coverage function `IsCovered` is added to track the page access at run time and to update the entry for this page ID in the coverage database. Other statements added to the `PageIns` block are used to assist in globally defining other PHP arrays associated with other coverage information related to database interactions and server environment variables, as well as a call to our PHP instrumentation coverage library `sensfuncDBJan92009.php`. The
**CHAPTER 7. AUTOMATING COVERAGE METRICS FOR WEB APP.**

127

```
rule instrumentPage InputFile [stringlit]
  % Get global unique page id generated by the main rule
import uniqePageid [id]
  % Transform the entire page exactly once
replace $ [Document]
  PHPO [PHPOpenTag]
  TopS [TopStatement*]
  PHPC [PHPCloseTag]
  % Insert page instrumentation and coverage code
by
  PHPO
  'global $Sql_index;
  'global $PhpFileName;
  'global $PhpFileIndex;
  'global $Http_Source;
  "$GLOBALS['"PhpFileName'"]'[uniqePageid'] = InputFile;
  "$GLOBALS['"PhpFileIndex'"]' = uniqePageid;
  'include_once('"sensfuncDBJan92009.php");
  'IsCovered '('$GLOBALS '':['"PhpFileIndex'"]');
  '
  TopS
  PHPC
end rule
```

Figure 7.3: The TXL instrumentPage rule adds page coverage instrumentation to the top of each processed page

The first few lines of Figure 7.4 show the result of adding page coverage instrumentation to the search.php page.

**Instrumenting Server Environment Variables**

Each page is also transformed by a specialized TXL rule to identify server environment variables, replacing them with an instrumentation function which collects the server environment variable's names, values, and the PHP variables which receive the values. This information is passed as parameters to the instrumentation function `HttpVar_track()`. The server environment variables are also added to the PHP user defined global array `$GLOBALS"Http_Source"` and passed as a parameter to the same function as well. When the application under test is executed and the `HttpVar_track()` function is called, the server environment variable access information is inserted in the `HttpVar` table, and the coverage count for the accessed variable is updated in the AllHttpVar coverage table. The `$search_id` assignment statement in Figure 7.4 shows an example in which `$_GET_Vars ['search_id']` has been identified,
instrumented, and added to the \$GLOBAL ["Http\_Source"] coverage global array.

Instrumenting SQL Statement Sources

Identifying, extracting, and analyzing the dynamically constructed SQL statements in the context of the overall web system is not a trivial process, and often requires a great deal of complicated analysis using robust parsing, pattern matching, and control and data flow analysis. SQL statements are often constructed inter-procedurally, using a combination of string concatenation statements and host language statements that work together to construct the text of the SQL statement. These combinations are not only constant strings, but also include SQL statement fragments and host application variables.

In our approach, the most complex set of transformation rules is used to handle this task, and is mainly composed of three parts: The first part is to identify the beginning of dynamically constructed SQL statements. We do that by distinguishing string literals that begin with the SQL keywords *Select, Insert, Update, Delete, Create, Alter* and *Drop* using a separate TXL token class, and then use the parser to recognize concatenations built from these strings. Our transformation targets assignment statements that use these strings to build larger strings. Prior to the transformations, the code is normalized, replacing string expressions in other statements with a temporary PHP variable and inserting an assignment before the statement.

Once an assignment is found that uses one of the SQL keyword strings, other assignments using the same PHP variable are also checked and instrumented. Each identified statement is followed by a newly constructed assignment statement which is aimed at constructing the SQL sub strings in a user defined PHP global array entry, specifically created by our transformation approach to hold the identified SQL strings, instead of the original application variable. Our transformation process generates a unique identifier to associate each newly identified SQL statement which will be used as index for the newly constructed string in the global array.
Figure 7.4: Coverage instrumentation added by DWASTIC to the search.php dynamic page of the PhpBB 2.0 application. Sections in boldface have been added by our instrumenting transformation to instrument coverage for pages, server environment variables and SQL statements.
In the second part, while constructing the SQL statement source from string fragments and concatenation statements, special care is given to the kind of the concatenated fragment, so that we can retain the original PHP variable names rather than their run-time values in the SQL statement text. This retains in our database the link between dynamically generated SQL statements and the variables they use. Our approach distinguishes four SQL fragment types: PHP constant variables, PHP variables, string expressions, and SQL fragment variables. The final SQL statement is constructed by single quoting all fragment types other than constant variables and SQL fragments, which are kept unquoted for later substitution in the execution phase (Section 7.3.3).

The third part is to identify and instrument the application database interaction points. At those points, the database interface call statement `mysql_query()` is replaced with a call to our instrumenting function `mysql_query_track()` as shown in Figure 7.5. The instrumenting function call takes both of the two versions of the SQL statement, the source statement collected in the previous step and available globally at this point, and the execution instance of that statement, both of which are then sent for storage at our instrumentation database table `Database Interactions`. The instrumenting function then updates the coverage database by incrementing the coverage count of the executed SQL source statement in the `ALLSQL_Sources` coverage table. Finally, it executes the original database interaction statement. Figure 7.4 shows examples of database statement source fragments identified, instrumented, and added to the `GLOBAL["SqlParts"]` coverage global array.

Our methodology can capture, instrument, and correlate SQL source statements and
CHAPTER 7. AUTOMATING COVERAGE METRICS FOR WEB APP.

% Begin with PHP grammar
include "php.grm"

% Override to isolate coverage instrumentation parts
redefine Expr [CoverageAspect]
|
...end redefine

% Custom grammar to identify coverage instrumentation parts
define CoverageAspect
'\$GLOBALS'['"Sql_Source"]'[ [Expr?] ']'['0'] '=' [SqlPartExpr];
'|'\$GLOBALS'['"Sql_Source"]'[ [Expr?] ']'['1'] '=' [Expr];
'|'\$GLOBALS'['"Sql_Source"] '[Expr?] ']'['0'] ','. [SqlPartExpr];
'|'\$GLOBALS'['"PhpFileName"] '[ [Expr?] '] '.[Expr];
'|'\$GLOBALS'['"Http_Source"] '[ [Expr?] '] '.[Expr][NL]
end define

% Allow for output of coverage aspect only
redefine program
|
|[CoverageAspect*]
end redefine

% Transform instrumented PHP program to its coverage aspect
function main
  replace * [program]
P [program]
| Use TXL grammatical type extraction to gather aspect fragments
Construct CoverageInstrumentationAspect [CoverageAspect*]
_ [^P]
by CoverageInstrumentationAspect
end function

Figure 7.6: TXL program to identify and extract the coverage aspect of an instrumented PHP program

the database interaction points even when they are spread over separate source files. There is no need to combine the source files into a single processing unit, since the relation is done using global arrays.

7.3.2 Extracting the Instrumentation Slice

Once the web application is instrumented for page access, server environment variables and databases interactions as described above, the DWASTIC tool extracts the instrumentation slice from the application based on the grammatical patterns defined in Figure 7.6. The patterns identify four kinds of assignment statements: the first three statements are aimed at identifying instrumentation generated for collecting SQL statement sources from
their fragments using assignments or concatenation statements, the fourth pattern identifies page access instrumentation statements, and the fifth pattern identifies server environment variables instrumentation expressions.

Based on the grammatical patterns, the main transformation rule shown in Figure 7.6 extracts all instances of these instrumentation statements from the application source code, and groups them into a single new file. This file is then automatically transformed into a PHP program by enclosing it in PHP opening and closing tags, including a reference to the application constants, and inserting call statements to PHP functions that insert the coverage information collected in the global arrays into the coverage database as the initial coverage tables. An elided view of the generated slice as a PHP program is shown in Figure 7.7.

7.3.3 Executing and Analyzing the Instrumentation slice

The extracted instrumentation slice PHP program constructed in the previous step uses three global arrays, one for SQL statement sources, one for server environment variables, and one for page access. When the slice program is executed, it populates the global arrays with one instance of every SQL source statement, every server environment variable access, and every page access that is instrumented in the application. This effectively builds the coverage tables of each, which are then inserted as the initial tables of the coverage database described in Figure 7.2. The SQL statement sources are analyzed upon insertion to identify the basic query components, and to decide about any server environment variable or application variable embedded in the statement and insert these details in the database as well.
<?php

$GLOBALS["PhpFileName"] = "C:\WAMP\WWW\PHPBB2\admin\admin_forums.php";
$GLOBALS["Http_Source"] = array ("mode", "POST", 467, "$mode");
$GLOBALS["Http_Source"] = array ("mode", "GET", 467, "$mode");

$GLOBALS["SqlParts"]['table'] = ((FORUMS_TABLE));
$GLOBALS["Sql_Source"] = ((('SELECT * FROM ' . $GLOBALS["SqlParts"]['table'])),
$GLOBALS["Sql_Source"] = 467;
$GLOBALS["Sql_Source"] = (('WHERE $catfield = $cat'));
$GLOBALS["Sql_Source"] = ('ORDER BY $orderfield ASC'));
$GLOBALS["Sql_Source"] = (('UPDATE ' . $GLOBALS["SqlParts"]['table'] . 
'WHERE $orderfield = $i WHERE $idfield = ') . 
'$row[$idfield]' ));
$GLOBALS["Sql_Source"] = 467;
$GLOBALS["Http_Source"] = array ("addforum", "POST", 467, '');
$GLOBALS["Http_Source"] = array ("POST_FORUM_URL", "GET", 467, '$forum_id');
$GLOBALS["Sql_Source"] = (('SELECT * FROM ' . PRUNE_TABLE). 
('WHERE forum_id = $forum_id'));
$GLOBALS["Sql_Source"] = 467;

BuildAllPHPPagesTable ($GLOBALS["PhpFileName"]);
BuildAllHttpTable ($GLOBALS["Http_Source"], $GLOBALS["$PhpFileIndex"]);
BuildAllSQLSourcesTable ($GLOBALS["Sql_Source"]);
?>

Figure 7.7: Part of the extracted instrumentation slice for PhpBB 2.0 augmented with database insertion code

7.4 Evaluation

We have evaluated the effectiveness of our approach by analyzing a production dynamic web applications PhpBB 2.0, with millions of installations, the world’s most popular internet forum system. WATIR (Web Application Testing In Ruby) [39] [169], a scriptable library to drive web browsers by clicking links, pressing buttons, and filling in forms, is used to automate the collection of usage traces.

We now show some more detailed results for two specific test cases: an anonymous user interacting with a PhpBB forum, and an Admin user visit. These two test cases are simple interactions using only the hyperlinks, not including the population of forms. The complete results are too large to include in this chapter, but the example data in the initial coverage
databases shown in Tables 7.1, 7.2, and 7.3 refer to a subset of the instrumentation points available to be covered in these tests. Using the results from the two tests, we calculate values for the metrics proposed in Section 7.2.2.

Table 7.4, shows the overall results for the Page Cov., the SQL_Stm Cov., and the Server_Env_Var Cov. metrics for each test. In the table, the Total is the total number of instrumentation points in each of the categories. For example, the anonymous user test covered 28 of the 69 total pages, while the Admin user test covered 56 of the 69 total pages. Since the two example test cases do not include forms, neither all SQL statements nor all environment variables are covered. We have manually confirmed that the Total number of each coverage aspect extracted by our approach is equal to the actual number of aspects in the code.

Table 7.5, shows the results for the Page Server_Env_Var Cov., the Page SQL_Stm Cov. metrics for two pages, Viewforum.php and Viewtopic.php. Even without form filling, the Admin user covers more SQL statements and more variables because he can access more links. Some examples are the new topic and post reply links.

Table 7.6, shows the results for the SQL Server_Env_Var Cov. metric for the same two pages. This metric measures the number of SQL statements that reference server environment variables. Comparing Table 7.5 and Table 7.6, we find that the page Viewtopic.php has 12 out of 13 SQL statements that reference server variables. The admin user test cases covers 5 of them. This measure is essential for evaluating test cases for SQL statements that use user input such as test cases for SQL injection.

Since the run-time instrumentation must update the coverage database, it imposes a runtime penalty on the web application. Table 7.7 shows the runtime measure for 17 pages. The second column (Non Instrumented Exec Time) indicates the time needed to execute the original, uninstrumented page. The third column (Instrumented Exec Time) shows the time needed to execute the instrumented page. The performance penalty of the instrumentation ranges from non-existent (faq.php) up to about 50%. The average penalty is 32%, which is
Willmor and Embury [171] propose two test adequacy criteria for database applications. The first criterion checks the coverage of the structural aspects of the database application. This includes aspects such as the different types of operations, transaction statements, and the entities represented in the database. The other criterion is a Define-use criterion,

<table>
<thead>
<tr>
<th>Role</th>
<th>SQL Statements Coverage</th>
<th>Page Access Coverage</th>
<th>Server Environment Variables Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous User</td>
<td>Covered: 41 Total: 440 %9.3%</td>
<td>Covered: 28 Total: 69 %40.0%</td>
<td>Covered: 21 Total: 374 %5.6%</td>
</tr>
<tr>
<td>Admin</td>
<td>Covered: 123 Total: 440 28%</td>
<td>Covered: 56 Total: 69 81%</td>
<td>Covered: 68 Total: 374 18.2%</td>
</tr>
</tbody>
</table>

Table 7.4: Coverage metrics results for pages, server environment variables and SQL statements at the application level for a sample test case

<table>
<thead>
<tr>
<th>Role</th>
<th>Page Name</th>
<th>SQL Statements Coverage</th>
<th>Server Environment Variables Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous User</td>
<td>Viewforum.php</td>
<td>Covered: 5 Total: 7 71.42%</td>
<td>Covered: 1 Total: 12 8.33%</td>
</tr>
<tr>
<td>Viewtopic.php</td>
<td>Covered: 4 Total: 13 30.76%</td>
<td>Covered: 1 Total: 16 6.25%</td>
<td></td>
</tr>
<tr>
<td>Admin</td>
<td>Viewforum.php</td>
<td>Covered: 6 Total: 7 85.7%</td>
<td>Covered: 5 Total: 12 41.6%</td>
</tr>
<tr>
<td>Viewtopic.php</td>
<td>Covered: 6 Total: 13 46.2%</td>
<td>Covered: 5 Total: 16 31.2%</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.5: Coverage metrics results for server environment variables and SQL statements at the page level for a sample test case

acceptable for a testing environment.

7.5 Related Work

Several approaches and coverage metrics have been proposed to assess the quality of test cases aimed at ensuring the correctness of database interactions in standard database systems, for instance, Suárez-Cabal and Tuya [37] propose a coverage metric and a tool specialized for a subset of SQL SELECT statement in order to help improve test suites to detect faults on the level of the isolated SQL SELECT statements in database application. A coverage tree is built out from the SELECT Query conditions, where each path on the tree corresponds to conditions that exist in the where or join clause of the query. The approach is aimed at analyzing static SQL Select statements, while ours handles all SQL statements including dynamically constructed ones.

Willmor and Embury [171] propose two test adequacy criteria for database applications. The first criterion checks the coverage of the structural aspects of the database application.
CHAPTER 7. AUTOMATING COVERAGE METRICS FOR WEB APP.

<table>
<thead>
<tr>
<th>Role</th>
<th>Page Name</th>
<th>Server Environment Variables</th>
<th>Coverage in SQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous User</td>
<td>Viewforum.php</td>
<td>Covered 5</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Viewtopic.php</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Admin</td>
<td>Viewforum.php</td>
<td>Covered 6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Viewtopic.php</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 7.6: Coverage metrics results for pages and server environment variables at the SQL statement level for a sample test case

<table>
<thead>
<tr>
<th>Page Name</th>
<th>Uninstrumented Exec. Time (sec.)</th>
<th>Instrumented Exec. Time (sec.)</th>
<th>Performance degradation (percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://localhost/phpBB2/index.php">http://localhost/phpBB2/index.php</a></td>
<td>2.172</td>
<td>3.11</td>
<td>30.2%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/faq.php">http://localhost/phpBB2/faq.php</a></td>
<td>1.328</td>
<td>1.328</td>
<td>0.0%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/search.php">http://localhost/phpBB2/search.php</a></td>
<td>2.14</td>
<td>3.15</td>
<td>34.0%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/memberlist.php">http://localhost/phpBB2/memberlist.php</a></td>
<td>0.953</td>
<td>1.485</td>
<td>35.8%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/groupcp.php">http://localhost/phpBB2/groupcp.php</a></td>
<td>1.016</td>
<td>1.5</td>
<td>32.3%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/profile.php?mode=register">http://localhost/phpBB2/profile.php?mode=register</a></td>
<td>2.14</td>
<td>3.156</td>
<td>32.2%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/profile.php?mode=editprofile">http://localhost/phpBB2/profile.php?mode=editprofile</a></td>
<td>2.031</td>
<td>3</td>
<td>32.3%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/privmsg.php?folder=inbox">http://localhost/phpBB2/privmsg.php?folder=inbox</a></td>
<td>2.172</td>
<td>3.11</td>
<td>30.2%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/login.php">http://localhost/phpBB2/login.php</a></td>
<td>2.984</td>
<td>3.284</td>
<td>33.0%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/index.php">http://localhost/phpBB2/index.php</a></td>
<td>1.344</td>
<td>1.828</td>
<td>26.5%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/search.php?search_id=unanswered">http://localhost/phpBB2/search.php?search_id=unanswered</a></td>
<td>2.188</td>
<td>3.188</td>
<td>31.4%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/index.php?c=1">http://localhost/phpBB2/index.php?c=1</a></td>
<td>1.328</td>
<td>1.796</td>
<td>26.1%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/viewforum.php?f=1">http://localhost/phpBB2/viewforum.php?f=1</a></td>
<td>1.453</td>
<td>2</td>
<td>27.4%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/profile.php?mode=viewprofile&amp;u=2">http://localhost/phpBB2/profile.php?mode=viewprofile&amp;u=2</a></td>
<td>1.016</td>
<td>1.984</td>
<td>48.8%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/viewonline.php">http://localhost/phpBB2/viewonline.php</a></td>
<td>0.953</td>
<td>1.484</td>
<td>35.8%</td>
</tr>
<tr>
<td><a href="http://localhost/phpBB2/profile.php?mode=viewprofile&amp;u=2">http://localhost/phpBB2/profile.php?mode=viewprofile&amp;u=2</a></td>
<td>1.016</td>
<td>1.984</td>
<td>48.8%</td>
</tr>
</tbody>
</table>

Table 7.7: Performance penalty of DWASTIC instrumentation on dynamic pages of PhpBB 2.0

which measures all of the possible database system operations define-use pairs. The metrics proposed by Willmor and Embury are more comprehensive than Suárez-Cabal and Tuya metrics, but both target traditional database applications and neither has evaluated the effectiveness of their proposed metrics using a working tool. The coverage information collected in our approach is sufficient to compute the coverage metrics proposed by Willmor and Embury.

Halfond and Orso [93] propose a coverage criterion which measures the coverage of all the possible SQL command forms that can be issued at each database interaction point.
They describe a prototype tool DITTO which statically analyzes the application source code, identifying database interaction points and the string variables containing the SQL commands.

Java String Analysis (JVS) is used to build a character-based NDFA for each string variable, which is converted into an SQL-level NDFA. The SQL NDFA represents a static model of all SQL queries that can be generated. Their metric compares the number of forms covered by the test cases to the total number of forms possible at each interaction point. Our approach constructs exact versions of the SQL statement templates constructed between define-use paths for a web application. It is easily extendable to web applications implemented in any technology, while Halfond and Orso’s is limited to Java applications and has limitations when collecting SQL statement fragments from external sources.

Smith et al. [156] propose two coverage metrics for SQL injection vulnerability testing. The first metric measures the percentage of database interaction points (API calls) that are tested at least once to the total number of identified database interactions points. The second coverage metric measures the percentage of input variables tested at least once to the total number of variables found in any target SQL statement. The database interaction points and input variables are counted manually, and the instrumentation process is also done manually. As mentioned earlier, their first metric does not consider all dynamically constructed SQL statements. Also, our proposed coverage criteria not only cover input variables used in SQL statements on the application level, but also at the page level, and our instrumentation is automated using source transformations.

There are other approaches that are similar to ours, identifying, extracting and analyzing database interactions. Cleve and Hainaut [51] use aspect-based tracing to relate and extract the basic components of prepared statements. While the tracing approach used does not modify the source code, it does not deal with the dynamically constructed SQL statements using string concatenations scattered throughout the code that our slicing resolves. Their approach is also yet to be evaluated on a production system.
Brink et al. [165] propose a tool for assessing the quality of database interactions in standard applications. They extract embedded SQL statements using control and dataflow analysis. The identification of SQL string literals are done using a standard Java program that tokenizes the source program based on predefined SDF grammars. The purpose is to extract the queries for quality assessment, while our purpose is to determine the coverage of test cases.

Ngo and Tan [135] propose an automatic static technique to extract database interaction points from web applications. The approach first identifies all program paths that include a database interaction and then slices them out as an Interaction Control Flow Graph (ICFG). Each interaction path is then symbolically executed, and all possible interaction types are derived from the generated symbolic expression using inference rules. Evaluating the approach on a case study, the approach is able to extract 80% of the database interactions. The complexity of the extraction process is high, as it is composed of five stages, and is affected by factors such as number of the interaction paths (i-paths), and the length and complexity of each ipath. The authors also do not specify how to handle SQL statements constructed from sequences of string fragments and concatenations, which is handled by our instrumentation slice technique.

7.6 Conclusion

In this chapter, an original approach to automate coverage metrics for dynamic web application has been proposed, implemented and evaluated. First, we proposed a set of coverage criteria specialized for web application, which takes into consideration the complex and distributed structure of this application. We demonstrated how a dynamic web application written in PHP could be automatically instrumented using source transformations, and that database SQL statements dynamically constructed from string fragments could be handled using a source slicing technique to identify and build a coverage database. The
proposed coverage criteria and the automatic tool helps improve the quality of test cases and focus the testing efforts towards the application component interactions, which are usually the source of many web applications vulnerabilities. The approach is being used to provide a completeness measure for extracting an access control security model from a web application under test. The accuracy of the results is both hand verified and robust since it is automatically back-checked at run time.

In the next chapter we will elaborate in more detail the role of DWASTIC, as well as all the tools discussed in Chapters 4, 5 and 6, in our approach for RBAC security model recovery.
Chapter 8

Recovering Role-Based Access Control Security Models from Dynamic Web Applications

In this chapter we present an approach to automatically construct a role-based access control security model from the recovered structural and behavioural models presented in Chapter 5 and 6. We use TXL, a source transformation technology, to implement the automatic model-to-model transformation and composition. The generated model is represented in the UML 2.1 exchange format, XMI 2.1.

Based on the model-to-model transformation approach, we have also developed a tool to automatically transform the semi-formal UML 2.1 security model into a formal one to ease the process of verifying the system against security properties.

This chapter is organized as follows: We first motivate the need for a new methodology and approach in Section 8.1. We give a high-level overview of the approach in Section 8.2, and review how the other framework components contribute to the process of recovering the security model in Sections 8.3, 8.4, and 8.5. Section 8.6 presents our approach to analyzing
the resulting security model, and Section 8.7 places our method in the context of other work. Finally, Section 8.8 concludes the chapter and presents possible future work.

8.1 Motivation

According to the Model Driven Architecture (MDA) approach, the application specification and development phases can be expressed using models, and transformation functions are used to map between models as well as to automatically generate executable code [81]. Model transformation is of particular importance in this process. It maps source models into new target models, or relates the source models to existing ones, thus enabling model integration, platform-independent (PIM) to platform-specific model (PSM) mapping, reverse engineering, and platform migration. Transformations can be defined in terms of meta-models of languages as well.

In this chapter, we apply source transformation technology using TXL to create a process of model analysis and integration to construct the target security model. TXL [55] is a programming language designed for manipulating and experimenting with programming language notations and features. It has been used in many production applications with transformations involving billions of lines of source code.

Even though TXL has been primarily used for the transformation of languages and design recovery, a new promising direction of bringing the transformation power of TXL to the modeling world has recently been investigated [141, 119]. Researchers have been attracted by the efficiency, robustness, and platform independence provided by this language. Such transformations are applicable to large models, and even heterogeneous ones that integrate components in a variety of languages.

Although the model transformation process accepts models as an input and generates models as an output, and each of those conforms to a specific meta-model and reflects a specific view of the system, TXL can handle the transformation process between source and
target models as long as they can be serialized into a text-based format. Fortunately, this can be easily done by most of modeling tools, including ArgoUML and RSA, using XMI export and import facility.

While modeling tools can handle different versions of XMI, making the process of model transformation and model interchange between them a challenge, TXL’s robust grammar easily adapts these variations. TXL grammars can be adapted to accept and manipulate a wide range of XMI versions, and can generate multiple versions of the serialized models to match a wide range of modeling tools.

8.2 Approach

Our approach comprises two main phases: the first phase uses an automated role mining process for dynamic web applications, 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, and pattern matching to identify the RBAC security elements. In the previous chapters, we have developed four approaches and tools to support this phase, and we provide a brief description of them here to better understand their role.
**CHAPTER 8. RECOVERING RBAC SECURITY MODELS FROM WEB APP.**

<table>
<thead>
<tr>
<th>Application Aspects</th>
<th>Ref No</th>
<th>XMI Tag</th>
<th>XMI Type</th>
<th>XMI ID</th>
<th>XMI additional Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application behavioral modeling</td>
<td>1</td>
<td>packagedElement</td>
<td>uml:Collaboration</td>
<td>CollabID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>ownedBehavior</td>
<td>uml:Interaction</td>
<td>InteractID</td>
<td></td>
</tr>
<tr>
<td>Application Entities</td>
<td>1.1.1</td>
<td>lifeline</td>
<td>uml:Lifeline</td>
<td>LifelineID</td>
<td>name=Entity Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>represents=1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>coveredBy=1.1.2,1.1.3,1.1.4</td>
</tr>
<tr>
<td>Entities Represented</td>
<td>1.1.2</td>
<td>fragment</td>
<td>uml:MessageOccurrenceSpecification</td>
<td>MOccID</td>
<td>Covered=1.1.1</td>
</tr>
<tr>
<td>As send and receipt of messages during a specific period of time</td>
<td></td>
<td></td>
<td></td>
<td>Event=3 or 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Message=1.1.4</td>
</tr>
<tr>
<td></td>
<td>1.1.3</td>
<td>fragment</td>
<td>uml:BehaviorExecutionSpecification</td>
<td>BehOccID</td>
<td>Covered=1.1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Start=1.1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Finish=1.1.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.1.4</td>
<td>fragment</td>
<td>uml:ExecutionOccurrenceSpecification</td>
<td>ExecOccID</td>
<td>Covered=1.1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Event=5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Execution=1.1.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.1.5</td>
<td>message</td>
<td>uml:Message</td>
<td>MessID</td>
<td>Name= constructed according to Table2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>messageSort=SynchCall</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>receiveEvent=1.1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sendEvent=1.1.2</td>
</tr>
<tr>
<td>Actual interaction is represented as classes with operations as well.</td>
<td>1.2</td>
<td>ownedAttribute</td>
<td>uml:Property</td>
<td>ProbID</td>
<td>Name= EntityName</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type=2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>packagedElement</td>
<td>uml:Class</td>
<td>ClassID</td>
<td>Name= EntityName</td>
</tr>
<tr>
<td></td>
<td>2.1</td>
<td>ownedOperation</td>
<td>uml:Operation</td>
<td>OpID</td>
<td>Name= same as message name</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>packagedElement</td>
<td>uml:SendOperationEvent</td>
<td>SOpID</td>
<td>Name= Any Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Operation=2.1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>packagedElement</td>
<td>uml:ReceiveOperationEvent</td>
<td>ROpID</td>
<td>Name= Any Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Operation=2.1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>packagedElement</td>
<td>uml:ExecutionEvent</td>
<td>ExecEvID</td>
<td>Name= Any Name</td>
</tr>
</tbody>
</table>

Figure 8.2: Mappings between Database trace model and UML SD meta-model

in the security model recovery process.

The second phase provides an automated model driven engineering approach to construct a RBAC security model. This phase is based on model transformation and composition, and makes use of the structural and behavioral models recovered in the previous phase. Figure 8.1 shows the architecture of our approach, and the following sections elaborate on the approach in more detail.
8.3 Structural Model Recovery

In this phase we aim at recovering a data model of the application resources, as they pertain to the users’ functional use of systems, applications and business processes. Data models constitute one of the main sources of such information, and visualizing data models facilitates the process of understanding the structure of the system, its basic entities and their relationships. While UML is considered the standard for application modeling, there is really no corresponding open standard for data modeling. Therefore, we have developed an approach and a tool to bring the data model to the UML world, and thus our tool enables the manipulation and integration of both data and application models using the same UML-based tools in an interoperable way.

Our open tool (SQL2XMI), presented in Chapter 4, automatically transforms an SQL DDL schema to a UML 2.1 ER diagram which can be visualized by any UML tool that supports XMI 2.1. Unlike other tools that reverse engineer to proprietary formats, SQL2XMI explicitly aims at open and flexible portability, requiring only the SQL DDL schema and targeting the official OMG XMI 2.1 UML representation. Figure 8.5, shows the meta-model for the UML-ER data model.
CHAPTER 8. RECOVERING RBAC SECURITY MODELS FROM WEB APP.

Figure 8.4: A sample UML2.0 Entity-level Sequence Diagram (SD) as generated by WAPA and

PHP2XM.
8.4 Behavioural Model Recovery

Exploring existing user permissions in current applications and systems is essentially one of the critical steps in the role mining process. It requires a complex set of analyses techniques including static analysis, pattern matching, data and control flow analysis, as well as coverage analysis. Thus, we have developed (PHP2XMI), presented in Chapter 5, a new reverse engineering tool aimed at recovering the permissions associated with each user role and representing them as a behavioural model expressed as a UML sequence diagram. Visualizing execution traces as a sequence diagram facilitates the process of understanding the interaction behavior of the system, and helps us deduce the permissions for each user role. The XMI 2.1 textual representation of the sequence model is analyzed and combined with the XMI 2.1 representation of the ER model to construct a UML-based RBAC model, which can be converted into a formal model to be checked for access control vulnerabilities using a standard model checker. Thus, the PHP2XMI is an essential part of a our web application RBAC conformance-testing framework.

The approach we use filters execution traces directly on insertion into the database, automatically eliminating redundant information that may complicate the understanding process. In PHP2XMI, the interaction elements in the resulting sequence diagram are the user and the dynamic pages of a browser session, represented as lifelines, and the dynamic transitions between the pages along with their parameters, represented as messages.

To help raise the recovered behavioural model into an entity-based sequence diagram, we have developed (WAFA), presented in Chapter 6, an automated reverse engineering approach to recover a fine-grained interaction behavioural model from dynamic web applications. To the best of our knowledge, our approach is the first one to extract the web application’s embedded SQL subsystem, which includes both the original SQL statement source as well as corresponding execution instances, and an analysis to attach it to both static host application variables and dynamic server environment variables.
Based on the fine-grained analysis performed by WAFA, which populates our instrumentation database with trace information, we automatically generate sequence diagrams that conform to the UML2.1 meta-model, shown in Figure 8.5. We have defined mappings between elements of the trace model stored in the instrumentation database and corresponding UML 2.0 sequence diagram. The mappings better reflect our interest in capturing user interactions with the dynamic web application as they affect the multiple tier of this distributed system, such as the browser, the application server and the database back-end, Figure 8.2 shows the mapping between the web application components and the UML 2.0 sequence diagram meta-model elements.

These interactions, when comprehensively covered, represent the set of permissions the dynamic web application offers the users. To this end, the sequence diagram lifelines are used to map the application’s secured resources, which are comprised of the application’s pages as rendered in the browser, the application’s server environment, and the application’s entities as stored in the application’s database back-end.

The diagram’s operations are used to map the different type of access allowable to the user over the application’s secured resources. All aspects of this access are captured as either operations’ parameters or constraints, and these include the access’s type, timestamp, condition, return value, and unique id to identify the access’s relation with the source code and the source page. Messages are used to present a combination of those values in a single string which constitutes the message name. Figure 8.3 shows the different types of messages constructed by our approach.

8.4.1 Instrumentation Coverage

Dynamic analysis often needs to be combined with a coverage measure in order to decrease the number of false positives due to an analysis that yields a model that only partially covers the code (leading to verification of properties that may in fact not hold). For this purpose, we have developed (DWASTIC), presented in Chapter 7, a tool used to augment
the dynamic analysis with instrumentation for code coverage. The tool is based on our newly proposed set of coverage criteria specialized for web application, which takes into consideration the complex and distributed structure of this application. The tool can be used to support any testing activity for web applications. It focuses the testing efforts on the vulnerable parts of the code, which are most likely the source of web application faults and attacks such as SQL injection. It also provides a direct way to trace the part of code that is not covered by test cases. The approach is being used to provide a completeness measure for extracting an access control security model form web application under test. The accuracy of the results is both hand verified and robust since it is automatically back-checked at run time.

8.5 SecureUML Model Construction

In the previous sections, we have discussed how SQL2XMI, WAFA and PHP2XMI help us to recover the behavioural and structural models from web applications, and how DWASTIC provides a measure for an acceptance rate of coverage for some representative users. At this stage, our aim is to make use of the relevant elements from the two recovered models to construct a RBAC security model that conforms to the SecureUML meta-model, shown on Figure 8.5.

SecureUML is an implementation of the Model Driven Security approach, 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 the UML, and the textual language used to specify the security models. Therefore, it is built on a modular schema that comprises three basic elements: a language for security policies specification, a language for design models construction, and a dialect for defining integration points in the preceding languages.
The abstract syntax for the security language (SecureUML) is based on RBAC. It defines a meta-model that extends the 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 the values of the system’s environment variables.

The modeling notation for SecureUML is based on a UML profile, which uses UML stereotypes and tagged values to represent the abstract syntax elements expressed via the meta-model schema. Users, groups, and roles are represented as classes with stereotypes <<User>>, <<Group>>, <<Role>> respectively. Permission is represented as an
function main

%import the name of the sequence diagram serialization from the Txl command arguments
import TXLargs [repeat stringlit]
  Input2Filename [stringlit]

%read the file content and parse it to check conformance with SD meta-model
construct BehavioralModelXMI [repeat XMItoken]
  _ [read Input2Filename]

%bound the main replacement with the ER diagram serialization
replace [program]
  ERModelXMI [repeat XMItoken]

% Collect all the entities involved in the behavioral interaction
construct collectAllEntites [repeat XMItoken]
  BehavioralModelXMI [collect_AllEntites]
import EntitiesName %[id*]

% filters the collected entities by removing repeating caused by views structure
construct UniqeEntitesName %[id*]
  EntitiesName [removeRepetitions]
import UniqeEntsName%[id*]

%extracts the behavioral interactions performed on the application entities
construct secureUML [repeat XMItoken]
  BehavioralModelXMI [GenerateERDElements ERModelXMI]
import Entitybehavior2 [repeat XMItoken]

by

% combine results to construct the XMI2.1 SecureUML model
_[createXMIH]. Entitybehavior2[/createXMIE]

end function

Figure 8.6: TXL main rule for PHP2SecureUML

association class with a <<permission>> stereotype.

Figure 8.5 shows the relationship between the UML-sequence diagram meta-model and our UML 2.0 data meta-model. The entity element in the UML data meta-model corresponds to the class element of the sequence diagram meta-model via a stereotype relationship, represented as dotted line in the figure. Based on this relation, the structural information for each entity in the sequence diagram can be pulled from the data model.

8.5.1 Entity Extraction and Filtration

The set of classes (Entities) in the sequence diagram is the actual representation of the diagram’s lifelines and maps the application’s secure resources, which include: application server, browser session, and database-backend entities. Based on source transformation
technology, in this step these elements are identified and filtered to remove any redundancies.

We have developed a TXL union grammar for XMI schemas, which enables the manipulation of models that conform to the UML sequence diagram (SD), UML-based ER diagram and SecureUML meta-models. Along with this grammar, the TXL process will accept as input a serialization for both SD and ER models, as well as a set of rooted transformation rules to enable the model’s manipulation, integration and transformation to help construct the target security model.

The main TXL rule, shown in Figure 8.6, starts by searching for the set of secure resources that are engaged in the interaction behavior modeled via SD. These elements are represented abstractly as a set of classes, and graphically as a set of lifelines. Thus, the TXL rule matches all SD classes’ elements and filters any redundant ones.

The redundancy occurs due to the fact that multiple secured resources which receive the same set of actions have been represented as a single class, and modeled via a single Lifeline. Yet the names of those resources have been combined in a single string which represents the class name. Hence, the TXL rule performs a process of refactoring the combined string to identify the name of all secure resources.

8.5.2 Entity Attribute and Relation Matching and Extraction

Once the set of secure resource elements engaged in interaction behavior has been identified, the main TXL rule applies another rule on each of the identified elements. This subrule consults the UML ER diagram to search for structural information relevant to those elements. This includes the secure resource attributes and relations with other resources.

Conceptually, the TXL rule searches for all class elements with entities stereotype in the ER model that match the ones identified from the previous section. It extracts the entities’ attributes elements, and associations with other entities in the identified set. The result of this phase is an ER diagram for secure resources engaged in interactions within a specified browsing session.
8.5.3 Constrained Event Identification and Extraction

The set of permissions allowed on each of the recovered entities (resources) is modeled graphically as message receive events of the corresponding lifeline. Each recipient event element in the sequence diagram meta-model is represented as an operation which may be associated with parameters and constraints. A TXL rule, which receives as a parameter the set of recovered resources, matches the elements of the serialized SD, and whenever a class with the same resource name is matched, the set of all operations elements associated with that class along with their parameters and constraints are identified and extracted.

The rule then constructs the meta-model elements of SecureUML to represent the recovered permissions. Each operation element and its parameters is mapped to a permission action, and operation constraints are mapped to Authorization constraints. The rule constructs an association class to represent the set of recovered operations on a specific resource. The Association class is marked as a permission stereotype to reflect its security semantics.

8.5.4 SecureUML Model Elements Construction

The previous steps have identified all the security elements necessary to construct the RBAC security model. In this step, a UML 2.0 RBAC security model which conforms to the SecureUML meta-model shown in Figure 8.5, is constructed. A set of transformation rules are developed to construct the new security model, in which the extracted sequence diagram’s operations are mapped into permissions, operation constraint into Authorization constraint, and the ER data model as resources.

According to the SecureUML notation, the representation of resources is left open, so that developers can decide later which elements of the system they consider secure and upon which they want to apply access constraints. These elements are defined via a dialect. In section 8.5.2, we have recovered secure-resources, and represented them as an ER diagram. Also in section 8.5.3, we have recovered permissions-actions with authorization constraints,
CHAPTER 8. RECOVERING RBAC SECURITY MODELS FROM WEB APP.

Figure 8.7: RSA project explorer shows the SecurUML elements as recovered by our tool and represented them as an association class with parameters and preconditions. In this part, using another TXL rule, two association links are also created; one of them connects
CHAPTER 8. RECOVERING RBAC SECURITY MODELS FROM WEB APP.

Figure 8.8: A sample of a generated SecureUML model instance
include "XMI.grm"
include "prolog.grm"
redefine program
    [repeat XMItoken]
    [repeat Factdot]
end redefine
function main
    replace [program]
        SecureUML [repeat XMItoken]
        PrologFacts [repeat Factdot]
    by
        PrologFacts [GenerateEntitiesFacts SecureUML]
end function

Figure 8.9: TXL main rule for SecureUML2Prolog

the association class, stereotyped by permissions, with the entity (resource) affected by
the permission’s actions, and the second association-link connects the acting Role with the
constructed association class.

8.6 Security Model Analysis

In this phase, we use Prolog to analyze the recovered SecureUML models. Although there
are several available tools to transform from UML models into formal ones (e.g., UML2Alloy
[33], XMI2SMV [40]), thus providing model analysis and formal verification, there are none
that are both complete (supporting all the SecureUML meta-model mappings), and UML
2.0 compliant.

We have chosen Prolog as an analysis engine for its ability to handle large scale industrial
software models, as shown by Störrle [159]. In another recent study, Opoka et al. [47] have
shown that Prolog performance in model querying and analysis is higher than that provided
by the Object Constraint Language (OCL), which is originally designed for expressing con-
straints about a UML model. Therefore, Prolog allows for the implementation of a generic
and powerful querying approach for software models. Prolog syntax is less complicated than
the OCL and thus easier to understand, and the language is supported by many efficient
interpreters and tools such as the SWI Prolog. For a thorough comparison with other tools
function IsPermAction ownedAttrib [XMItoken*] EntityName [charlit]

deconstruct ownedAttrib
    'ownedOperation' 'xml:type'="uml:Operation" 'xml:id' = AttribID [stringlit]
    'name' = OpName [stringlit] 'precondition' = preconID [stringlit]'>
    ownedAttrib1 [XMItoken*]

construct idOpName [id]
    _[unquote OpName]

construct charOpName [charlit]
    _[quote idOpName]

construct PermActionF [Factdot]
    'permActionAssign' '{EntityName', charOpName'}'.

construct PermActionconst [Factdot*]
    _[IsPermActionConst ownedAttrib1 EntityName ]

replace * [Factdot*]
    by
    PermActionF

end function

Figure 8.10: A sample TXL rule for generating permission facts from SecureUML meta-model elements

and approaches for UML model analysis and checking one can refer to Störrle [159].

Thus, we have developed a SecureUML2Prolog, a tool by which a serialized SecureUML model is automatically transformed into Prolog. A set of mappings between each SecureUML meta-model constructs and Prolog Facts and Rules is defined, then, a set of Prolog queries is built to check for some specific RBAC security concerns in the target application. In our application we began by defining a grammar for the input SecureUML serialization, and a second (override) grammar for our target Prolog format.

The main grammatical form (called [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 is started by the main rule, shown in Figure 8.9. TXL begins by parsing the SecureUML serialization, to insure its conformance with its meta-model (pattern variable SecureUML), with an initially empty Prolog output (pattern variable PrologFacts).
<table>
<thead>
<tr>
<th>Structural Facts { ER diagram based resources}</th>
<th>Facts Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity[1,'phpbb_forums'].</td>
<td>entity[ RoleNum , ResourceID]: represents the applications secure resources. Gets as parameter the resource name.</td>
</tr>
<tr>
<td>entity[1, 'phpbb_auth_access'].</td>
<td>entityAttrib[ RoleNum , EntityName, Attribute]: represents the application resources attributes. Gets as parameter, the resource name and its attributes names.</td>
</tr>
<tr>
<td>entityAttrib[1, 'phpbb_users', 'username'].</td>
<td>entityAttrib[ RoleNum , EntityName, Attribute]: represents the application resources attributes. Gets as parameter, the resource name and its attributes names.</td>
</tr>
<tr>
<td>entityAssociation[1, 'phpbb_forums', 'phpbb_auth_access', 'phpbb_auth_access(forum_id)'].</td>
<td>entityAssociation[ RoleNum , entityName, AssociationID1, AssociationID2(PK)]: represents relations between the resource applications. Gets as parameter, resource name, Relation’s ends names including key attribute(s).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Role Facts</th>
<th>Facts to represent the application roles and user assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>appRole[1,'Admin'].</td>
<td>appRole(RoleNum, RoleName): represent the application roles. Get as parameter the name of the role.</td>
</tr>
<tr>
<td>appRole[2, 'Anonymous'].</td>
<td>userAssign[RoleName, RoleNum]: represents the assignment of users into roles. Gets as parameter the roles and user names.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Permissions Facts</th>
<th>Facts to represent the application permissions, constraints and permission assignments.</th>
</tr>
</thead>
<tbody>
<tr>
<td>perm[1, 'Application serverPerm'].</td>
<td>Perm[ RoleNum , PermissionID]: represents the application permissions. Gets as parameter the application permissions ID.</td>
</tr>
<tr>
<td>permActionAssign[1, 'phpbb_forumsPerm', 'Select(viewforum, forum_id = $forum_id)'].</td>
<td>permActionAssign[ RoleNum , PermissionID, Action]: represents the application permission-action assignments. Get as parameter, the permission ID, and the permission atomic actions.</td>
</tr>
<tr>
<td>permResourceAssign[1, 'phpbb_forumsPerm', 'phpbb_topicsPerm'].</td>
<td>permActionAssign[ RoleNum , PermissionID, Action]: represents the application permission-resource assignments. Gets as parameter the permission ID and the role ID.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic and contextual Facts</th>
<th>Facts to represents environmental information such as executions timestamps, relevant server environment variables and so on.</th>
</tr>
</thead>
<tbody>
<tr>
<td>permActionTime[1, 'phpbb_forumsPerm', 'phpbb_topicsPerm', 366, 1249518435].</td>
<td>permActionTime[ RoleNum , PermissionID, Action, Timestamp]: represents the application Action timestamp. Get as parameter the permission ID, the Action ID and the Actions’ execution time.</td>
</tr>
<tr>
<td>HttpVarID[1, 'Y', 'S4', 'GETID'].</td>
<td>Facts for the server environment variables: relates variable names to their type (VarType), value (VarValue), location in code (PageID), and the name of the related PHP variable (PhpVar).</td>
</tr>
</tbody>
</table>

Figure 8.11: Prolog Facts generated for SecureUML model analysis
The main rule replaces this entire input by constructing the Prolog output beginning with the empty PrologFacts and transforming it into the Prolog representation of the input SecureUML model starting with a main transformation subrule GenerateEntitesFacts. Figure 8.10 shows a rule for transforming an SecureUML permission’s action with condition into the permActionAssign Prolog fact. Also, the construct PermActionconst generates a Prolog fact for the action constraint. Figure 8.11 shows the resulting facts generated by our tool.

8.7 Related Work

In recent years, language-based security has become an active area of research as it leverages well established techniques provided by the program-analysis research community to automatically detect security problems, a challenging task facing the security and privacy research community. For access control, researchers have focused their efforts on solving three basic challenges: to identify an application’s role requirement, to detect if an application’s RBAC policy is restrictive or permissive, and to determine if an application’s RBAC policy preserve consistent access to data.

Letarte and Merlo [118] use static analysis to extract a simple role model from PHP code, and more specifically for database statements. The approach then checks if the recovered model is restrictive or permissive for binary roles. The change of authorization level in the code is modeled as an inter-procedural control flow graph with three type of edges: positive-authorization represents the change in the security privilege to admin level, negative-authorization represents the change to user level, and generic represents no change in the security level. A predefined authorization pattern is used to identify the transfer of control in the code and the change in the authorization level in the extracted model. The approach is simple as it accounts only for two roles (admin vs. user) for which access may or may not be granted to database statements. The model is entirely based on an
authorization pattern which is application dependent, and lacks any correspondence to the source code.

Koved et al. [116] propose an approach to automatically compute the access rights requirements at each program point in Java2.0 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, which represents the authorization model of the code. Consequently, this enables the identification of classes in each path that contain a call to the java 2 security authorization subsystems.

Pistoia et al. [144] propose an approach to detect inconsistencies in an application’s RBAC policy, and more specifically, any potential insufficiency or redundancy in the policy. They statically constructed a call graph to represent the flow of authorization in an application, by over approximating methods-calls in the application and identifying access-restricted methods. The graph forms the basis for several security analysis including detecting if the application’s RBAC security policy is restrictive or permissive. Then, the authors generate reports on code locations that have such inconsistencies and suggests a replacement policy which can eliminate the vulnerabilities. The approach is implemented as a part of IBM’s Enterprise Security Policy Evaluator and evaluated on a number of Java EE applications.

Centonze et al. [42] have identified the need for checking the consistency of data and resources when two methods access the same data in the same mode, but are not restricted to the same role. They have introduced the concept of location-based RBAC policy and correlate it to method-based RBAC via a location consistency property. Their aim is to identify the implicit access of methods on data and to ensure the equivalence of access rights on locations when either the location- or method- based RBAC policy have been employed. The authors have implemented this approach in a tool called SAVE.

Mendling et al. [125] propose a meta-model based integration approach to enhance the security features of Business Process Management Systems that operate via Web Services
(BPEL). The meta-model elements of web services’ BPEL, the defacto standard of web services business process composition, are mapped into RBAC elements. Roles and partners in BPEL, which represent the sets of operations that are carried out during a business process, are mapped into RBAC roles. Activities, which provide a channel for an external party to send a message to a BPEL, are mapped into RBAC permissions. The authors developed an XSTL transformation script to extract roles and permissions from a BPEL process definition based on the proposed mapping. The extracted information is stored in XML, which can be imported into the tool xoRBAC that enables the definition and enforcement of RBAC polices and constraints for web services.

In the area of web applications, many model-based testing techniques have been proposed; however, the main focus is on testing structural, dynamic, or interaction aspects of web applications rather than security testing or access control properties. A state of the art discussion of these techniques can be found on our recent survey [13].

Several methods in the literature propose tools for the translation from UML diagrams into formal models that can be checked using existing formal verification tools. Here, we will review two methods as an example of methods used in the context of web application verification. First, the Castelluccia et al. [40] and Sciascio et al. [73] use the Canollen model in order to build a diagram for the web application, where the aim is to verify the design of the application. In order to apply model-checking techniques to any model, it must be formal, so in this work the authors implement a component, XMI2SMV, that converts UML diagrams in XMI format into a Web Application Graph (WAG), where the WAG can itself be translated into an SMV model which is given as input to the NuSMV model checker [50].

Second, in the Bordbar and Anastasakis method [33] the authors use the Alloy [107] constraint solver to find bugs in the process of 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 (ADI), which models the interaction with the web application, to an Alloy model, which can then be verified using
CHAPTER 8. RECOVERING RBAC SECURITY MODELS FROM WEB APP.

the Alloy analyzer. Other sets of translation tools are discussed in [92].

8.8 Conclusions and Future Work

In this chapter we have presented an approach and a tool to recover a role-based access control (RBAC) security model from dynamic web application. We used source transformation technology in PHP2SecureUML to implement the model-to-model transformation and composition, and in SecureUML2Prolog to transform the recovered model into Prolog. The resulting formal model can be used to check for RBAC security properties in the application under test. In the next chapter, we will demonstrate the usefulness of our approach on the process of analysis, testing and maintenance of web applications.
Chapter 9

Examples

In this chapter we demonstrate our framework on the analysis of PhpBB 2.0, a popular internet bulletin board system whose characteristics are summarized in Table 9.1. Good candidate systems to assess our approach are web applications that are open source, and built using the combination of Apache server, PHP, and MySQL. Our framework is adaptable to other technologies as well. The most important requirement is that the web application should have some kind of permission system.

Because our approach is based on static and dynamic analysis, we require source code. Our choice of the combination of PHP, MySQL, and Apache server is based on the popularity of these technologies. According to (Netcraft) [134], Apache web server is the most deployed web server on the internet with a 58.7% market share. PHP has been the most popular server-side scripting language for years and is likely to remain so for some time. As of April 2007, there were more than 20 million websites (domain names) using PHP [142]. MySQL as well is the fastest-growing database in the industry, with more than 10 million active installations and 50,000 daily downloads [3]. The approach could be applied to other technologies as well.
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 row 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 (a password text field) exist in the current page then
            12. username.value = registeredUser_username
            13. password.value = registeredUser_password
        14. if the login button exist
            15. press the login button
       16. }% end function
    17. function navigate (Lns: list of links)
    18. for each link(I) in Lns do
       19. { click link I
          20. collect all links in the page generated from link I: Lns2
          21. collect_formElements
          22. Navigate(Lns2)
       23. }

Figure 9.1: The CollectFormInputs algorithm for collecting the application’s forms input elements

<table>
<thead>
<tr>
<th>Total lines of code</th>
<th>~100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP pages</td>
<td>72</td>
</tr>
<tr>
<td>HTML Pages</td>
<td>15</td>
</tr>
<tr>
<td>Templates &amp; formatting files</td>
<td>109</td>
</tr>
<tr>
<td>Database tables</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 9.1: Characteristics of the PhPBB2.0 application

9.1 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 forms inputs, and a second time to do the actual navigation with an automated form filling. We developed two Watir test cases to implement this automated navigation and tailored these test cases for each of the three roles.
The first test case dynamically collects all 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 9.1 describes the algorithm for this test case. The resulting spreadsheet is used later to fill the value field for each form input.

The second Watir test case consults the spreadsheet created in the previous step 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 9.1, but instead of collecting forms inputs, it searches for forms’ inputs ids in the navigated pages, then picks the values of the matched fields from the spreadsheet and populates the form.

Tables 9.2, 9.3, 9.4 show some statistics about this experiment. Table 9.2 shows the total number of pages visited by users in different roles as well as the number of filtered client pages stored and analyzed by our approach.

Table 9.3 presents coverage information for the testing scenarios. This represents the percentage of SQL statement coverage, the percentage of page access coverage and the server environment variable coverage. Table 9.4 provides more detailed coverage information.
**CHAPTER 9. EXAMPLES**

### SQL statements Coverage

<table>
<thead>
<tr>
<th>Statement\Variable type</th>
<th>SEL</th>
<th>INS</th>
<th>DEL</th>
<th>UPD</th>
<th>SEL</th>
<th>INS</th>
<th>DEL</th>
<th>UPD</th>
<th>ALL</th>
<th>POST</th>
<th>GET</th>
<th>COOKIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anon. user-All trace</td>
<td>42/237</td>
<td>3/38</td>
<td>3/74</td>
<td>5/91</td>
<td>21</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>30/374</td>
<td>12</td>
<td>255</td>
<td>15</td>
</tr>
<tr>
<td>Register user-All trace</td>
<td>71/237</td>
<td>5/38</td>
<td>4/74</td>
<td>15/91</td>
<td>43</td>
<td>5</td>
<td>4</td>
<td>15</td>
<td>75/374</td>
<td>39</td>
<td>255</td>
<td>21</td>
</tr>
<tr>
<td>Admin. user-All trace</td>
<td>75/237</td>
<td>5/38</td>
<td>4/74</td>
<td>15/91</td>
<td>47</td>
<td>5</td>
<td>4</td>
<td>15</td>
<td>92/374</td>
<td>54</td>
<td>255</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 9.4: More detailed coverage information for SQL statements, and server environment variables

% computes role actions per page access
rolePagesActions(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
rolePagesActionsList(RoleName,Bag):-
    setof([PageName, ActionID, RList],
          rolePagesActions(RoleName, PageName, PageID, ActionID, RList), Bag).

% computes actions’ set difference between two roles
anonAccessAdmin_Actions(RoleName1,RoleName2):-
    rolePagesActionsList(RoleName1,Bag1),
    rolePagesActionsList(RoleName2,Bag2),
    remBag(Bag2,Bag1,[],Result),
    printlists(Result).

Figure 9.2: Prolog rules to check for unauthorized access on the application’ entities

based on the server environment variables accessed and SQL statement covered and SQL statement’s use of the application variables.

In the following subsections we demonstrate three testing scenarios to illustrate how our framework can be used for:

- Web application testing for unauthorized access.

- Web applications role-based access control maintenance.

- Web applications role-based access control reengineering.
### 9.1.1 Testing for Unauthorized Access

Most web applications try to implement access control policies using obscurity, where links to pages are not presented to unauthorized users. This method of protection is not sufficient because attackers can attempt to access hidden URLs, knowing that sensitive information and functions lie behind these URLs. In this testing scenario we show how our framework can be used to check for unauthorized access to the application resources including pages, server environment variables and entities. Specifically, we check whether an anonymous user can access any unauthorized content of the PhpBB2.0 by allowing him to access all of the links that an administrator can see when accessing the same forum.

To check for this capability, we first implement and run a Watir test case that dynamically collects all the links and forms’ inputs in all the PhpBB 2.0 pages for the administrator role, and stores them in an Excel spreadsheet. We excluded the administrator visits to the administration panel which includes the forum’s management’s tasks. Second, using the data collected in the spreadsheet, we run another Watir test case that uses this data to navigate the forum as an anonymous user role. Third, the execution trace collected for the anonymous user attempting to access administrator links and data is used by PHP2XMI and WAFA to generate a sequence diagram to reflect this behaviour. We then used 2SecureUML and SecureUML2Prolog to generate the SecureUML model and Prolog program for this scenario.

Our goal is to compare the access control Prolog facts collected in this scenario with those collected for a legitimate visit of an anonymous user to the forum. We run a Prolog
Table 9.6: Unauthorized pages access with parameters for a guest user attempting to access administrator’s links. *PhpBB react code: access redirected(1), error message(2), access allowed(3)*

<table>
<thead>
<tr>
<th>PID</th>
<th>P_Name</th>
<th>P_Parameter</th>
<th>Link Name</th>
<th>PhpBB react code</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>faq</td>
<td>?mode=bbcode</td>
<td>BBCName</td>
<td>3</td>
</tr>
<tr>
<td>109</td>
<td>index</td>
<td>?mark=forums</td>
<td>Mark all forums read</td>
<td>3</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?logout=true, sid=f0266abbcb89963e0df82dcd77ad44f3</td>
<td>Log out (alalfi)</td>
<td>3</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>?f=2, sid=f0266abbcb89963e0df82dcd77ad44f3</td>
<td>moderate this forum</td>
<td>2</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>?f=2, start=0, sid=f0266abbcb89963e0df82dcd77ad44f3</td>
<td>moderate this forum</td>
<td>2</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>?mode=p, p=5, t=3, sid=f0266abbcb89963e0df82dcd77ad44f3</td>
<td>View IP address of poster</td>
<td>2</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>?mode=lock, sid=f0266abbcb89963e0df82dcd77ad44f3</td>
<td>lock this topic</td>
<td>2</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>?mode=split, sid=f0266abbcb89963e0df82dcd77ad44f3</td>
<td>Move this topic</td>
<td>2</td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>?mode=delete, p=5</td>
<td>Delete this post</td>
<td>2</td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>?mode=editpost, p=5</td>
<td>Edit/Delete this post</td>
<td>2</td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>?mode=newtopic, f=2</td>
<td>new topic</td>
<td>1</td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>?mode=quote, p=5</td>
<td>Reply with quote</td>
<td>1</td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>?mode=reply, t=3</td>
<td>post reply</td>
<td>1</td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>?mode=smilies</td>
<td>view more Emoticons</td>
<td>3</td>
</tr>
<tr>
<td>257</td>
<td>privmsg</td>
<td>?folder=inbox, mode=read, p=1</td>
<td>Log in to check your private messages</td>
<td>1</td>
</tr>
<tr>
<td>257</td>
<td>privmsg</td>
<td>?folder=inbox, sid=f0266abbcb89963e0df82dcd77ad44f3</td>
<td>Log in to check your private messages</td>
<td>1</td>
</tr>
<tr>
<td>324</td>
<td>search</td>
<td>?search_id=egosearch</td>
<td>View your posts</td>
<td>1</td>
</tr>
<tr>
<td>324</td>
<td>search</td>
<td>?search_id=newposts</td>
<td>View posts since last visit</td>
<td>1</td>
</tr>
<tr>
<td>391</td>
<td>viewtopic</td>
<td>?t=3, start=0, postdays=0, postorder=asc, highlight=</td>
<td>Watch this topic for replies</td>
<td>3</td>
</tr>
<tr>
<td>794</td>
<td>adminindex</td>
<td>?sid=f0266abbcb89963e0df82dcd77ad44f3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=admin/index.php, sid=67f7242f66b2f47a7651675e</td>
<td>Redirection</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=posting.php, mode=newtopic, f=2</td>
<td>Redirection</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=posting.php, mode=quote, p=3</td>
<td>Redirection</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=posting.php, mode=reply, t=3</td>
<td>Redirection</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=privmsg.php, folder=inbox, mode=post</td>
<td>Redirection</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=privmsg.php, folder=inbox, mode=reply, t=1</td>
<td>Redirection</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=privmsg.php, folder=inbox, mode=move, p=1</td>
<td>Redirection</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=search.php, search_id=egosearch</td>
<td>Redirection</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td>?redirect=search.php, search_id=newposts</td>
<td>Redirection</td>
<td>1</td>
</tr>
</tbody>
</table>

query that computes the set differences of page access, server environment variable access, and access to the application entities. Figure 9.2 shows a sample of Prolog queries used to implement this goal, and query results are presented in Tables 9.5, 9.6, 9.7, 9.8.

After analyzing the results of the Prolog queries, we have determined that PhpBB 2.0 reacts to attempted access to unauthorized resources in three ways:
Table 9.7: Unauthorized server's environment variables access for a guest user attempting to access administrator’s links

<table>
<thead>
<tr>
<th>PID</th>
<th>P_Name</th>
<th>Var_ID</th>
<th>Http Var Name</th>
<th>Http Var Value</th>
<th>Http Var Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>faq</td>
<td>19</td>
<td>mode</td>
<td>bbcode</td>
<td>GETDT</td>
</tr>
<tr>
<td>109</td>
<td>index</td>
<td>119</td>
<td>c</td>
<td>$viewcat</td>
<td>GETDT</td>
</tr>
<tr>
<td>109</td>
<td>index</td>
<td>120</td>
<td>mark</td>
<td>$mark_read</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>191</td>
<td>f</td>
<td>forum_id</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>193</td>
<td>p</td>
<td>$post_id</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>200</td>
<td>t</td>
<td>$topic_id</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>209</td>
<td>confirm</td>
<td>$confirm</td>
<td>POSTDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>210</td>
<td>start</td>
<td>$start</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>211</td>
<td>mode</td>
<td>p</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>215</td>
<td>sid</td>
<td>$sid</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>225</td>
<td>posting</td>
<td>S Var</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>225</td>
<td>posting</td>
<td>S Var</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>257</td>
<td>privmsg</td>
<td>Sprivmsg_id</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>319</td>
<td>profile</td>
<td>$sid</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>324</td>
<td>search</td>
<td>$search_author</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>365</td>
<td>viewforum</td>
<td>$mark_read</td>
<td>GETDT</td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td>391</td>
<td>viewtopic</td>
<td>$postorder</td>
<td>GETDT</td>
</tr>
</tbody>
</table>

1. Attempted access to some pages is redirected to the login page. Examples are access to the posting pages (255), privmsg pages (257), adminindex page (794), and the search pages (324). The last eight entries in Table 9.6 show these redirections to the login page.

2. Other pages are not properly protected, as they are not redirected to the login page. If such pages are illegitimately accessed an error message is reflected back such as “invalid session ID”. Access to the modcp (145) pages, shown in Table 9.6, are examples of this case.

3. Some others 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 posting (255) are examples of this case.

Table 9.7 shows the result of executing the Prolog set difference query on server environment variables, and Figure 9.8 shows the result of executing the Prolog set difference
CHAPTER 9. EXAMPLES

### Table 9.8: Unauthorized SQL statement access for a guest user attempting to access an administrator’s links

<table>
<thead>
<tr>
<th>PID</th>
<th>P_Name</th>
<th>Action ID</th>
<th>Action return-value</th>
<th>Action constraint</th>
<th>Action aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>145</td>
<td>modcp</td>
<td>152</td>
<td>$forum_id, $forum_name, $forum_topics</td>
<td>t.topic_id = $topic_id and f.forum_id = t.forum_id</td>
<td>phpbb_topics t, phpbb_forums f</td>
</tr>
<tr>
<td>157</td>
<td>forump_name, forum_topics</td>
<td>forum_id = $forum_id</td>
<td>phpbb_forums</td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>226</td>
<td>t.forum_id = $forum_id</td>
<td>phpbb_forums</td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>227</td>
<td>f.*, t.topic_status, t.topic_title, t.topic_type</td>
<td>t.topic_id = $topic_id and f.forum_id = t.forum_id</td>
<td>phpbb_forums f, phpbb_topics t</td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>229</td>
<td>poster_id IN ($matching_userids)</td>
<td>phpbb_smilies</td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>search</td>
<td>329</td>
<td>topic_id</td>
<td>topic_replies = 0 and topic_moved_id = 0</td>
<td>phpbb_topics</td>
</tr>
<tr>
<td>324</td>
<td>search</td>
<td>341</td>
<td>pt.post_text, pt.bbcode_uid, pt.post_subject, p.<em>, f.forum_id, f.forum_name, t.</em>, u.username, u.user_id, u.user_sig, u.user_sig_bbcode_uid</td>
<td>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</td>
<td>phpbb_forums f, phpbb_topics t, phpbb_users u, phpbb_posts p, phpbb_posts_text pt</td>
</tr>
<tr>
<td>388</td>
<td>viewonline</td>
<td>389</td>
<td>forum_name, forum_id</td>
<td>phpbb_forums</td>
<td></td>
</tr>
<tr>
<td>388</td>
<td>viewonline</td>
<td>390</td>
<td>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</td>
<td>u.user_id = s.session_user_id and s.session_time &gt;= (time() - 300)</td>
<td>phpbb_users u, phpbb_sessions s</td>
</tr>
<tr>
<td>391</td>
<td>viewtopic</td>
<td>393</td>
<td>t.topic_id</td>
<td>t2.topic_id = $topic_id and t.forum_id = t2.forum_id and t.topic_moved_id = 0 and t.topic_last_post_id &gt; t2.topic_last_post_id</td>
<td>phpbb_topics t, phpbb_topics t2</td>
</tr>
</tbody>
</table>

query on the application actions, entities and attributes.

**9.1.2 Web application maintenance**

Our framework can be used for web application role-based access control (RBAC) maintenance purposes. For example, this can be useful when the testing engineer has identified an access control security feature that is legitimacy permitted to a specific role and wants to disable this feature. Our framework will help locate all the pages and database statements that allow this feature as a step towards fixing the code to prevent this access. In PhpBB 2.0, we have noticed than an anonymous user is allowed to see other users’ profile information, a feature that may lead to a privacy violation for the forum members. We executed a Prolog query that searches for profile information in all accesses represented in
CHAPTER 9. EXAMPLES

Testing scenario 2: Our framework can be used for web application RBAC maintenance purposes. Particularly if the testing engineer has identified an access control security feature that is legitimacy permitted to a specific role and wants to disable this feature. Our framework will help locate all the pages and database statements that allow this feature as a step towards fixing the code to prevent this access. In PHPBB 2.0, we have noticed than an anonymous user is allowed to see other users profile information, a feature that may lead to some privacy violation for the forum members. We executed a prolog query that will search for profile information in all return facts in the recovered model of an Anonymous user. The query returns information all about the pages that permits such access, and it also identifies SQL statements that retrieves such 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 for such features.

Figure 9.3 shows a sample of the Prolog queries used to implement this goal, and Table 9.9 presents the result of executing these queries.

9.1.3 Web Applications Reengineering

In PHPBB 2.0, we have noticed that the administrator management tasks are protected by providing a valid administrator username and password via a login page. Users who provide such information can access all the forum management tasks afterwards. The way this is implemented is by having all the role validation checked at the beginning of each restricted page using a call to a “pagestart” function that implements the validation process based on sessions information, after which it decides the level of access. After commenting out such code in the “pagestart”, an anonymous user was able to access all the administrator management pages, given that he knows the URL address for them. Figure 9.2 shows the source code of the “pagestart” file and highlights the part of it that is responsible of controlling the access.

To address this issue along with the fact that most web application’s access control
Table 9.9: List of pages and actions that permits a user to access other user’s Email address are implemented using obscurity, such applications need to be reengineered to employ a strict security model not only on the level of page access but also on the level of server environment variable access as well as access to application entities and attributes.

Our framework automatically generates a role-based security model for an existing web application, which can then be reviewed by a security engineer either by accessing the visualized SecureUML model using any modeling tool that supports UML2.0 (such as RSA [105]), or by exploring the Prolog representation for the generated security model. Either way the security engineer can check for the absence of any legitimate access or the existence of any unauthorized access to the web application under test. Since the recovered model provides a fined-grained access information down to the level of the application’s entities’ attributes, the software engineer can explore the effect of a proposed update to the access

<table>
<thead>
<tr>
<th>PID</th>
<th>Page_Name</th>
<th>Action ID</th>
<th>Action return-value</th>
<th>Action_constraint</th>
<th>Action aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>index</td>
<td>140</td>
<td>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_allowavatars</td>
<td>user_id &lt;&gt; -1 ORDER BY $order_by</td>
<td>phpbb_users</td>
</tr>
<tr>
<td>139</td>
<td>memberlist</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>257</td>
<td>privmsg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>profile</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>search</td>
<td>400</td>
<td>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, u.user_rank, u.user_sig, u.user_sig_bbcode_uid, u.user_avatar, u.user_avatar_type, u.user_allowavatars, u.user_allowsmile, p.*, pt.post_text, pt.post_subject, pt.bbcode_uid</td>
<td>p.topic_id = $topic_id $limit_posts_time and pt.post_id = p.post_id and u.user_id = p.poster_id</td>
<td>phpbb_posts p, phpbb_users u, phpbb_posts_text pt</td>
</tr>
<tr>
<td>126</td>
<td>login</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>modcp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>391</td>
<td>viewtopic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>posting</td>
<td>863</td>
<td>u.user_id, u.user_email, u.user_lang</td>
<td>tw.topic_id = $topic_id and tw.user_id NOT IN ($userdata[&quot;user_id&quot;,&quot;-1, $row[ban_userid]]&quot; and tw.notify_status = 0 and u.user_id = tw.user_id</td>
<td>phpbb_topics_watch tw, phpbb_users u</td>
</tr>
</tbody>
</table>

are implemented using obscurity, such applications need to be reengineered to employ a strict security model not only on the level of page access but also on the level of server environment variable access as well as access to application entities and attributes.
<?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, $lang ['Not_admin']);
} else
if ($HTTP_GET_VARS ['sid'] != $userdata ['session_id'])
{
    redirect ("index.$phpEx?sid=".$userdata ['session_id']);
} else
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 9.4: PageStart.php file in PhpBB 2.0, code that control the access on administration management pages is highlighted

model either by updating the security model directly 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 database schema, such that the security check will be employed on each access of any of the application entities’ attributes based on the role of the user accessing the application. The database access in the legacy web application can be updated accordingly based on the new database restructuring. This will be possible by the unique numbering generated by our framework and associated with each SQL statement and server environment variable located in any of the application’s server pages.
9.2 Conclusions

This chapter presents a demonstration of our security analysis framework on one of the medium-sized production applications, PhpBB 2.0. Specifically, we have illustrated the use of our framework in security analysis, testing, maintenance and reengineering using PhpBB 2.0 as an example. In the next chapter, we present our vision for a future large scale evaluation to better test the effectiveness of this work. We also discuss possible extensions and adaption of the framework to address other security analysis tasks.
Chapter 10

Summary and Future Work

In this thesis, we focused on one of the most serious web application vulnerabilities, broken access control. Current technologies such as anti-virus software programs and network firewalls provide reasonably secure protection at the host and network levels, but not at the application level. When network and host-level entry points are comparatively secure, public interfaces of web applications become the focus of malicious software attacks. Attackers often try to access unauthorized objects and resources other than URL pages in an indirect way; for instance, using indirect access to back-end resources such as databases. The consequences of these attacks can be very destructive, especially when the web application allows administrators to remotely manage users and contents over the web. In such cases, the attackers are not only able to view unauthorized content, but also to take over site administration.

To protect against these types of attacks, we have designed and implemented a security analysis framework for dynamic web applications. A reverse engineering process is performed on an existing dynamic web application to extract a role-based access-control security model. A formal analysis is applied on the recovered model to check access-control security properties. This framework can be used to verify that a dynamic web application conforms to access control polices specified by a security engineer.
Our approach makes use of Model Driven Engineering. It automatically constructs a role-based access control security model from the recovered structural and behavioral models. We use TXL, a source transformation technology, to implement the automatic model-to-model transformation and composition. The generated model is represented in the UML 2.1 exchange format, XMI 2.1.

Based on the model-to-model transformation approach, we have also developed a tool to automatically transform the semi-formal UML 2.1 security model into a formal model to ease the process of verifying the system against security properties. Our framework provides a set of novel techniques for the analysis and modeling of web applications for the purpose of security verification and validation. It is largely language independent, and based on adaptable model recovery which can support a wide range of security analysis tasks and systems.

Following are some of the key contributions of this research:

**A comprehensive survey of modeling methods used in web site verification and testing**

Models are considered an essential step in capturing different system behaviors and simplifying the analysis required to check or improve the quality of software. Verification and testing of web software requires effective modeling techniques that address the specific challenges of web applications. In our survey, based on a short catalogue of desirable properties of web applications that require analysis, two different views of the methods are presented: a general categorization by modeling level, and a detailed comparison based on property coverage. To the best of our knowledge this is the first study that provides a comprehensive review and comparative study of modelling methods that are currently applied in the field of web application verification and testing.
SQL2XMI: an automated transformation approach from an SQL (DDL) schema to an open UML2.0-adapted class model

In the second part of this thesis we designed and implemented an automated transformation from an SQL (DDL) schema to an open XMI 2.1 UML-adapted class model. The adapted model is a tailored UML class model to represent the basic ER diagram components, including entities, attributes, relations, and primary keys. Our transformation technique with its tool, SQL2XMI, is a novel one in that it is open, non-vendor specific, and targeted at the standard UML 2.1 exchange format, XMI 2.1. Although comparable commercial transformations exist, they are closed technologies targeted at formats tightly coupled to the vendor’s tools, hindering portability and preventing users from choosing their preferred tools in the development process.

To bring our prototype tool to an industrial level, several improvements will be needed. It must be generalized to handle SQL database schemas other than MySQL and XMI 2.x versions other than 2.1. Using TXL gives us the ability to integrate handling of other implementations of the SQL standard quickly, simply by overriding the SQL grammar to add the forms of each vendor’s specific extensions.

In Chapter 4, we have begun with just the MySQL implementation of the SQL data definition language (DDL), leaving the improvement of the grammar file to include the data manipulation part (DML) and support for other vendors’ implementations to future work. Our transformation also does not yet take advantage of all of the information available in the schema. Using a more comprehensive transformation rule set, we hope to recover a richer ER model.

Finally, while in this work we have concentrated on reverse engineering an existing MySQL schema to a UML entity relationship diagram, in the future we could use the same technique in the forward engineering direction, using the same technology to generate different SQL database implementations from an ER diagram designed
using any UML toolset that supports XMI 2.1 export.

**Php2XMI and WAFA: An approach to automatically instrument, analyze and model the dynamic behaviour of web applications**

In the third part we designed and implemented an approach to automatically instrument dynamic web applications using source transformation technology, and to recover a sequence diagram from the execution traces generated by the resulting instrumentation. Using an SQL database to store generated execution traces, our approach automatically filters traces to reduce redundant information that may complicate program understanding.

While our current implementation ignores any redundant traces, loops and conditions in web applications can be easily detected and explicitly modeled. We are also planning the integration of sequence diagrams from different sessions to generate one complete sequence diagram for the entire web application.

Furthermore, we developed WAFA, an automated reverse engineering approach to recover fine-grained interaction behavior of dynamic web applications. To the best of our knowledge, our approach is the first one to extract the web application’s embedded SQL subsystem, which includes both the original SQL statement source as well as corresponding execution instances, and an analysis to attach it to both static host application variables and dynamic server environment variables.

We are currently expanding the set of test cases for PhpBB and Moodle, and plan to extend our evaluation to other PHP-based applications. Our approach is primarily aimed at server side code, since we have been working with traditional PHP-based web applications. AJAX requests can also be, and in many cases are, implemented in PHP. When used with AJAX, our technique can be used to directly link HTTP request variables to the database interactions of the AJAX request. This may help in analysis of AJAX applications as well as traditional applications. Finally, we are
working on generalizing PHP2XMI and WAFA for use in testing other web application vulnerabilities.

**DWASTIC: An automated instrumentation coverage approach**

The dynamic analysis is supported by our developed automated instrumentation coverage approach to decrease the percentage of false positives. In support of this approach we proposed a set of new coverage metrics, specialized for dynamic web applications. In addition, we performed a great deal of analysis on the embedded database interaction in the host application. This includes automated distilling of the SQL embedded system, analyzing it, and modeling it as a part of the whole system.

As future work, we plan to extend the approach to handle other web technologies and database engines and to evaluate it on a wide range of applications of different sizes. We also plan to use DWASTIC to support other testing activities for web applications, such as SQL injection and cross-site scripting analysis.

**PHP2SecureUML and SecureUML2Prolog: an approach to automatically construct a role-based access control security model**

In the fourth part, making use of Model Driven Engineering, we automatically constructed a role-based access control security model from the recovered structural and behavioral models. We use TXL to implement the automatic model to model transformation and composition. The generated model is also represented in the UML 2.1 exchange format, XMI 2.1. In the last part, we developed, based on model-to-model transformation approach, a tool to transform the semi-formal UML 2.1 security model into a formal model represented in Prolog to ease the process of verifying the system against security properties.

This Ph.D. dissertation work has thus far been aimed at automatically detecting security vulnerabilities in dynamic web applications [11]. As future work I will extend my security
analysis framework to help improve attack tolerance of software systems. Critical information systems are becoming more distributed and open, which increases their vulnerability to attackers and calls for new methods and technologies to counter-attack and protect the systems [147].

Attacks can target different levels of software such as the network, the host or the application. Firewalls, anti-virus software, and intrusion detection systems act as preventive anti-attack techniques. However, these are frequently targeted by attackers aiming to disable these defenses so that future attacks go undetected. A survivable and dependable system needs not only to detect the presence of attacks or faults, but also to function properly in the face of these faults, especially in mission-critical systems. At the same time, these mission critical systems should also be able to survive faults that are random and unpredictable in nature [124]. As a result, there is an urgent need for frameworks that focus on integrating different security techniques to block, evade, and react to attacks, thus improving system survivability [168].

In this future project, I will generalize my PhD framework to account for different kinds of attacks. Source transformation technology [55] will be used to automatically transform the system into a more attack-tolerant one [147]. A protection mechanism will be proposed and modeled, and a set of testing techniques will be defined and applied to the proposed protection system to ensure its feasibility.

I will start with hardening the application to evade the problem of broken access control, then generalizing my technique to handle other security attacks. Most web application access control policies are implemented using obscurity. Thus, such applications need to be reengineered to employ a strict security model not only on the level of page access but also on the level of server environment variable access and the application entities and attributes.

My PhD framework automatically generates a role-based access control (RBAC) security model for an existing web application [11], which can then be reviewed by a security engineer either by accessing the visualized SecureUML model on any modeling tool that supports
When the security model is revised to reflect the new RBAC security requirements, it can be used to restructure the application database schema such that the RBAC security check will be employed on each access of any of the application entities attributes. My plan is to leverage the same technique I used to develop SQL2XMI [8], a tool that constructs UML2.0 ER diagram from an SQL scheme, to automatically restructure the schema, but this time starting with the recovered security model as source of the transformation and the RBAC secured SQL schema as a target. The resulting SQL schema represents the new database restructuring which will assist in avoiding the RBAC security vulnerabilities in the legacy web application.

My PhD framework creates links between the recovered security model elements and the source code. These links will be used to reengineer the source code so that the database access in the legacy web application can be updated accordingly to comply with the new database restructuring.
Bibliography


