Business Rules Discovery in Web Application through Process Mining

by

Hamza Alkofahi

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 8, 2020

Keywords: process mining, web application, security, business rules mining, business logic vulnerability

Copyright 2020 by Hamza Alkofahi

Committee Members

David Umphress, Chair, Professor of Computer Science and Software Engineering
Levent Yilmaz, Professor of Computer Science and Software Engineering
Tao Shu, Associate Professor of Computer Science and Software Engineering
Anthony Skjellum, Professor of Computer Science and Engineering, University of Tennessee at Chattanooga

Abstract

Advances in information technologies have allowed businesses to deliver their services to new markets that did not exist before, especially in fields such as e-commerce, healthcare, e-education, and cloud services. Web technologies, in particular, have revolutionized the way solutions are built and deployed by enabling the development of platform-independent systems. However, as web applications grow in terms of features and popularity, their complexity also increases accordingly. Aspects of such complexity include role management and decision-making processes, which are formally defined through business rules. More importantly, maintaining the business rules along with code changes is a key factor here, as they formalize how the system should behave. Neglecting or failing to maintain them over time increases the chance of faulty application logic, as the system implementation continues to diverge from its specifications. Therefore, degrading the confidence in these business rules and opening the door for potential Business Logic Vulnerabilities (BLV). BLVs are considered one of the most critical web flaws. Traditional scanning techniques fail to detect them as their detection requires a deep understanding of business processes.

The absence of formal specifications defining the expected system behavior represents a significant challenge for detecting BLVs. In this research, we propose a novel black-box approach for discovering business rules in e-commerce web applications through process mining. Our proposed solution is capable of recovering system specifications while under normal usage addressing the major difficulty toward detecting BLVs. This provides a better understanding of business logic and helps evaluate if they are maintained during the application execution.

This research presents a novel framework for capturing and converting HTTP traffic into high fidelity event logs that complies with the IEEE 1849-2016 XES standard. The proposed solution allows users and developers to build and utilize many process mining techniques. Moreover, using the new framework, we introduced advanced black-box automated approaches for discovering authorization and if-then business rules from the web application's dynamic artifact (HTTP traffic) only. The results of our evaluation indicate high precision in recovering business rules based on perceived behavior.

Acknowledgments

First of all, I would like to thank God Almighty, who has given me the strength and patience to complete this dissertation. Let me take the chance to thank those people who made my achievements possible. My most sincere gratitude and appreciation go to my advisor Dr. David Umphress, for his constant support, guidance, and patience throughout my Ph.D. studies. Thank you for always challenging me to think and work further. You have made this an unforgettable experience that I have enjoyed and learned from over the years.

I would also like to thank my committee members. Dr. Anthony Skjellum, for the continuous support and opportunities you have provided me over the years, as well as Dr. Levent Yilmaz, and Dr. Tao Shu, for your support, guidance, and suggestions.

Many thanks to my parents, brothers, and sisters for their love, support, and continuous encouragement throughout my studies. You believing in me always pushed me forward and made this accomplishment possible.

Finally but, by no means least, I'm deeply indebted to my wife Heba for being always there for me. Raising two lovely kids (Qais and Omar) and pursuing your Ph.D. degree never wavered your support, not even for once. You simply took the stress away from our journey and made it a wonderful time to remember. Thank you Qais and Omar for all the joy and happiness you brought to our life.

Table of Contents

List of Figures

List of Tables

Chapter 1

Problem Statement

Aggressive integration of validation checks into web framework software has altered the attack surface of web applications by reducing the opportunity to inject flaws using input fields as the primary vector. The reaction of the hacking community has been to shift to a more subtle – and more difficult to detect – form of attack, that of discovering and exploiting underlying application business logic. Because compromising systems in this fashion requires an understanding of how the system functions, each attack is uniquely tailored to the system being targeted. The nature of such attacks relegates detection and recovery to time-consuming manual techniques which are difficult to scale to other systems.

This type of vulnerability is referred to as a Business Logic Vulnerability (BLV). BLVs take advantage of flaws in the logic of an application. They are difficult to detect because each application has its business logic that defines how it operates. Detecting flaws in the logic entails comparing actual behavior to a formal specification of expected behavior, referred to as business rules. A formal specification rarely accompanies its respective application. Even if it did, it would become quickly outdated during the application maintenance process. In most cases, the source code is the most enduring application specification but, often, it is not available, especially in cases where the application consists of third-party or proprietary software. Even when available, source code does not guarantee that the intended business logic is implemented correctly.

The objective of this research is to discover business rules by observing HTTP traffic to and from a web application. Extracting business rules in this fashion documents observed behavior which can be analyzed for gaps, the ultimate goal being to detect flaws in logic that can be exploited. The research will use explore the use of techniques such as process mining, web content mining, machine learning, and knowledge extraction and analysis to mine the business rules.

Chapter 2

Background and Literature Review

2.1 Process Mining

Web servers and enterprise resource planning systems produce log data that is typically used to monitor performance, detect cyber intrusions, and identify issues related to network traffic. Attempts to analyze log data to discover much more than summary statistics has been hampered by the extreme volume of data as well as its unstructured nature.

Advanced data exploitation techniques can help in uncovering and gaining much more in-depth insights into business processes. Data mining can be an option when analyzing such large datasets to extract relationships and transform data into a more understandable and useful format, except that it is not a process-centric approach. Unlike process mining, which is built on top of data mining techniques and focuses on process modeling and analysis [57]. Process mining provides a set of tools aims to improve process efficiency and understanding of processes based on event logs.

2.1.1 Perspectives of Process Mining

When collecting log data generated across multiple executions of an information system, several process mining algorithms can be used to discover different perspectives about the process. Figure 2.1 highlights the primary information that can be mined from process data.

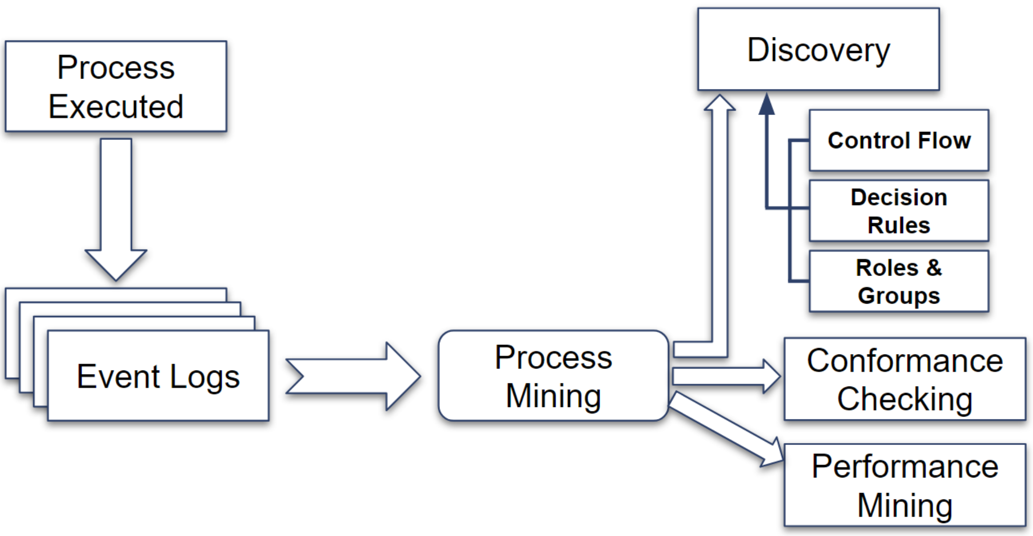


Figure 2.1: Perspectives of process mining

The control flow of a process can be extracted using different process discovery algorithms, such as α-algorithm, heuristic miner, fuzzy miner, inductive miner and genetic miner [94, 95, 96, 97, 98]. Each algorithm follows a different strategy (e.g. divide and conquer vs direct approaches) and produces different types of process models such as perti nets (as in figure 2.2), transition systems, casual net, and business process model and notation (BPMN).

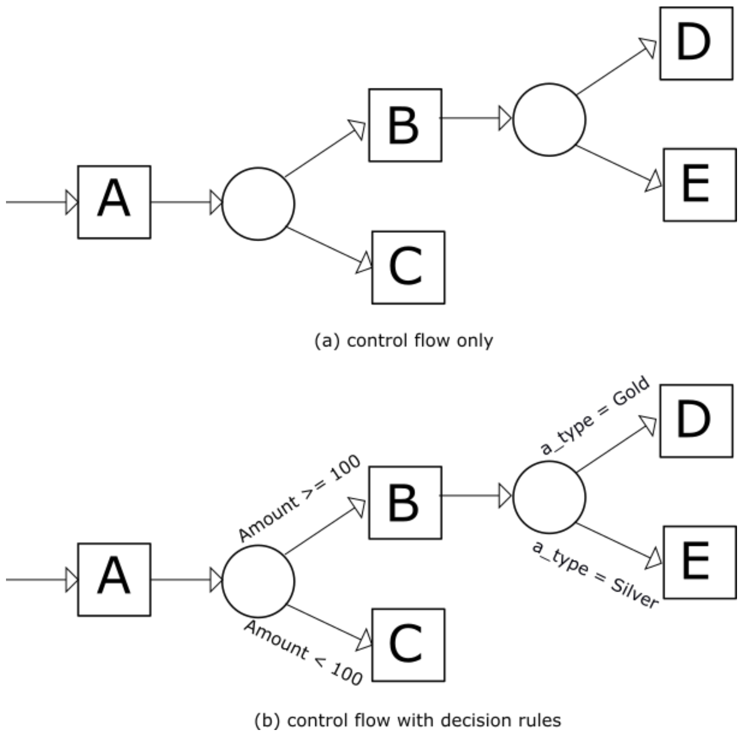


Figure 2.2: Decision rules discovery

In real world applications, transitions between activities within a process are not random but are determined by application logic. Process mining can be used to discover decision points. For example, Figure 2.2 part (a) shows the control flow of a process composed of five activities **A**−**E**. There are two transition options from **A** to (**B** or **C**) and the same applies for activity **B**. But it is impossible to know which transition is taken using only the control flow. To achieve that, process mining uses supervised machine learning aiming to classify instances based on predictor variables [57], which are values that are likely to influence control flow. As a result of decision mining it is possible to identify under what conditions the transition from activity **A** to **B** is likely to take place, as shown in figure 2.2 part (b).

2.1.2 Event Logs to Process Models

Successful process mining relies heavily on event logs, the concept being that such logs contain inputs and corresponding outputs which, if sufficiently comprehensive, provide an insight into the logic needed to transform inputs into outputs. The information used to discover the transforms vary based on the process mining technique[57]. Complicating matters further, event logs may have to be consolidated from raw server logs, databases, enterprise resource planning systems, etc., and be pre−processed into a consistent and usable form before process mining can take place.

In order to use the process mining tools, the pre−processed data needs to be in a certain structure. In general, event log files are composed of arbitrary number of cases (also known as traces), where each case represents one instance of process execution. It consists of a unique identifier accompanied by zero or more events. Each event represents an atomic activity that was recorded by the system and consists of a non-zero number of attributes. Process mining depends on the presence of the following attibutes:

- **Case ID**: each case must have a unique ID.
- **Activity**: each event must have an activity name.
- **Timestamp**: each event must have timestamp to specify when that event happened.
- **Resource**(optional): event attribute to identify the executor of the event.

- **Other Data Attributes**(optional): more event attributes such as cost, completion time, and activity id ...etc.

2.1.3 The Event Log Format: XES Standard

XES (eXtensible Event Stream) is an XML-based standard for event logs. It was adapted by the IEEE Task Force on Process Mining in 2010 and become an official IEEE Standard in 2016 [85]. The standard is supported by a number of process mining tools and libraries such as ProM, Disco, XESame, and OpenXES [63, 64, 65, 66]. Figure 2.3 shows the class diagram that describe the meta-model for XES standard.

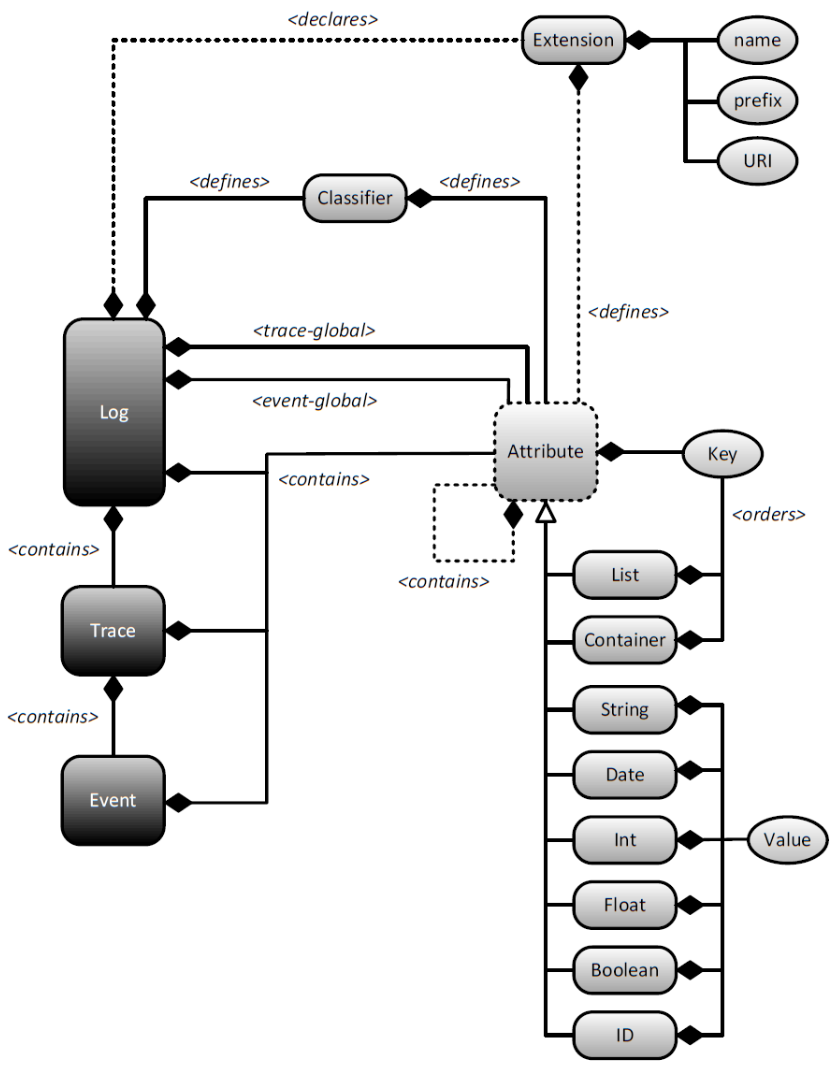


Figure 2.3: The complete meta-model for the XES standard [56]

2.2 Web Application Vulnerabilities and Detection Strategies

A *web vulnerability* is a design flaw or implementation bug in a web application that, when exploited, compromises the application in some unintended fashion. For example, if a developer builds a login page that does not check user-provided input for allowable values, an attacker could supply an SQL statement that gets evaluated and reveals the contents of a database. Such SQL injection attacks represent one of many types of methods by which to exploit an application.

2.2.1 Web App Vulnerability Categories

Web vulnerabilities fall into two general categories: Input Validation Vulnerability (IVV) and Business Logic Vulnerability (BLV) [17] [38]. Vulnerabilities in the former category arises from an application failing to examine values received from the network sufficiently to prevent introduction of unintended executable elements. Vulnerabilities in the latter category stem from orchestrating an application's legitimate sequence of actions to induce unintended consequences.

IVVs are an obvious vector of exploitation because any interaction between the user and the web application involves some exchange of information. The input might be as explicit as the contents of a form being posted back to web application or as subtle as meta information being returned as a result of an HTTP response. Information returning to the web application must be examined before being processed because it might consist of programming code that, if used in conjunction with the equivalent of an $eval()$ method, could perform a function of the attacker's choosing. SQL injections, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) are examples of exploitation methods that fall into the IVV category.

IVVs may seem simple to prevent. While this may have been true a decade ago, advancements in web technologies have increased the complexity of modern web applications which, in turn, have complicated the testing process. For example, a stored XSS attack can be tricky to detect using automated testing techniques if an executable script is an permissible input when the application is in a particular state. Duchene et al. [13] highlight the complexity of XSS detection, especially when the attack relies on factors such as user interaction with a certain part of the system; system views and the different states for each view; and targeted attacks.

BLVs, in contrast, pose an even more difficult challenge to prevent and detect. They rely on the attacker coaxing illicit operations from a web application by manipulating legitimate interaction between it and the user. This requires a considerable understanding of the target web application because the application's own logic is leveraged against itself, logic that is unique to the application itself. Even if two applications share similar business rules, their logic will vary. In short, to exploit a BLV, the attacker must derive some sort of specification of expected behavior [48], then look for some irregularity that can be misused.

2.2.2 Detection of Web Vulnerabilities

Addressing web vulnerabilities can take place at different stages in the Software Development Life Cycle (SDLC) [21], ranging from the planning phase to deployment. While it has been well established that the cost of addressing problems in the early stages of the SDLC is lower than addressing them in the later stages, there is no general prescriptive solution that can guarantee the prevention of vulnerabilities. The most effective approach is to apply protective mechanisms at each phase of the SDLC, in order to provide a defense-in-depth approach as the application emerges from mental image to code. Table 2.1 highlights significant research into vulnerability detection and prevention at different phase in the SDLC.

Phase	Reference	Activity
Requirement	Sindre and Opdahl [2] Haley et al. [3]	Establishing Security Requirements
Design	Shostack [4] [5] Burns [6]	Threat modeling
	Verdon and McGraw [7]	Design Risk Analysis
	Allen [8]	Architecture Risk Analysis
Code	**Input Validation Flaws:** Thomas and Williams [24] Grabowski, Hofmann and Li [33] Juillerat [34] Johns and Beyerlein et al. [35]	Secure Coding

	Logical Flaws: OWASP AppSensor [39]	
Testing	**Input Validation Flaws:** Huang et al. [9] Ciampa et al. [11] Appelt et al. [37] Duchene and Rawat et al. [12] [13] Rathore et al. [14] **Logical Flaws:** Felmetsger et al. [17] Doupé et al. [18] Pellegrino and Balzarotti [20] Deepa et al. [21] [36] Wen et al. [48] Alkhalaf et al. [51] Bisht et al. [50] [51]	Vulnerability Detection
	Input Validation Flaws: Bisht et al. [23] Thomas and Williams [24] Scholte et al. [25] **Logical Flaws:** Krishnamurthy et al. [27] Dalton et al. [28] Son, McKinley and Shmatikov [29]	Vulnerability Prevention
Deployment	Martínez, Cosentino and Cabot [41].	Secure Configuration
	Input Validation Flaws: Lee et al. [30] Halfond and Orso [31] Jang and Choi [32] **Logical Flaws:** Li and Xue [16] Cova et al. [19]	Attack Detection
	Logical Flaws: Skrupsky et al. [26]	Attack Prevention

Table 2.1: Security at different phases of SDLC

Detection of Input Validation Web Vulnerability

The premise of IVVs – that of supplying a web application with an impermissible executable element in lieu of valid data – is well defined, making IVVs feasible to detect by tracking their signature over time. Consider, for example, a shopping application that displays product information specified by a URL along the lines of `http://www.example.com/view_item.php?id=xx`, where xx identifies the desired product. The value of the variable, *id*, is supplied by the user and raises the question as to whether it is checked for validity by the application. With some probing, an attacker could provide a questionable value and use the result to assess the extent of the validity checking mechanism. If the attacker were to attach a single quotation mark to the value (e.g., `...?id=10'`) and receive an SQL-related error, this would indicate that the application uses the value as part of an SQL statement that is constructed during in the course of providing product information. This would lead the attacker to possibly follow-up by inputting an SQL statement that might yield useful information, such as `...?id=10;SELECT * FROM sys.tables;`.

The detection of IVVs received considerable research attention over the past decade. Huang et al. developed WAVE[9], a framework for detecting web vulnerabilities in an automated black-box approach. SQL injection attacks determined the effectiveness of their approach. They developed WebSSARI[10] to achieve the same goal through a white-box approach that analyzed source code to detect possible flaws. V1p3R, developed by Ciampa et al. [11], uses a heuristic-based approach to detect SQL injection vulnerabilities. It constructs a knowledge base from observations made from running an application and analyzing its output. It runs the application using non-random SQL queries generated from information inferred from the output, thus ”learning” the extent of the application's exposure to SQL injection attacks. Ciampa et al. show that this approach is superior to other tools, such as SQLMap[75] that generate SQL queries with non-dependent query strings. 4SQLi [37] goes a step further by randomize the generation of valid SQL queries in an effort to increase the likelihood of discovering as many SQL injection vulnerabilities as possible. This technique has been shown to bypass web application firewalls that monitor and block malicious HTTP traffic.

Cross-site scripting (XSS) attacks pose complications beyond those of SQL injections due to their use of legitimate web interaction mechanisms, such as cookies or HTML statements, to circumvent security and privacy controls. Duchene et al. [13] developed a black-box fuzzer, LigRE, that aims to find advanced stored XSS vulnerabilities by inferring control flows and data flows from execution traces of a target application and using the acquired knowledge to detect XSS flaws in an automated fashion. LigRE's dependence on reverse engineering to determine where to inject malicious values limits its use in on a large scale. In response, KameleonFuzz, extends LigRE by generating malicious inputs and studying their effects when injected [12]. KameleonFuzz employs a genetic algorithm-driven fuzzer to generate input suitable for XSS attacks and injects the values into pre-determined locations of the targeted application. It produces a taint-aware parse tree from output, which is used to assess whether the injected input actually triggered an XSS-generated exploitation of the application.

While dynamic analysis was the base for the previously mentioned XSS detection techniques, here we highlight some static analysis based solutions. Take, for example, social media networks (such as MySpace, Facebook and Twitter), the huge increase in the number of users attracted the attention of hackers, which as a result accelerated the growth of XSS worms targeting social media websites, such as Samy worms [83]. As a response Rathore et al. [14] used machine learning techniques to detect this type of worm, where the detection method relied on three sets of features: URL features, HTML tag features and social media features. Wassermann et al. [15] extended the PHP string analyzer to support tracking of untrusted information flow, which does not only detect unchecked untrusted data, but also check for insufficiently-checked untrusted data.

As we mentioned earlier in this section, input validation flaws rely on the existence of some level of interaction between a user and a website and the structure of the exchanged data between both parties. It also relies on a set of signatures to identify the existence of a flaw and distinguish between different types [11]. Capturing the characteristics of those flaws becomes possible [17], since those factors are well known for all input validation vulnerabilities across different web applications. This opened the door for building fully automated tools that can completely scan web applications and detect their flaws at a good level of accuracy [40], some example of opensource tools are: W3af, Vega, Nikto, and OWASP Zed Attack Proxy [67, 68, 69, 70]. Also several commercial options are available such as Burp Suite Scanner, Netsparker, AppScan, and Acunetix [71, 72, 73, 74].

Detection of Logical Flaws

Understating logic vulnerabilities requires a good knowledge of the business rules underlying the application being tested, and that is the major difference when compared to input validation flaws. The detection of logical flaws represents a real challenge due to the fact that exploitations use a legitimate input that bypasses input validation mechanisms. Take, for example, a money transfer web service which takes three parameters as inputs: `sender_id`, `receiver_id`, and `amount`, where `sender_id` represent the *ID* of the logged-in user (the one sending the money) and `receiver_id` is the user getting the transferred `amount`. Because the *IDs* of the sender and receiver represent legitimate inputs, a malicious logged-in user could swap the two and transfer funds in the opposite direction of what was intended unless there was a some check in the application logic that prevented this from happening.

Wang et al.[47] and Pellegrino et al. [20] show another example of the impact of logical flaws and the level of complexity involved in finding such flaws. Malicious shoppers were able to pay less or even shop for free in some cases by exploiting inefficiencies in communication between merchant websites and third-party payment systems. Since many merchant websites prefer to integrate third-party services (e.g. Cashier-as-a-Service) rather than building these services themselves, this increases the chance of introducing security flaws. Wang et al. were able to shop online for free [47] by taking advantage of logical flaws (input tampering and and workflow bypass) in the targeted website by altering and skipping several HTTP calls (API) between merchants and the third-party cashiers.

Recently research has focused on vulnerability analysis of business logical flaws by addressing exploits due to **parameter tampering**, **access-control bypass** and **workflow bypass**.

Parameter tampering attacks aim to manipulate the values of the information exchanged within the POST/GET transpiring between the client and the web application, the purpose being to force the web application to do things the originator of the request is not ordinarily authorized to do. Bisht et al. proposed a black-box approach to detect this type of flaw and implemented their approach in a tool called NoTamper [50], which analyzes HTML code and JavaScript to generate logical formulas that capture the constraints enforced on the client input. Using the generated formulas, it sends both valid and illegitimate input to the targeted web server and observes its response. The limited validation, combined with the high false positive and false negative rates, made the approach generally untenable. Bisht et al. published a follow to NoTamper, WAPTEC [49], which employs dynamic analysis to consider the client-side code but also the server-side source-code along with the database schema. Alkhalaf et al. [51] used a static analysis technique to model the server code and were able to find more parameter tampering vulnerabilities compared to WAPTEC because the performance of dynamic analysis techniques heavily relies on the percentage of the accomplished coverage.

Access-control bypass attacks take advantage of weaknesses in access control policies (ACPs) implementation to access restricted resources or execute unauthorized commands. In order to detect this attack, Cova et al. developed Swaddler [19], which is a white-box approach to detect access control and workflow violation. Swaddler first learned the normal behavior of the web application, then learn to create relationships between application's critical execution points and the learned behavior, and finally used anomaly detection techniques at runtime to detect violations in user behavior.

Li and Xue proposed a black-box approach to detect access control violations [16], where a proxy is installed between the web server and the clients to collect and monitor all HTTP traffic. Using session information to isolate traces, they extracted three types of invariants that flagged a request as invalid if violated. Pellegrino et al. introduced a more advanced technique that only required the HTTP conversations to extract behavioral patterns which then used by an oracle to detect different types of logical flaws by detecting [20].

Challenges, the nature of BLV makes the detection process very difficult and requires some level of specifications that describe the expected behavior. The absence of these specifications limit the capabilities of the proposed solutions for detecting BLV [20][21, 36][18][16]. One main contribution to this area is developing a general and automated solution for characterizing applications logic [22], which will enable the development of more advanced and automated detection techniques in a black-box fashion.

2.3 Business Rule Mining

Originally, the modernization of poorly documented legacy systems encouraged the need to extract business rules. As the source code was the only actual documentation, various static and dynamic techniques were introduced to address the problem. This section provides a brief background on business rules and highlights some of the mining efforts to recover these rules.

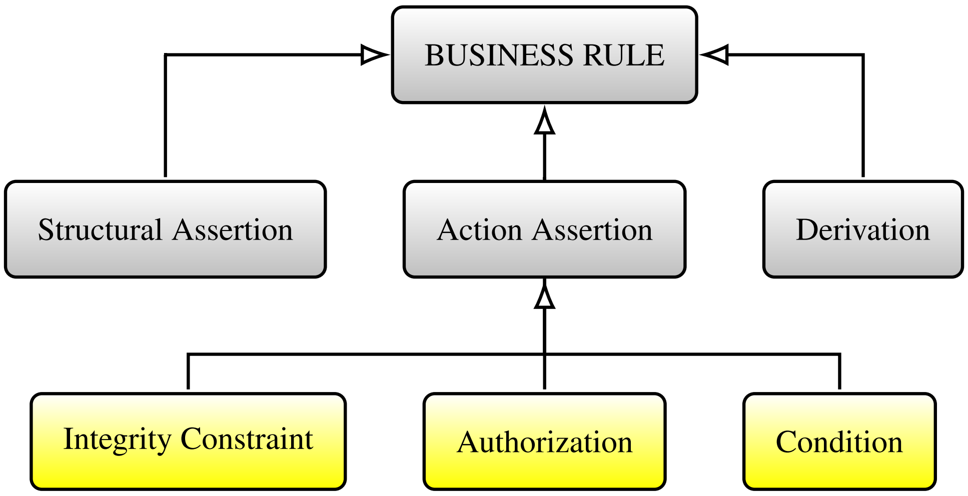


Figure 2.4: Business rules classification

2.3.1 Business Rules

The Business Rule Group (BRG) defines a *business rule* as a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business [42]. Put simply, a business rule defines what can and can not be done, and also it controls the decision-making process by defining conditions and their related actions. Restricting the use of a promotional code only once at check-out time is an example of an e-commerce application business rule. Ideally, each business rule should be atomic; clear and understandable by people within the business scope; and easily implemented by the development team.

There is no standard taxonomy for business rules [43], consequently, we use the classification defined by the business rules group as shown in figure 2.4, which classifies rules into three major types: Structural Assertion, Action Assertion, and Derivation [42]. Our focus is on action assertion, because this type of business rule is used to define constraints and conditions that control the behavior of the business, while the other two types are used to define facts and concepts that describe aspects of the business and deriving more facts influenced by base facts.

Action assertions are divided into three classes: Authorization, Integrity Constraint, and Condition. An **authorization** action assertion defines the privileges individual users are allowed to perform or execute. For example, only an administrator can reset a customer password. **Integrity constraint** is used to ensure that certain properties to be true all the time, and prohibit any action that can violate these properties. The requirement that a customer be logged-in before placing an order is an example of an integrity constraint. A **condition** action assertion prescribes the circumstances under which a particular action can be performed, such as stipulating that a total order value be greater than $99 before providing free shipping.

2.3.2 Business Rules Mining Approaches

Business rule mining methods can be categorized into three different approaches[45][46]: manual analysis, white-box analysis ,and black-box analysis.

Manual Analysis

In this approach the source code is manually examined to extract business rules. Earls et al. developed a methods for manually extracting business rules from the source code of legacy systems [60]. Their goal was to simplify the process and constraint the focus of the code reader on the relevant parts of the code, such as error handling code and the conditions that leads to those parts.

However, since everything is done manually, the productivity using this approach is proportional to the available manpower and their skills. But it become in-feasible for large and complex systems.

White-box Analysis

White-box approaches employ static analysis of program source code to extract business rules, where the examination process can be semi or fully automatic. Program slicing and pattern matching are the two commonly used methods to extract business processes and business rules. Initially introduced as a means of understanding and debugging programs [46], program slicing concept was adapted to extract business rules. Chiang [61] proposes use of program slicing as a means by which to modernize software by implementing business rules automatically extracted from legacy systems.

Research based on pattern matching focuses on figuring out the structure and the semantic of the code. Shekar et al. [62] define a method that incorporates both application code and database schema as input to a schema extractor algorithm to collect schema information. A semantic analyzer takes schema information in order to discover more semantic elements such as application variables, relationship between variables, application-specific meanings of variables and their relationships, constraints, database entities, relationship between entities and application-specific meanings of entities and their relationships.

White-box analysis has the potential to reveal more business rules than manual analysis thanks to the high level of automation involved, however, it requires source code. If the programmer fails to implement desired behavior, or implements it incorrectly, false or misleading business rules will result.

Black-box Analysis

This approach focuses on analyzing the artifacts generated during application execution, such as HTTP traces logged by a web server. In situations where the source code is not available (e.g. when using third party services), black-box analysis can potentially still analyze the observed behavior and extract useful information. For instance, using only web accesses logs of an e-commerce website, Poggi et al. [58] utilized process mining techniques to extract business models using Business Process Insight (BPI), a process mining platform used to evaluate the performance of different mining algorithms. Crerie ea al. [43] extracted business rules using event logs collected from a simulated information system. Their methodology aims to extract authorization and condition action assertion business rules in an automated fashion. Their solution is applicable for systems that generate logs in forms of transactions and that can be transformed into events in order to be processed using process mining tools.

The percentage of an application's business rules discovered by dynamic analysis is not measurable because there is no way of determining whether the application has been executed sufficiently to reveal all of its business logic. Events relating to exceptional cases, error processing, edge conditions, etc. may not be represented in the log data. This limitation needs to be considered when using black-box approaches.

2.3.3 Business Rule Representation Language

The Semantics of Business Vocabulary and Business Rules (SBVR) standard [86], adapted by the Object Management Group (OMG) in 2008, is the standard representation language for business vocabulary and business rules. SBVR defines the vocabulary and rules for documenting the semantics of business concepts and terms, business facts, and business rules. Documenting business vocabulary and business rules based on SBVR specifications will help removing any ambiguity and enable people within the business domain to easily understand and exchange them among organizations due to the fact that SBVR specifications supports the vocabulary and business rules from an organizational perspective. Also SBVR can be transformed from organizational into IT perspective, which will enable the transformed specifications to be usable in different domains and not limited to business software tools.

Chapter 3

Prepare Web Applications for Process Mining

Existing process mining frameworks (such as ProM, and Disco [63, 64]) offer a wide range of mining algorithms to gain insight into business processes based on observed behavior. Before mining can take place, a data collection framework is needed to capture details about events that take place during process execution. The event logs must also be formatted in a way that fulfills the minimum input requirements of the process mining frameworks. This chapter addresses the problem of capturing and formatting event logs in e-commerce web applications and opens the door for utilizing process mining platforms in discovering more information beyond basic logs in real-world applications.

3.1 Data Requirements for Process Mining

For a successful mining process, the provided data must meet some requirements defined by the mining technology. As mentioned in section 2.2, process mining algorithms rely on event logs consisting of cases that each have a unique identifier. Each case represents an ordered list of zero or more events. Events describes activities that have taken place and consist of least two minimum elements: activity name and date/time of the activity.

The challenge to process mining is that it requires careful attention to ensuring that, first, information about software behavior is captured, and second, that the resulting workflow fulfills the minimum requirements of the process mining frameworks.

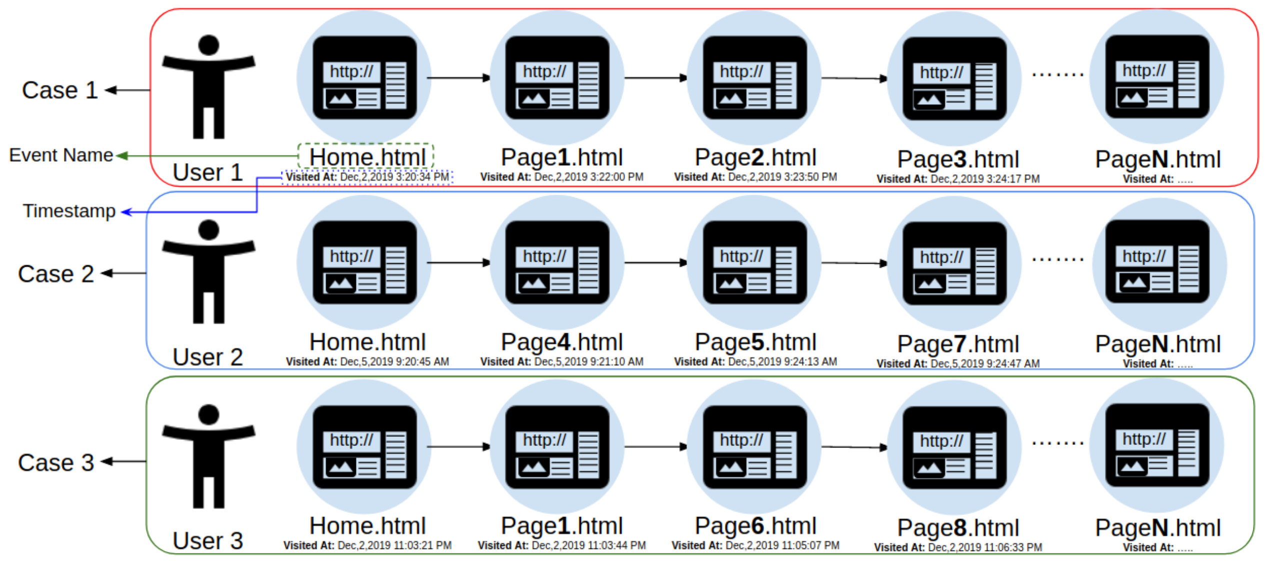

Case 1
Event Name
Timestamp
Case 2
Case 3
User 1
Home.html
Visited At: Dec,2,2019 3:20:34 PM
Page1.html
Visited At: Dec,2,2019 3:22:00 PM
Page2.html
Visited At: Dec,2,2019 3:23:50 PM
Page3.html
Visited At: Dec,2,2019 3:24:17 PM
PageN.html
Visited At: .....
User 2
Home.html
Visited At: Dec,5,2019 9:20:45 AM
Page4.html
Visited At: Dec,5,2019 9:21:10 AM
Page5.html
Visited At: Dec,5,2019 9:24:13 AM
Page7.html
Visited At: Dec,5,2019 9:24:47 AM
PageN.html
Visited At: .....
User 3
Home.html
Visited At: Dec,2,2019 11:03:21 PM
Page1.html
Visited At: Dec,2,2019 11:03:44 PM
Page6.html
Visited At: Dec,2,2019 11:05:07 PM
Page8.html
Visited At: Dec,2,2019 11:06:33 PM
PageN.html
Visited At: .....
http://

Figure 3.1: Converting web application logs to event logs

Capturing and generating workflow logs from web applications is possible because websites are a collection of web resources (pages, media, scripts) that are connected with each other via hyperlinks. Enabling users to easily navigate and use the system. Normally, a user session begins from a landing page (e.g., `index.php`) then navigates the system by executing one page at a time. Based on that, one can create an event log using only the user behavior (as shown in Figure 3.1), where each page can represent activity in the web application process, and the time of the visit can be considered the timestamp. The entire user session can represent a case uniquely identified by the user's Internet Protocol (IP) address.

Web applications can take advantage of a wide range of process mining algorithms because they naturally meet the minimum data requirements for process mining. The question becomes one of capturing and collecting data. The following section addresses this question and discusses the potentially available data sources.

3.2 Web Applications Logging Levels

Web applications are usually configured to collect logs while being executed. They record events triggered by the users' interaction with web applications or a webserver interacting with the operating system. The collected weblog consists of a time series of events, where each entry describes an activity performed. The level of details and the usefulness in the information captured in the weblog can vary, depending on the log maintainer . This section highlights the common logging levels.

3.2.1 Level 1: Access Log

The simplest form of logs (referred to as access logs) is the one maintained by the webserver. Most web servers, including Apache, Tomcat, IIS, and JBoss, support access logs by default [78, 79, 80, 81]. The logs contain basic information about user interactions with the web application, such as the time of a request, the requested page, and the user IP address. Table 3.1 illustrates the general format of Apache default access log.

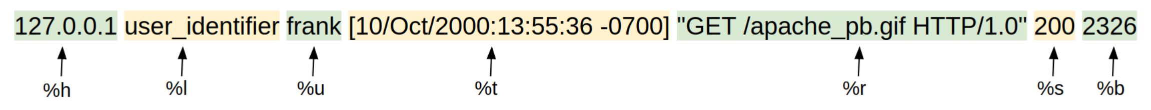
	 127.0.0.1 user_identifier frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
%h	The IP address of the client.
%l	The identity of the client determined by ”identd” on the client’s machine.
%u	The userid of the person requesting the document as determined by HTTP authentication.
%t	The time that the request was received.
%r	The request line from the client http request.
%>s	The status code that the server sends back to the client.
%b	The size of the object returned to the client (not including the response headers).

Table 3.1: Apache common log format

Although, access logs collect this information by default, the level of detail is limited. In the case of POST HTTP requests, the included parameters are attached to the request’s body, which cannot be captured with only the access log. Furthermore, to uniquely identifying users based on the available features in the access log, the user’s IP address and the user-agent information combined can create a potential identifier. Depending on the log maintainer configuration, the user-agent information may not be captured by default (e.g., as in Apache servers), which leaves the IP address as the only way to differentiate between users. This can present difficulties when an intervening web proxy shares a common IP address among multiple users.

3.2.2 Level 2: Full HTTP Traffic Logging

The second-level of logging extends the amount of data captured from simple access logs to the entirety of the HTTP conversation taking place between the users and the web application, including both the request and the response. At this level of logging, much more information can be collected, including HTTP request/response headers and response body, as shown in Table 3.2. Second-level logging produces more data that may not be useful (e.g., .jpg, .mp3, .css. javascript, etc.) and, consequently, requires data filtering and pre-processing to come up with a workable data set.

```
POST /echo/post/json HTTP/1.1
Host: reqbin.com
Content-Type: application/json
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux
Content-Length: 42

{"login":"admin","password":"password123"}
```

```
HTTP/1.1 200 OK
CF-Cache-Status: DYNAMIC
CF-RAY: 558c54b19ef6e841-EWR
Connection: keep-alive
Content-Length: 19
Content-Type: application/json
Date: Tue, 21 Jan 2020 21:20:13 GMT
Expect-CT: max-age=604800, report-uri="h
Server: cloudflare
Set-Cookie: __cfduid=db226096552b9ef5fc4

{"success":"true"}
```

User Request	Server Response

Table 3.2: Post request and response

Level 2 logging can be implemented at different layers of the OSI network model, however, log information from anything below the application layer (Layer 7) typically has limited utility. Because most web traffic takes place over encrypted protocols (i.e., HTTPS), the loggers capturing data from lower layers will yield little useful information since encryption and decryption take place at the application layer. A logger capturing information at the application layer can use a proxy that is pre-configured with SSL/TLS certificates, allowing it to see all the HTTPS conversations. Using such an approach, no additional modifications to the application are required.

3.2.3 Level 3: Custom Application Event Logging

A logging feature can be integrated into the web application source code, allowing it to capture custom logs while being executed. The logs are generated with respect to the application context, which results in higher quality and richness in content when compared with level 2 logging. Such logs lower the level of noise in the captured events as the generated logs are typically focused on the most important aspect of the process and avoid unimportant events. Logging at this level presents a distinct disadvantage in that the application has to be modified to accommodate logging. When compared to level 1 and level 2 approaches, with the two other logging approaches (level 1 & level 2), this one is considered harder to implement and maintain.

3.3 Methodology

In this section, we discuss our proposed solution to capture event logs based on users' behavior when interacting with web applications and convert them into workflow logs format with respect to IEEE XES standard #1849 [85]. Our proposed method consists of five phases: 1) defining a business-centric scope, 2) collecting full HTTP conversations between the clients and the web application, 3) cleaning and segmenting conversations based by users and splitting into sessions, 4) mining event related information, and identify unique activities, and 5) generating workflow logs in XES format.

3.3.1 Scope Definition

Defining the scope of the mining process is an important step because the defined scope should be measurable and driven by business impact. Targeting information that has a business-centric objective (e.g., mining for rules about free shipping eligibility) helps to discover useful business insights, whereas attempting to discover information that is overly broad (e.g., mining for all rules) becomes unproductive.

The focus of this research is on e-commerce web applications, meaning, web services for buying and selling goods over the Internet. We are limiting our scope to traditional (synchronous) web applications that limit the use of asynchronous technology (AJAX calls) when handling user interaction (more in section 3.3.3).

3.3.2 Data Collection

There are multiple levels of logging aiming to record events that happen while executing the system. They differ in several aspects, such as the simplicity of use (e.g., is an extra modification or configration required?), the amount of information they can capture, and the structure of the captured data. We use level 2 as our data source, since the amount of data captured is comprehensive and it does not require any modification to the web application source code, thereby making it an easy-to-adapt approach.

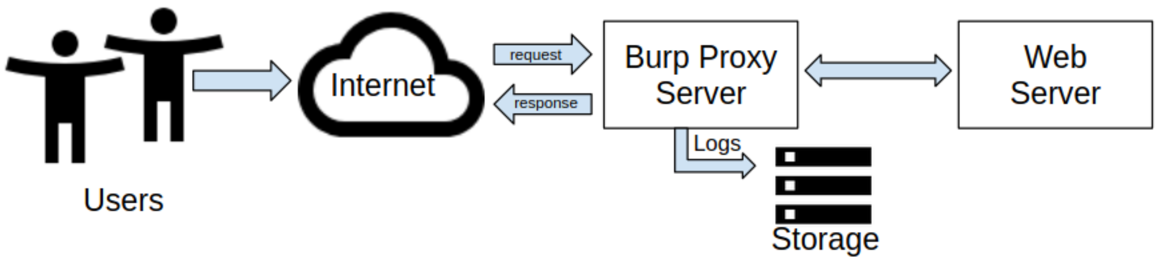


Figure 3.2: Data collection system architecture

To deploy a level 2 logs maintainer capable of collecting full HTTP traces, we used the BurpSuite proxy server [71], which operates at the application layer and can be pre-configured with SSL/TLS certificate to decode HTTPS messages. As shown in figure 3.2, it acts as an intermediate node setting between the web application clients and the webserver. The Burp proxy does not support logging full HTTP traffic by default, so we used Burp extender API to integrate a new extension capable of collecting complete HTTP traces [77]. Righetto D. developed a BurpSuite extension capable of recording HTTP requests [82], which we extended and reused to meet our needs. The extension is loaded into the proxy as an external module and configured to store the captured data into an SQLite database.

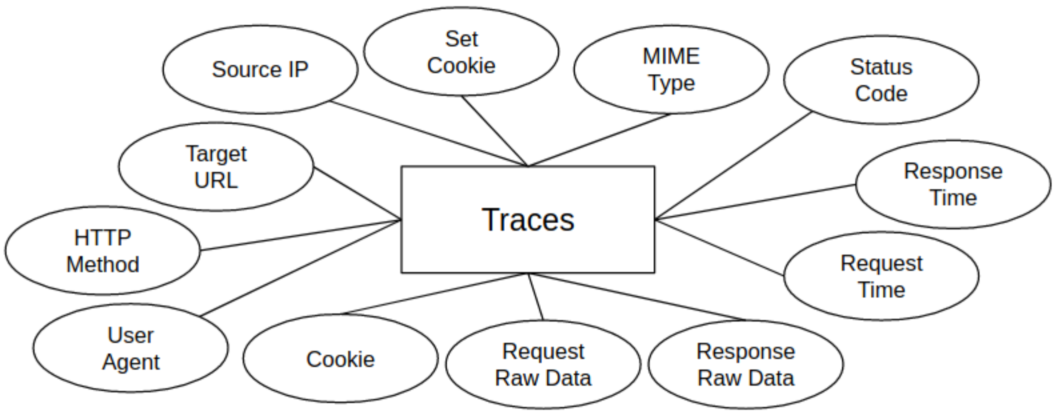


Figure 3.3: Entity relationship diagram (ERD) of the traces database

The extension actively listens to all the incoming and outgoing HTTP/HTTPS conversations based on a predefined web application address. Every time a client requests a page from the targeted web applications, the extension is activated and captures different information such as the client IP address, the time of the request and the response, the raw request and response, as well as other information (see Figure 3.3).

Worth mentioning, some of the information captured by the extension is not supported or accurately indicated in the HTTP protocol, such as the time of the response and the actual content-type in the response. The former is computed based on the arrival time of the web server response, while the latter is generated using BurpSuite build-in API for file format detection.

3.3.3 Data Preprocessing

The effectiveness of the preprocessing phase is one of the factors that greatly affects the quality of the mined process models. A weakly generated workflow log will make it infeasible to generate process models. Thus, it is important to carefully preprocess the captured logs with respect to the data requirements of process mining (see section 3.1) as described below.

Data Cleaning

Web resources in web applications are not limited to static and dynamic HTML documents. They also include other types of resources, such as media files (i.e., images, videos), and

Description	Extension
images	.gif, .jpg, .jpeg, .ico, .png
text	.css, .js, .txt
video/audio	.mp4, .mov, .mp3, .wav
others	.pdf, .doc, .docx, .csv, .xlsx

Table 3.3: Excluded files based on extension

scripts (e.g., JavaScript, Cascading Style Sheets). Such resources are required for web application functionalities and rendering, but they do not represent user behavior since they are static resources requested automatically by the browser. Including them does not add any extra information and increases the complexity of the final log. Table 3.3 shows all the excluded extensions.

Removing the unnecessary files depends on detecting them first, either by checking the `Content-type` header in the response, or the extension of the requested file or by analyzing the content of the response file (a.k.a, Multipurpose Internet Mail Extensions (MIME) sniffing). The former is not a reliable approach because it relies on the server to include the type of content sent through the response, which is not always available or accurate. To ensure accuracy, we use the file extension as an indicator of the content type. If it is not available, we employ MIME sniffing to find it out.

Complicating the data cleaning process, web applications can be build based on AJAX technology to create asynchronous web solutions. Such applications handle user interactions through asynchronous calls that take place in the background between the browser the webserver. AJAX can be used to dynamically keep a page content up to date by pulling data automatically from web servers without the need for user interaction. The use of such technology allows actors other than the actual user to perform events, leading to confusion when it comes to detecting the originator of an event. Solving this problem is beyond the scope of this research; our focus is on synchronous web applications so as to simplify the problem sufficiently to demonstrate proof of feasibility of the overall process mining approach.

The use of AJAX technology is often still present in synchronous web applications. The HTTP protocol does not use any special request format for AJAX calls so they appear as normal HTTP requests. We detect them using a heuristic approach that checks if the header, X_REQUESTED_WITH, (or HTTP_X_REQUESTED_WITH in some browsers) exists in the HTTP request and set to ”xmlhttprequest”. Once discovered, AJAX calls are removed.

User and Session Identification

Having a unique identifier to determine the originator of each HTTP request is valuable information in process mining since multiple users can simultaneously access a web application system, and we wish to track individual interactions. There is no exact solution to this issue if access logs (level 1) are the only data source available, due to the fact that proxy servers, local caches, and firewalls can mask the identity of multiple users using a common resource to access an external web application[53]. Collecting level 2 logs help us to overcome this problem to some degree. We uniquely identify users by their source IP addresses along with `user-agent` header information and session identifier from the user's cookies whenever its available. We acknowledge that this method can fail if the webserver has not yet assigned session IDs for users. Finding a guaranteed solution to this problem is beyond the scope of this research, yet the proposed approach can perform much better than when compared to other approaches based on level 1 logs.

The use of session identifier from users' cookies can help in uniquely differentiate between users, yet such information needs to be tracked and maintained. Session identifiers can change over time, such as when a cookie expires or when a web application issues a new session identifier to a user that logs in. We maintain such information by checking the presence of `Set-Cookie` header in the HTTP responses, which indicates that the webserver is creating or renewing the user's cookies information. In the case of renewing, the old session identifier is replaced with a newer one and we create a relationship between the old and new session identifier to prevent associating it to another user.

After identifying all users present in the data set, we further split the data into sessions per user. In a real-world scenario, users interact with websites at different time intervals [53], which means we cannot just process users' requests as one long chain of accesses and neglect the time factor. Time can be used to unveil interesting relationships: how long an average user spends on each page and the time gap between requests can be used to identify sequential pages (activities).

A time-oriented approach is used to reconstruct user sessions[87]. Based on an inactivity period that measures how long a user was inactive, a new session will be created if the period exceeded a predefined threshold. The inactivity threshold we use is 30 minutes as in [88] and [89].

3.3.4 Advanced Content Mining

One of the main advantages of level 2 logs over level 1 logs is that they are more comprehensive, facilitating collection and extraction of more data from HTTP exchanges. For instance, they can be used to collect user parameters from `GET`/`POST` requests much better that level 1 logs, since level 1 logs are limited to `GET` parameters and do not capture the body of HTTP requests.

Parameter Mining

Parameters in web applications are usually passed through two commonly used HTTP methods `GET` and `POST`. In `GET` requests, the parameters are sent as query string along with the requested page (e.g., `/index.php?`**`id`**`=20&`**`c`**`=h2`), while `POST` requests can pass parameters through the query string and the request body. The structure of the parameters sent using the request body can vary depending on the web application and the complexity of the datatype. For example, some applications may send parameters in a format similar to the simple query string structure, others may pack data in a `JSON` or `XML` object . In this research, our focus is on web applications that use parameters in the form of query string structure in requests.

Algorithm 1: Extract and Set Parameters List in HTTP Messages

```
Result: Initialize each http message with a list of its parameters
Input: http_messags: list of all http messages
1 for msg: http_messags do
2     listOfParams = {};
3     for pair: msg.getRequestParameters() do
4         type = decodeDataType(pair.val());
5         listOfParams.add(type, pair.key(), pair.val()) ;
6     end
7     msg.setParameterList(listOfParams);
8 end
```

As shown in algorithm 1, we create a list of parameters for each HTTP message (request and response) by processing every single HTTP request and extracting its parameters from the query string or/and request body depending on the HTTP method used. We detect the data type for each parameter using `decodeDataType` function, which uses regular expression to detect a variety of data types ranging from simple to complex types, such as, email addresses, currency, percentage, dates, street addresses, alphanumeric, and paragraphs.

Algorithm 2: Eliminate Static or Parameters with High Degree of Uniqueness

```
Result: A list of parameters with accepted degree of uniqueness
Input: http_messags: list of all http messages
allParamsValsMaps = {};
for msg: http_messags do
    for plist: msg.getParameterList() do
        for param: plist do
            if !allParamsValsMaps.contain(param.key()) then
                allParamsValsMaps.createMapNode(param.key());
            end
            allParamsValsMaps.get(param.key()).add(param.val());
        end
    end
end
for node: allParamsValsMaps do
    uniqueCount = countUniqueValues(node.val());
    totalNumOfValues = node.val().size();
    uniquenessPercent = uniqueCount/valSize;
    if uniquenessPercent >= %90 OR uniqueCount == 1 then
        allParamsValsMaps.removeMapNode(node.key());
    end
end
```

Some parameters tend to be static over time, making them less useful when mining for decision logic. For this reason, we eliminate parameter that have a fixed value or that are unique a high percentage of the time. This is done by counting the number of unique values for each parameter and dividing it by the total number of values (see algorithm 2 for more details).

Detecting Unique Activities

In web applications, the name of a web resource is usually a unique identifier. This has the potential to uniquely identify events (activities) names in the workflow we are trying to build. The challenge is that not all web applications follow such URL naming convention. Some web applications, for example, employ query parameters as page identifier (e.g., `/index.php?`**`page=home`**). If such a cause is not recognized, two different events (pages) will be considered the same just because they share the same name. To overcome this problem, our proposed solution relies on measuring the structural similarity between pages, which is possible since HTML pages are structured documents made of HTML elements (i.e., header, body, and div). One can detect similar/different pages based on the assumption that pages with a high level of structural similarity represent the same page type [90], otherwise, pages are most likely to be different (i.e., create account vs. login page).

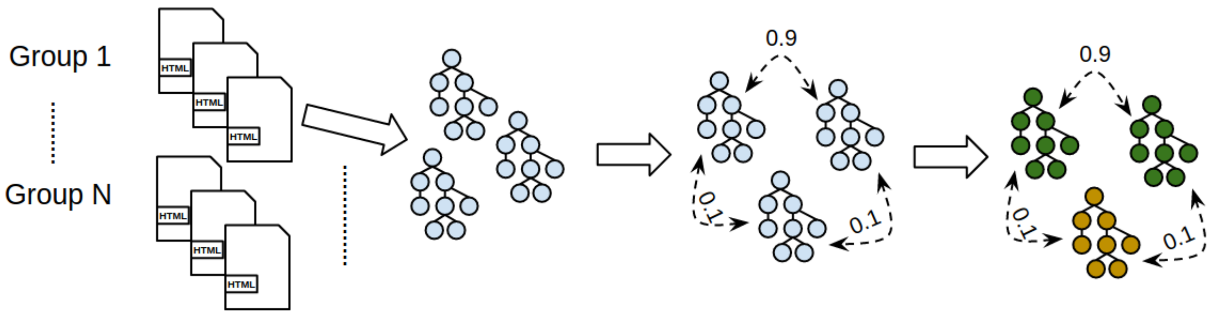


Figure 3.4: Detect different pages with similar names

To measure the structural similarity among pages, we place all sets of instances (pages) sharing the same name into separate groups, after which we convert all the HTML instances in each group into a trees structure by traversing over the tags in each page and skipping unrelated tags (e.g., comments, meta, etc.). Bracket representation is used to construct the trees (i.e., ABC is a tree where A is the root with two child nodes B and C). We compute the edit distance between pages in each group using the APTED algorithm (a memory-efficient solution compared to RTED for computing the tree edit distance [91, 92]) with a fixed cost model of 1 for both node insertion and deletion. Based on the computed edit distance between every pair of trees, we calculate their similarity ratio using equation (3.1). We repeat the same process between all pairs in each group to construct an `NxN` similarity matrix for each group, where N is the number of instances per group.

$$similarity\_ratio = \frac{MaxTreeSize - distance}{MaxTreeSize} \times 100, \tag{3.1}$$

where

$$MaxTreeSize = Max(tree1.size(), tree2.size())$$

$$dissimilarity\_matrix = 100 - similarity\_matrix \tag{3.2}$$

We then compute the dissimilarity matrix (as shown in equation (3.2)) for each group, and use it to measure the average dissimilarity to detect groups that are experiencing a high level of dissimilarity based on a predefined threshold. We compute the average dissimilarity using equation (3.3), which is the sum of all dissimilar pairs divided by the number of pairs per group. For groups that exceed the threshold, we apply the agglomerative hierarchical clustering (with average linkage strategy) algorithm on the dissimilarity matrix to cluster different pages into subgroups. For example, if a group has 3 pages where two of them are very similar and the third page is not, the resulting dendrogram after clustering will consist of two main sub-clusters one linking the similar pages and another for the different page. After discovering all the sub-groups we rename pages based on the main group name plus the sub-group index they belong to (i.e., if the group name is index.php then the sub-groups will be index.php-0, index.php-1 ...etc).

$$average\_dissimilarity = \frac{\sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} dissimilarity\_matrix[i][j]}{\frac{N(N-1)}{2}}, \tag{3.3}$$

where N is the number of instance per group

3.3.5 Generating Workflow Logs

At this phase, processed HTTP traces are ready to be converted into workflow event logs represented, in our case, in XES format. The next step is to map users' traces and sessions into a stream of cases and events. The mapping for some elements is straightforward (i.e., HTTP request time $\rightarrow$ event timestamp); other elements can have multiple configurations/options when it comes to mapping (e.g., activity/event name). In this section, we highlight some of these elements and the different mapping options available.

One-To-One Mapping

Elements in the workflow log are considered to have a one-to-one relationship if they are mapped only to a single element in the HTTP traces. Table 3.4 shows a list of workflow elements with a one-to-one relationship and also notes that each parameter in the HTTP request (if it exists) is also mapped to a single element in the workflow.

Workflow Elements	**Mapping Type**	**HTTP Elements**
event:time:start_timestamp	one-to-one	request time
event:time:complete_timestamp	one-to-one	response time
event:status_code	one-to-one	response status code
event:set-cookie	one-to-one	set-cookie header
event:referer	one-to-one	referer header
Additional Parameters		
event:parameter_1	one-to-one	parameter 1
event:parameter_2	one-to-one	parameter 2

Table 3.4: Elements with one-to-one mapping relationship

One-To-Many Mapping

A one-to-many relationship exists if an element in the workflow log can be mapped to one or more elements (at the same time) from the HTTP trace. As shown in table 3.4, several elements in the workflow log can be constructed using a combination of HTTP elements. In our proposed solution, if both one-to-one and one-to-many relationships exist for an element in the workflow logs, the latter is favored. The reset of the section highlights such elements and discusses the advantages of one-to-many mapping.

The case id attribute is one example of such elements, which is a unique identifier assigned to all the cases in the workflow log. By mapping it to three different HTTP elements (IP, user-agent, and session-id), we can guarantee the uniqueness of the attribute, which is not the case when compared to one-to-one mapping (i.e., case:id $\rightarrow$ IP-address) relationship. However, since cookies information may not be always available (see section 3.3.3), that leaves us with only the user's IP address and `user-agent` header information, which can cause duplicate case ids. To deal with this problem, we append the ordering number of the duplicate cases to the id attribute. Similarly, the resource attribute (the user performing the event) in events is defined with the same three elements as in the id attribute, except that duplication is allowed here since the same user can show-up in different cases.

Workflow Elements	**Mapping Type**	**HTTP Elements**
case:concept:name	one-to-one	IP Address
	one-to-many	IP Address + User Agent
	one-to-many	IP Address + User Agent + Cookie:Session_ID
event:concept:name	one-to-one	Page Name
	one-to-many	Page Name + HTTP Method
	one-to-many	Page Name + HTTP Method + Status Code
event:org:resource	one-to-one	IP Address
	one-to-many	IP Address + User Agent
	one-to-many	IP Address + User Agent + Cookie:Session_ID

Table 3.5: Elements with one-to-many mapping relationship

Another interesting element is the activity (event) name, which is a workflow element that can be mapped to one or more elements in the HTTP traces. It can be directly mapped to the page name, which represents a good candidate to describe the event that was executed as proposed in [58]. However, relying only on the page name to capture the performed event can deliver an incomplete picture of what happened. Because users can interact with the same page differently, as it may support more than one HTTP method (i.e., GET, POST and HEAD). Also, pages can behave differently based on the user state and input, which is clearly missing when only using the page name. For example, multidimensional (one-to-many) task/activity IDs enable us to extend the state-space of each task from $\binom{1}{1}$ to up to $\binom{252}{1}$ (depending on the tested applications), allowing to precisely express task's state.

3.4 Evaluation

To evaluate our proposal, we used open-source online store management web applications tested by real users (as intended) to increase code coverage and to expose most of the supported roles and functionalities. Allowing us to generate in-depth HTTP traffic based on a real-world e-commerce web application, from which we could extract information based on actual user behavior.

The rest of the experiment is organized as follows, section 3.4.1 discusses the set up of our experiment. Section 3.4.2 talks about our data collection approach and show some results. Section 3.4.3 demonstrates how we detected pages sharing similar names but different structures. Section 3.4.4 compare different mapping setups and discuss their advantages. Lastly, section 3.4.5 evaluates the quality of our generated workflow logs.

3.4.1 Experiment Setup

The experiments involved two main components, the web proxy and the webserver running an e-commerce web application. The users were expected to interact with the web application were assigned unique IP addresses and configured to use the proxy, which guarantees that all HTTP requests and responses will go through the web proxy.

A Linux box running Ubuntu 18.04 was used to run the web proxy, which was based on Burp suite proxy the community version. The proxy listened on all interfaces at port 8080 for incoming connections and was configured to run our HTTP traffic collection extension (see section 3.3.2) and store the collected data in an SQLite database on the Linux machine local storage.

The criteria for selecting web applications to be used in this experiment was based on three requirements: 1) the application had to be used by a large community, 2) it had to be categorized as ”e-commerce”, and 3) it had to limit the use of AJAX. The web application of choice that met our selection criteria was OsCommerce version 2.3.4.1 (released in August 2017). It is an open-source e-commerce and online store management system used in more than 20k websites. To locally host it, we used a Raspberry Pi 3 B+ running an Apache web server along with a MySQL 5.0 server and PHP version 5.0. The web server was connected to the same local area network used by the proxy server, allowing the latter to easily forward incoming requests.

Lastly, we used ProM 6.8 as our process mining platform, which is an open-source framework for process mining algorithms. We used it in our experiments for conducting process discovery, and help in evaluating the quality of the generated workflow logs through conformance checking.

3.4.2 Data-set

To evaluate our proposed data collection approach, we set up and configured a web server and a proxy running in a closed network environment. We asked real users (connected through our proxy) to interact with a locally hosted OsCommerce website. Each user was assigned to test a specific role in the system, where the support roles are guest, logged-in user, and admin. Table 3.6 shows the total number of collected HTTP requests before and after cleaning the data by removing unnecessary requests.

Role	**Raw HTTP Requests**	**Cleaned (without Media)**
Admin User	297	237
Logged-in User	666	401
Guest User	416	225
Total	**1379**	**863**

Table 3.6: Total number of collected HTTP request before and after cleaned

The collected data-set shows that our approach successfully was able to capture the entire HTTP conversations (requests and responses) and handled multiple users communicating at the application layer, without interfering with the application functionalities or causing any bottleneck.

3.4.3 Detecting Structurally Different Pages

In this experiment we detect pages that shared similar names but have different HTML structures. The experiment used the cleaned subset of the data-set generated in 3.4.2, which consisted of 863 HTTP requests and responses. Removing HTTP responses that did not include HTML documents, such as page redirection requests (status code 302) and page-not-found responses (status code 404), reduced the data set to 709 requests. For illustrative purposes, we will use the Logged-in users sub-data set. After computing the average dissimilarity per group (as shown in figure 3.5) with a 20% threshold, we can see that only the `index.php` page exceeded our threshold with an average dissimilarity around 24.

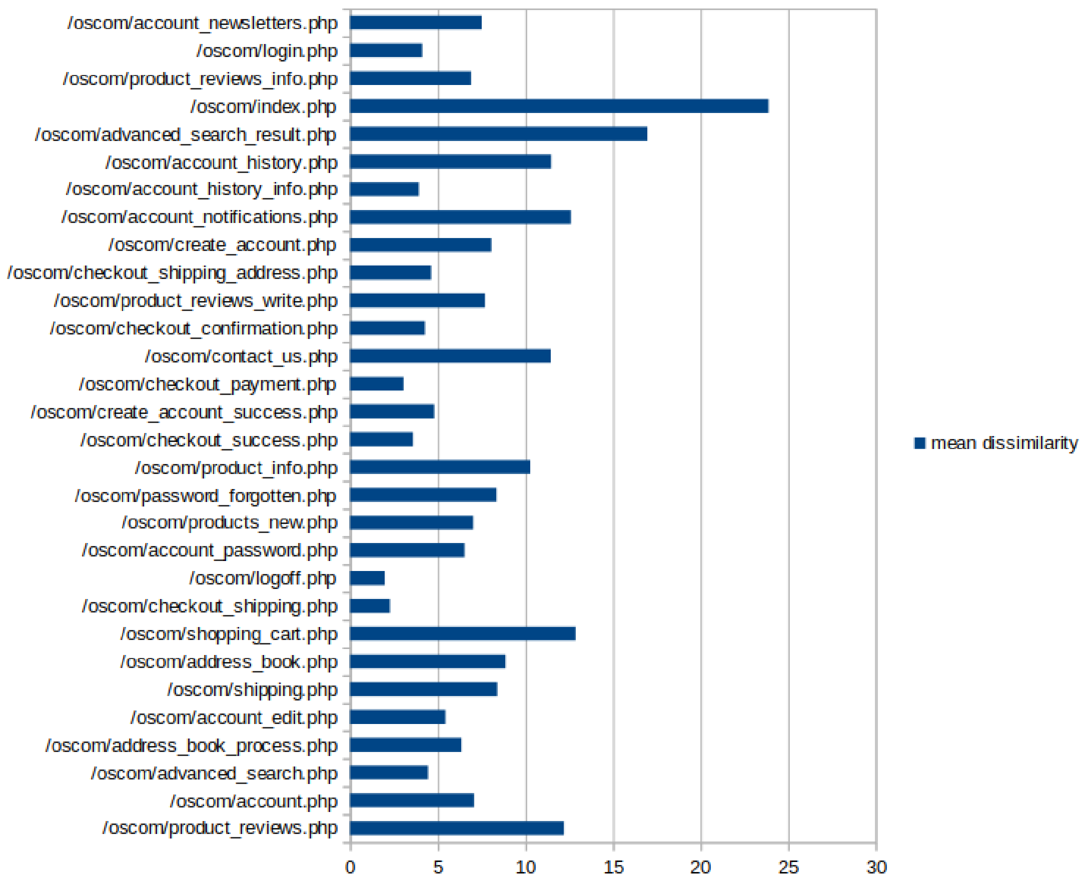


Figure 3.5: Average dissimilarity per group based on the logged-in users sub-data set

Based on the dissimilarity average threshold, the *index.php* group was selected as a potential group with structurally different pages. Figure 3.6 shows the resulted dendrogram after clustering all the instances in the *index.php* group based on the dissimilarity matrix. Three main clusters result when we manually cut the dendrogram at a dissimilarity level of 24, indicating the possibility of three different categories of pages were present in the group based on their structure. Examination of the instance names per cluster shows that two clusters have the `cPath` parameter, while it is not there in the third cluster. We can see that the `cPath` parameter value format differ between the two clusters, one is an integer and another consists of two integers concatenated with a symbol (i.e., 3_13).

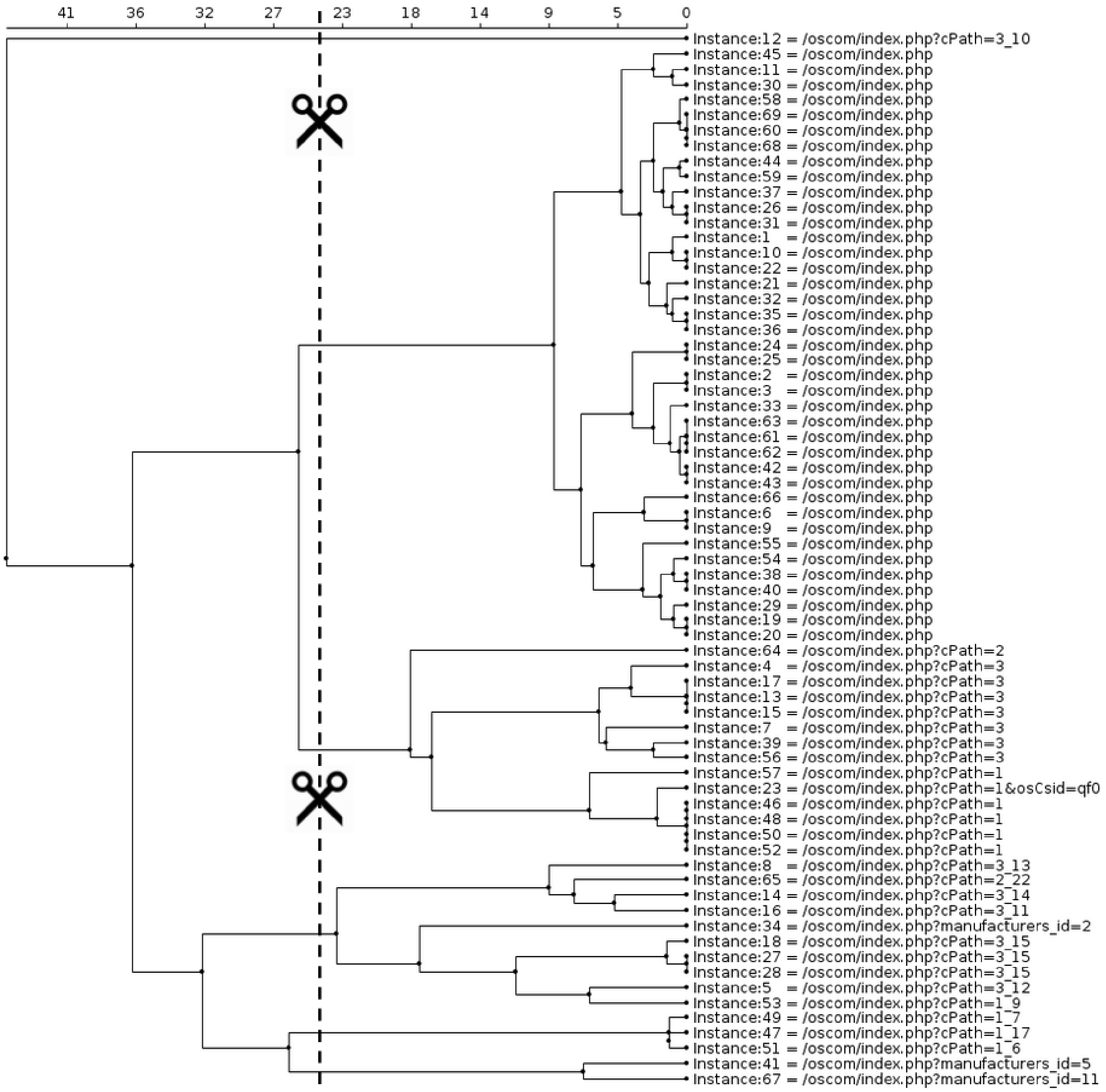
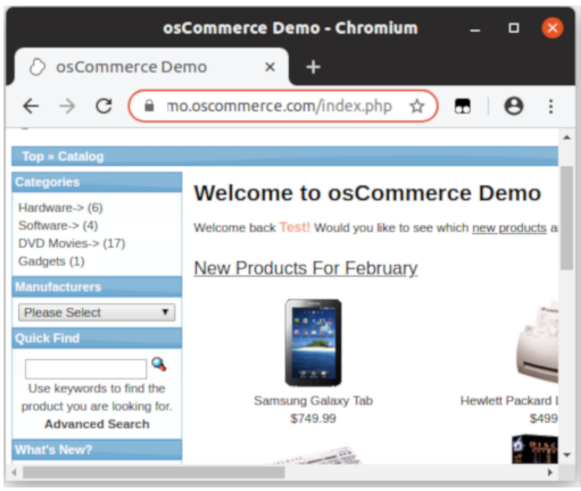
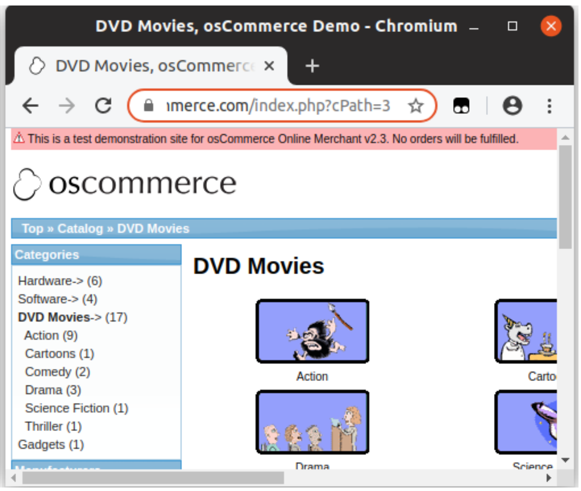

41
36
32
27
23
18
14
9
5
0
Instance:12 = /oscom/index.php?cPath=3_10
Instance:45 = /oscom/index.php
Instance:11 = /oscom/index.php
Instance:30 = /oscom/index.php
Instance:58 = /oscom/index.php
Instance:69 = /oscom/index.php
Instance:60 = /oscom/index.php
Instance:68 = /oscom/index.php
Instance:44 = /oscom/index.php
Instance:59 = /oscom/index.php
Instance:37 = /oscom/index.php
Instance:26 = /oscom/index.php
Instance:31 = /oscom/index.php
Instance:1 = /oscom/index.php
Instance:10 = /oscom/index.php
Instance:22 = /oscom/index.php
Instance:21 = /oscom/index.php
Instance:32 = /oscom/index.php
Instance:35 = /oscom/index.php
Instance:36 = /oscom/index.php
Instance:24 = /oscom/index.php
Instance:25 = /oscom/index.php
Instance:2 = /oscom/index.php
Instance:3 = /oscom/index.php
Instance:33 = /oscom/index.php
Instance:63 = /oscom/index.php
Instance:61 = /oscom/index.php
Instance:62 = /oscom/index.php
Instance:42 = /oscom/index.php
Instance:43 = /oscom/index.php
Instance:66 = /oscom/index.php
Instance:6 = /oscom/index.php
Instance:9 = /oscom/index.php
Instance:55 = /oscom/index.php
Instance:54 = /oscom/index.php
Instance:38 = /oscom/index.php
Instance:40 = /oscom/index.php
Instance:29 = /oscom/index.php
Instance:19 = /oscom/index.php
Instance:20 = /oscom/index.php
Instance:64 = /oscom/index.php?cPath=2
Instance:4 = /oscom/index.php?cPath=3
Instance:17 = /oscom/index.php?cPath=3
Instance:13 = /oscom/index.php?cPath=3
Instance:15 = /oscom/index.php?cPath=3
Instance:7 = /oscom/index.php?cPath=3
Instance:39 = /oscom/index.php?cPath=3
Instance:56 = /oscom/index.php?cPath=3
Instance:57 = /oscom/index.php?cPath=1
Instance:23 = /oscom/index.php?cPath=1&osCsid=qf0
Instance:46 = /oscom/index.php?cPath=1
Instance:48 = /oscom/index.php?cPath=1
Instance:50 = /oscom/index.php?cPath=1
Instance:52 = /oscom/index.php?cPath=1
Instance:8 = /oscom/index.php?cPath=3_13
Instance:65 = /oscom/index.php?cPath=2_22
Instance:14 = /oscom/index.php?cPath=3_14
Instance:16 = /oscom/index.php?cPath=3_11
Instance:34 = /oscom/index.php?manufacturers_id=2
Instance:18 = /oscom/index.php?cPath=3_15
Instance:27 = /oscom/index.php?cPath=3_15
Instance:28 = /oscom/index.php?cPath=3_15
Instance:5 = /oscom/index.php?cPath=3_12
Instance:53 = /oscom/index.php?cPath=1_9
Instance:49 = /oscom/index.php?cPath=1_7
Instance:47 = /oscom/index.php?cPath=1_17
Instance:51 = /oscom/index.php?cPath=1_6
Instance:41 = /oscom/index.php?manufacturers_id=5
Instance:67 = /oscom/index.php?manufacturers_id=11

Figure 3.6: Dendrogram of the index page instances based on logged-in user role sub-data set

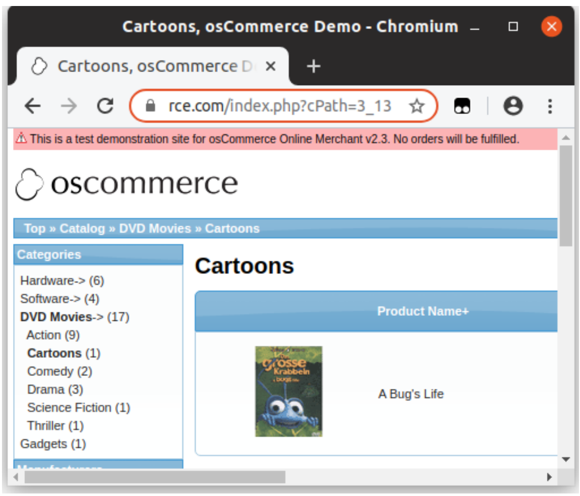
We verified this by examining the HTML documents in each cluster, where we saw three different categories of pages exits. Instances without the `cPath` parameter represented the website landing page (see figure a-3.7) and instances with a `cPath` parameter set to integer formated values represent a product category page(see figure b-3.7). Instances with the `cPath` parameter set to a combined integer values (see figure c-3.7) represent sub-categories.


osCommerce Demo - Chromium
osCommerce Demo
mo.oscommerce.com/index.php
Top » Catalog
Categories
Hardware-> (6)
Software-> (4)
DVD Movies-> (17)
Gadgets (1)
Manufacturers
Please Select
Quick Find
Use keywords to find the product you are looking for.
Advanced Search
What's New?
Welcome to osCommerce Demo
Welcome back Test! Would you like to see which new products a
New Products For February
Samsung Galaxy Tab
$749.99
Hewlett Packard
$499

(a) Index page without cPath parameter



(b) Index page with cPath parameter set to 3



(c) Index page with cPath parameter set to 3_13

Figure 3.7: Comparing the content of index pages with high structural dissimilarity

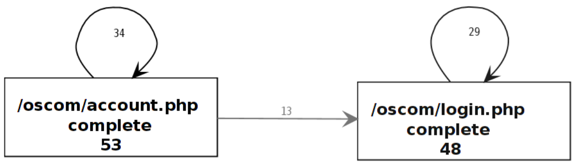
3.4.4 One-to-one vs. One-to-Many Mapping

The event/activity name is one of the elements that can be constructed by either mapping it to one or more elements from the HTTP trace. We used the data-set collected in section 3.4.2 to compare the number of event classes captured by both setups. Table 3.7 shows that the one-to-many mapping approach was able to capture much more behavior, as the number of event classes (types) increased over 57% in the overall data-set. This also indicates the majority of pages actually can represent more than one event, and ignoring that will drop a big chunk of user's behavior.

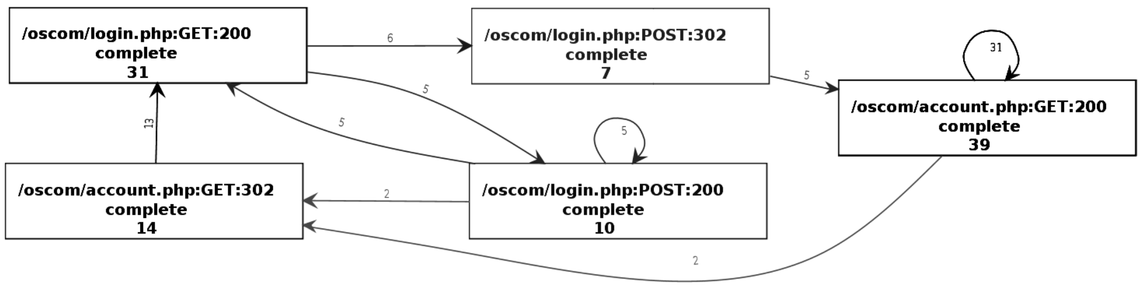
Role	**# of event classes based on one-to-one mapping**	**# of event classes based on one-to-many mapping**	**Percentage Change**
Admin User	32	49	53.13%
Logged-in User	37	62	67.57%
Guest User	23	34	47.83%
Total	92	145	57.61%

Table 3.7: One-to-one vs one-to-many mapping

Figure 3.8 shows the discovered heuristic nets based on traces of two pages the `login.php` and `account.php`, which were discovered using *heuristic miner* [95], a mining algorithm for discovering dependencies between events. The mined models demonstrate the difference in complexity. Specifically, figure a-3.8 consists of only two events only, indicating that the dependency connection between the events indicates that the `login.php` page directly flows the `account.php` page, which is not entirely accurate. Figure b-3.8, on the other hand, provides a much more accurate picture of the actual dependencies. The heuristic net consists of five different classes, two based on the `account.php` page and three are based on the `login.php` page. In this model, we can see that the only way to access the account (`account.php:GET:200`) page is through logging successfully through the login page(`login.php:POST:302`). Moreover, the relationship between `account.php` and `login.php` we saw in figure a-3.8 is actually referring to the relation "`account.php:GET:302` followed by `login.php:GET:200`", which happens when a non-logged-in user tries to access the account page.



(a) One-to-One Mapping


/oscom/login.php:GET:200
complete
31
6
/oscom/login.php:POST:302
complete
7
5
31
/oscom/account.php:GET:200
complete
39
5
5
13
5
/oscom/account.php:GET:302
complete
14
2
/oscom/login.php:POST:200
complete
10
2

(b) One-to-Many Mapping

Figure 3.8: Comparing the heuristic net when mapping the event name attribute of the login & account pages differently

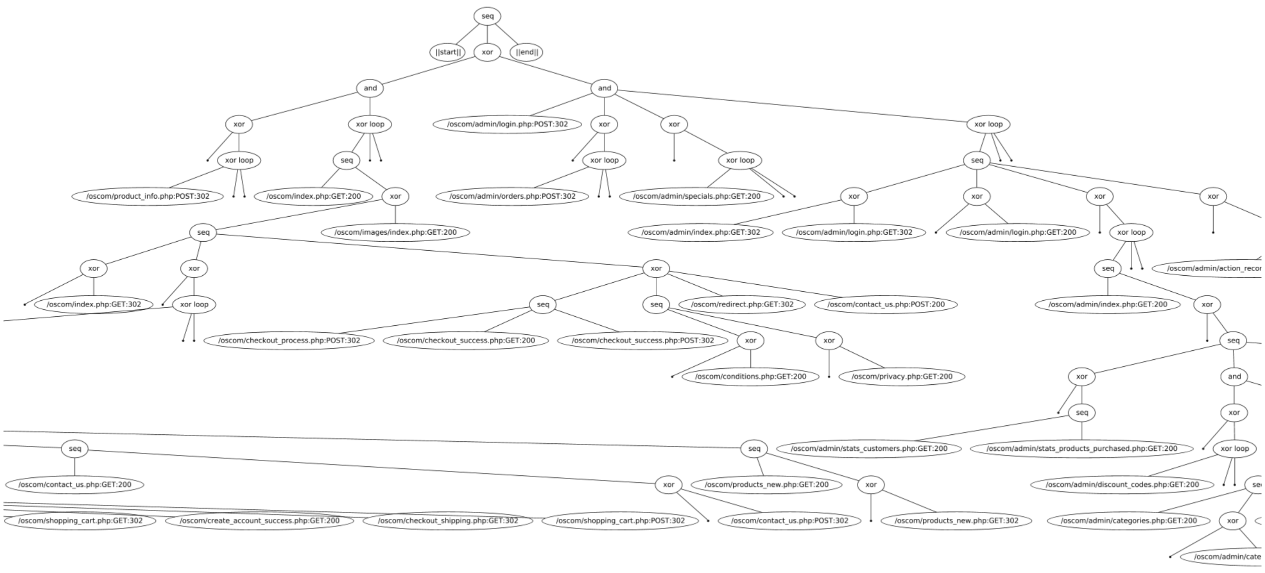
Mapping workflow elements to multiple related attributes in the HTTP trace helps in discovering more specific process models. While one-to-one mapping relationships are more abstract since they capture less behavior.

3.4.5 Generated Workflow Logs (XES)

The quality of discovered process models significantly relies on the quality of the workflow logs fed to the process mining algorithms [100, 101]. Therefore, we use the quality of the discovered models as an indicator to evaluate the performance of our proposed solution in generating the workflow logs. The quality is computed using alignment-based conformance checking to measure how the discovered model aligns with the observed behavior. Moreover, to limit the effect of other factors on the quality of the discovered models, a sophisticated process mining algorithm that guarantees sounds models was used.

The overall quality of the discovered process models is computed based on four different criteria, fitness, precision, simplicity, and generalization [93]. The fitness measures the extent to which the observed behavior (workflow logs) can be replied in the discovered model. The precision quantifies the amount of behavior allowed other than the observed. Simplicity captures the complexity of the discovered model. While the ability to reproduce unseen (future) behavior defines how generalized the discovered model is. Taken collectively, these four quality dimensions all together control the quality; focusing on one criterion is not sufficient. For example, a flower process model guaranties a perfect replay fitness, but it is too generalized and to be suitable discriminant of behavior, and is thus weak on precision. In this evaluation, we use two quality measures, fitness and precision.

Several mining algorithms are used for process discover, such as α-algorithm, fuzzy miner, heuristic miner, and inductive miner, and evolutionary tree miner [94, 96, 95, 97, 98]. However, they vary in different aspects, such as the ability to handle loops, the representation language used to generate models, the sounds of the model, and the time needed. The mining algorithm we chose is *inductive miner* becuase it is guaranteed not to deadlock, it balances quality dimensions, and it discovers models in a reasonable time.



seq
||start||
xor
||end||
and
xor loop
/oscom/admin/login.php:POST:302
/oscom/product_info.php:POST:302
/oscom/index.php:GET:200
/oscom/admin/orders.php:POST:302
/oscom/admin/specials.php:GET:200
/oscom/images/index.php:GET:200
/oscom/admin/index.php:GET:302
/oscom/admin/login.php:GET:302
/oscom/admin/login.php:GET:200
/oscom/index.php:GET:302
/oscom/redirect.php:GET:302
/oscom/contact_us.php:POST:200
/oscom/admin/index.php:GET:200
/oscom/checkout_process.php:POST:302
/oscom/checkout_success.php:GET:200
/oscom/checkout_success.php:POST:302
/oscom/conditions.php:GET:200
/oscom/privacy.php:GET:200
/oscom/admin/stats_customers.php:GET:200
/oscom/admin/stats_products_purchased.php:GET:200
/oscom/contact_us.php:GET:200
/oscom/products_new.php:GET:200
/oscom/admin/discount_codes.php:GET:200
/oscom/shopping_cart.php:GET:302
/oscom/create_account_success.php:GET:200
/oscom/checkout_shipping.php:GET:302
/oscom/shopping_cart.php:POST:302
/oscom/contact_us.php:POST:302
/oscom/products_new.php:GET:302
/oscom/admin/categories.php:GET:200

Figure 3.9: Part of the discovered process tree

The process tree shown in figure 3.9 was discovered from the generated workflow logs, using the inductive miner (IM) algorithm in ProM. To measure the fitness and precision of the discovered tree, we applied the alignment-based conformance checking plugin (Compute projected fitness and precision [99]) in ProM on the process tree and the event logs as input. As shown in figure 3.10, we achieved a perfect fitness and around 0.74 for precision.

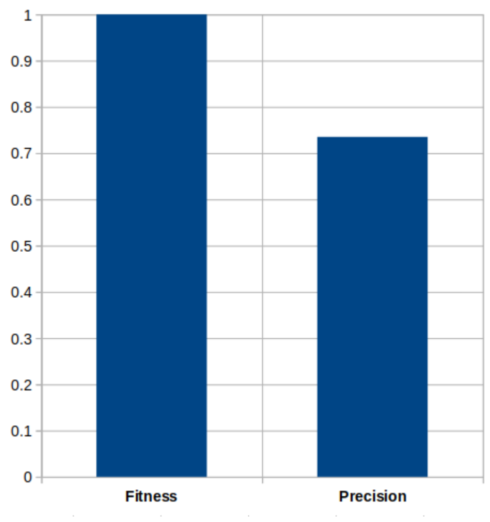


Figure 3.10: Fitness and precision score

Due to the fact that the size of our available data-set was relatively small, which affected the over all precision score. A larger data-set will add more stability to the discovered models, therefore result in better precision as more behavior will be captured.

Chapter 4

Discovering Authorization Business Rules

Advances in information technologies have allowed businesses to deliver their services to a wider range of users, affording companies the opportunity to create new markets that did not exist before, especially in fields such as e-commerce, healthcare, e-education, and cloud services. Web technologies, in particular, have enabled the development of platform-independent systems that can be reached in a few clicks from any web browser.

As web applications grow in terms of features and popularity, their complexity also increases accordingly. One aspect of such complexity is role management, which mainly deals with defining behavioral expectations and maintaining those expectations over time. Roles are defined based on the authorization business rules, as these types of rules formally specify what each role is allowed to do. Keeping track of up to date authorization rules becomes a challenge due to the rapid changes in business requirements. Missing or outdated authorization business rules limit the ability to determine if the current state of the system complies with its expected behavior. Further, weak role management has the potential to devolve into conflicting roles during system updates.

To address this, we propose a black-box approach to discover authorization business rules in web applications through captured HTTP traffic. The proposed solution uses a process mining framework and custom mining algorithms to discover the existing roles and what they are authorized to do. The rest of the chapter is organized as follows: Section 4.1 and 4.2 discuss the action assertion business rules and permission management in web applications respectively. Our proposed solution is discussed in section 4.3. In section 4.4 we evaluate our proposal based on several criteria.

4.1 Authorization Action Assertion Business Rules

An *action assertion* is one of three business rules categories defined by the Business Rules Group [42] (more in Section 2.3.1) to define business constraints and conditions, and thus control business behavior. Action assertion rules are subcategorized into authorization, integrity constraint, and condition business rules. Our focus in this chapter is on discovering authorization business rules, meaning, rules which constraint the actions a user may execute.

Authorization business rules are defined in the following format [42]: `(Only) X may do Y`, where `X` represents a user and `Y` the action that may be performed. We suggest that users observed performing a common set of actions belong to a common role. A small set of observations provides only enough information to identify vague and general roles; however, roles become more defined and refined as the number of observations increases due differentiated behavior.

4.2 Authorization in Web Application

The need for a permission management systems emerged with the popularity of client-server applications offering public services to a wide range of users on the Internet. One commonly-used mechanism in web applications is role-based access control (RBAC), which is a security approach for defining and controlling users' privileges. With RBAC, each role consists of a list of permitted operations. Users assigned to a role are authorized to only perform system functions permitted by that particular role.

When such role-based approaches are enforced in web applications, user behavior and site accessibility/reachability is limited accordingly. As the number and the type of permitted operations vary based on roles, users within the role perform a finite number of operations. Overlap in user behavior and access from different roles is likely because it is common in web applications to share access to resources, however, the effect of the overlap diminishes over time as more data is captured. For a dynamic-based approach to work, it is important to capture enough data to correctly reconstruct user roles.

4.3 Methodology

In this section, we discuss our approach for discovering authorization business rules based on observing the behavior of web applications while under normal use. The approach uses different profiling techniques to learn potential roles through a hierarchical clustering technique configured to be compatible with the format of the datasets used. Using the discovered clusters we then compute the optimal number of existing roles to generate the final list of authorization business rules per role.

4.3.1 Constructing Users' Profiles

Users' access and behavior in web applications are controlled by the role to which they belong. We employed two profiling approaches to mine roles from web application HTTP traffic. The first approach looks into the user's behavior in terms of the activities that were performed. The second focuses on the level of access a user has to system functionality per session.

Behavior-Based Profiles

Users who belong to the same role will behave similarly to some degrees because they are authorized to access a predefined list of functionalities. For example, successfully logging into the system is a common behavior among users that belong to the registered users' role. We can detect roles by grouping users based on behavior. We use the sequence of web requests per session generated by a user when interacting with a web application (per session) to create a behavior-based profile.

Case ID	Activity	Resource	Timestamp
U1	index.php	Sam	01-15-2020 03:01:00
U1	login.php	Sam	01-15-2020 03:01:50
U1	login_success.php	Sam	01-15-2020 03:02:40
U1	account.php	Sam	01-15-2020 03:02:45
U1	logout.php	Sam	01-15-2020 03:04:25
U2	index.php	Guest	01-11-2020 21:17:23
U2	contact_us.php	Guest	01-11-2020 21:19:38
U2	contact_us_success.php	Guest	01-11-2020 21:23:56
U2	index.php	Guest	01-11-2020 21:24:00
U3	index.php	Alice	01-13-2020 16:51:09
U3	login.php	Alice	01-13-2020 16:51:49
U3	login_success.php	Alice	01-13-2020 16:53:02
U3	account.php	Alice	01-13-2020 16:53:10
U3	update_email.php	Alice	01-13-2020 16:56:33
U3	account.php	Alice	01-13-2020 16:56:38
U3	logout.php	Alice	01-13-2020 16:58:49

Table 4.1: A workflow log for a simple web application

The workflow log in Table 4.1 shows a sample of activity on a simple shopping website. The log consists of three cases that represent three different users, where each case contains a list of the performed activities ordered in a timely fashion. To create profiles based on the performed activities per case, we first collect all the unique activities (UA) in the workflow log. We then create a binary profile of size $|UA|$ per case (Table 4.2) to record which activities each user visited and which activities were not visited (1:yes, 0: no).

Activities	Session ID		
	U1	U2	U3
index.php	1	1	1
login.php	1	0	1
login_success.php	1	0	1
account.php	1	0	1
logout.php	1	0	1
contact_us.php	0	1	0
contact_us_success.php	0	1	0
update_email.php	0	0	1

Table 4.2: Behavioral based profile for the log in Table 4.1

Reachability Based Profiles

Relying only on user behavior to discover roles is insufficient because users from different roles may share access to common functionalities and hence behave similarly. To address that, we add an extra profiling technique to capture not only what the users did but also what they can do. To create such profiles, we expanded the dataset in Table 4.1 to include the HTML document of each activity per case. We extracted all hyperlinks from each document to inventory additional system functions users could access, even though they may not have been observed to have accessed them. These links represent features that change based on the user's role [18]. We used the extracted links to creat a reachability map per case, repeating the same process to create maps for all cases. Figure 4.1 shows a map of all reachable pages at each performed activity in case U3.

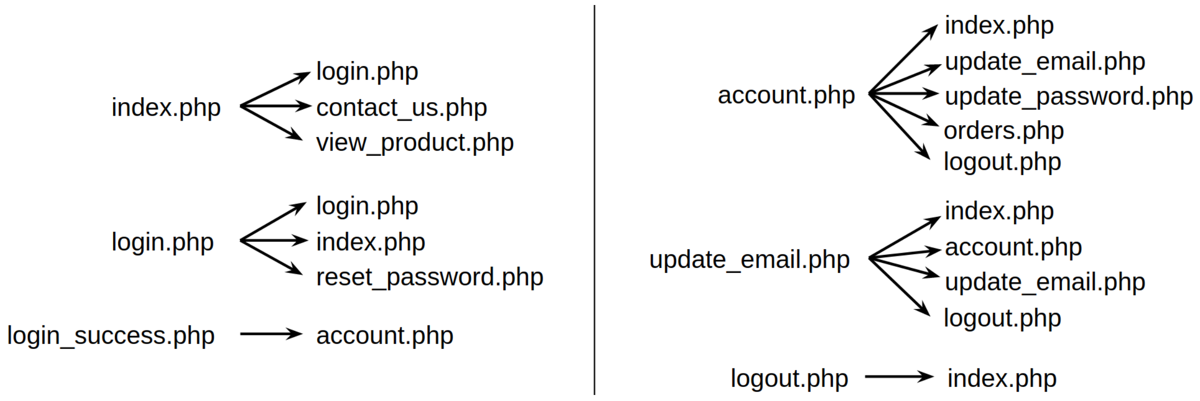


Figure 4.1: Reachable pages based on case id U3

We used the reachability maps to build a list of unique activities (*RUA*) which users had access to. Similarly, we created binary profiles of size $|RUA|$ per case to capture reachable activities for each user. Table 4.1 shows the resulting reachability based profile, which clearly reveals the similarity between case U1 and U3 and how different they are from case U2.

Activities	Session ID		
	U1	U2	U3
login.php	1	1	1
contact_us.php	1	1	1
view_product.php	1	1	1
index.php	1	1	1
reset_password.php	1	0	1
update_email.php	1	0	1
update_password.php	1	0	1
orders.php	1	0	1
logout.php	1	0	1
account.php	1	0	1

Table 4.3: Reachability based profile for the log in Table 4.1

4.3.2 Measuring the Distance

As mentioned earlier, users from the same role share common behavior and access level. To detect such similarity, we computed the distance between instances in the dataset. Selecting the right distance function proved to be a very important step, as it will influence the clustering algorithm outcome. The classic distance functions such as Euclidean and Manhattan are not appropriate for non-continuous, binary datasets, whereas other options, such as Jaccard and Hamann, are much more suitable for measuring the distance of binary variables [102, 103]. Features with zero variance were eliminated as they do not add any useful information in the clustering process.

Our profile dataset consists of binary features, where 1 (positive) means the user visited or had access to a particular page and 0 (negative) indicates the opposite. We are only interested in access, so all possible outcomes are not equally important.. Such binary variables are called asymmetric attributes, and require special handling when computing the distance as only one state of the variable is considered.

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} \tag{4.1}$$

$$d_J = \frac{M_{11} + M_{10}}{M_{01} + M_{10} + M_{11}} = 1 - J \tag{4.2}$$

The Jaccard coefficient (Eq. (4.1)) was used to measure the similarity between cases in the dataset, as it is capable of handling asymmetric binary attributed. To compute the distance we deduct one from the similarity ratio as in Eq. (4.2).

$$D = \sum_{i=1}^{N} \sum_{j=1}^{i} \alpha \, dist1_{ij} + \beta \, dist2_{ij} \tag{4.3}$$

Since our dataset consists of two profiles generated using different approaches (more in 4.3.1), their distances are computed separately. Eq. (4.3) linearly combined them into one distance [104], where α and β control the contribution of first and second distance matrices respectively. Because $dist1_{ij}$ is the same as $dist1_{ji}$, only the lower left triangular matrix is considered.

4.3.3 Role Discovery

In the previous sections we defined our profiling techniques and based on them generated a lineally combined distance matrix. In this section the selection of the clustering approach to group users based on their role is discussed.

The binary nature of our dataset requires special proximity measures (more in Section 4.3.2). For instance, the K-means clustering algorithm is only based on a single distance measure (squared Euclidean distance), which is not suitable when dealing with binary data. On the other hand, hierarchical clustering allows the use of many distance functions, including the ones that are compatible with binary data. In our proposed solution, we used hierarchical agglomerative clustering (HAC) based on the Jaccard distance function as our proximity measure.

The fact that HAC allows different proximity measures to be used suggested the need for different linkage criteria to help in determining the merge decisions between clusters or a singleton object and a cluster. Some of the commonly used linkage methods are the single linkage, complete linkage, unweighted and weighted average linkage, and Ward's linkage [105]. We employed Ward's method as the linkage criterion in HAC because it was confirmed to work well with distances based on binary data [106, 107, 108].

4.3.4 Determining the Number of Existing Roles

The result of the HAC method is best presented in a dendrogram diagram, which is a tree representation used to show the hierarchical relationship between clusters and singleton clusters. As shown in Fig. 4.2, the x-axis of a dendrogram represents objects and clusters, while y-axis represents the distance between objects or clusters. The HAC method follows a bottom-up approach when clustering the data points. It starts from the leaves and works up until all the points are merged under one root cluster. Sub-clusters, representing roles, are crated by splitting the dendrogram at different points. Splitting the dendrogram at a low distance results in many roles and splitting it at a high level removes roles.

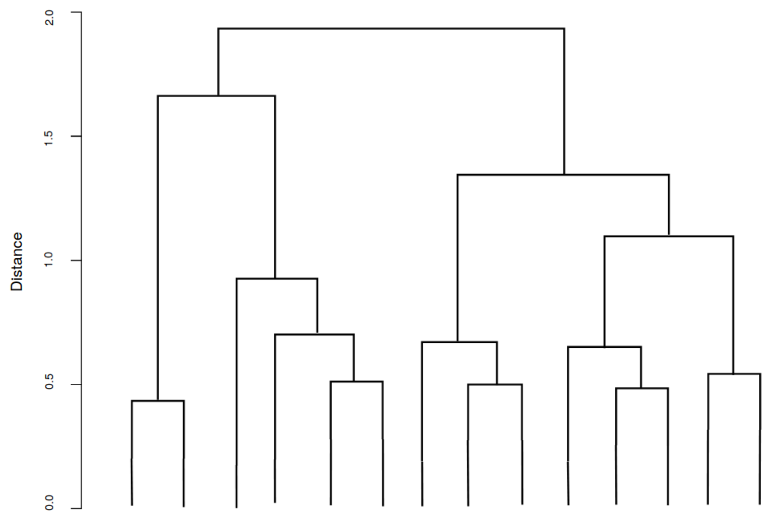


Figure 4.2: An example of a dendrogram

To identify the point at which to best cut the dendrogram, and thus identify the best possible set of distinguishable roles, we turned to the Ratkowsky-Lance clustering validation index (CVI) to find the optimal number of clusters present in the dataset [109]. The Ratkowsky-Lance index uses a compactness measure to estimate the optimal number of clusters. As shown in [110], Ratkowsky-Lance index was the best method for finding the right number of cluster when working with binary data compared to other indices.

$$BGSS_j = \sum_{k=1}^{q} n_k (c_{kj} - \overline{x}_j)^2$$

$$TSS_j = \sum_{i=1}^{n} (x_{ij} - \overline{x}_j)^2$$

$$\overline{S} = \frac{1}{p} \sum_{j=1}^{p} \sqrt{\frac{BGSS_j}{TSS_j}}$$

$$Ratkowsky = \frac{\overline{S}}{q^{1/2}} \tag{4.4}$$

where q is the number of clusters, and p is the number of features.

The Ratkowsky-Lance index is defined in formula (4.4) [111, 112], which equals the mean of $\sqrt{\frac{BGSS_j}{TSS_j}}$ divided by the square root of the number of clusters. The $BGSS_j$ is the sum of squares between the clusters for each variable j, while TSS_j stands for the total sum of squares for each variable j. The optimal number of clusters is q when $Ratkowsky$ is at its maximum value[110, 113].

$$\acute{x} = \frac{x_i - min(x)}{max(x) - min(x)} \tag{4.5}$$

where:

$i \in \{1, ..., N\}$ such that N is the number of pages per group

$$CI = \gamma\, I_1 + I_2 \tag{4.6}$$

where:

CI = combined clustering index

I_1 = clustering index based on behavior profile

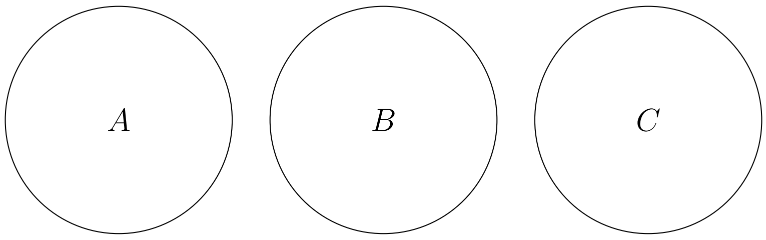
I_2 = clustering index based in access profile

γ = optimization variable, such that $\gamma > 1$

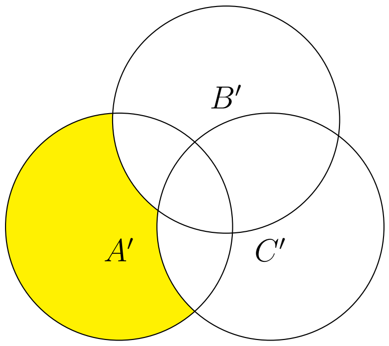
Using the highest clustering index as our selection criteria for determining the optimal number of roles (q). We computed the Ratkowsky-Lance index separately for the behavior and the access-based datasets, using different q values ranging from 2 to 8. Since two different datasets are used to compute the index score, we used Eq. (4.5) to normalize the scores and ensure both are within the same scale $[0, 1]$. We then linearly combined the normalized scores as defined in Eq. (4.6). We used the optimization variable γ to allow the behavior-based (I_1) index to contribute more to the overall index score (CI) because the access-based profiles were indirectly defined through users behavior, meaning such profiles tend to share more common features (e.g, website template links), which affect the compactness score of the cluster. This mitigates to some extent, misclassifying, for example, a registered user as a guest because the user did not visit significant pages (in access-based profiles alone), such as an account page.

4.3.5 Mining Authorization Business Rules

After discovering the optimal number of existing roles and clustering user traces, we generated authorization business rules based on traces that carry specific operations performed by the users (behavior) and operations they could perform (access). Extracting and combining behavior and access types of operations allowed us to create a set for each role consisting of the most common operations role holders perform. Infrequent operations are likely due to missed clustered traces, which can happen if the captured trace is short or observed traffic has not allowed roles to become sufficiently differentiated. Excluding such operations allow us to only focus on the general relationships between roles and infer more insights about the type of operations each role allows.



(a) Sets of operations for each role (A, B, and C)



(b) Operations only role A is authorized to perform

Figure 4.3: Extracting authorization business rules

While it may be common for roles to share similar operations, there will be other operations that are only allowed to be executed by a particular role. To detect such operations, we used the set difference operation (defined in Eq. (4.7)) to find the operational variation between discovered roles.

$$A' - B' = \{x \mid x \in A' \, and \, x \notin B'\} \tag{4.7}$$

where:

$A' \quad = \{x \mid x \in A \, and \, |x| \; in \; A \; >= \text{ freq-thr}\}$

$B' \quad = \{x \mid x \in B \, and \, |x| \; in \; B \; >= \text{ freq-thr}\}$

freq-thr $=$ frequency threshold

By way of illustration, suppose that there exists three sets of roles A, B, and C as shown in Figure a-4.3, where A={Op1 x20, Op2 x15, Op7 x33, Op13 x12}, B={Op1 x22, Op7 x16, Op4 x21}, C={Op17 x10, Op18 x8, Op20 x11, Op22 x7}, and a frequency threshold of 4. Determining the operations only role A is authorized to perform is computed by identifying the difference between the sets A, B, and C respectively. In this case, (A′ - B′) - C′ = {Op2, Op13}, which results in two authorization business rules: (i) ”only role A may do Op2”, and (ii) ”only role A may do Op13”.

4.4 Evaluation

In this section we evaluate our work using a dataset generated based on actual users' behavior. The evaluation is based on the following validation measures: (i) detecting the optimal number of clusters. (ii) quality of the detected clusters. (iii) the stability of the clustering approach.

4.4.1 The Dataset

The dataset used in this evaluation was captured and processed based on our proposed data pre-processing approach in Chapter 4. The generated workflow log was based on real users' interaction with an e-commerce shopping website named OsCommerce [76]. The website supports three different roles: admin, guest, and registered users. A local webserver was used to host and run the website. Users connected to the website through our data collection proxy used each role separately.

Role	**Total # of HTTP Messages**	**Total # of Cleaned HTTP Messages**	**Total # of Generated Traces (Sessions)**	**Avg # of Activities per Trace**
Admin	2594	1436	27	53.19
Registered	7455	3332	61	54.62
Guest	7501	2838	59	48.10
	17550	*7606*	*147*	**Sum**

Table 4.4: Dataset statistics

Table 4.4 shows the number of captured HTTP messages with and without media requests and shows the number of traces generated after processing the data. Approximately 20% of the traces pertain to the admin role, 40% to the guest role, and 40% to the registered user role. The generated workflow log was then used to build two new datasets (profiles) based users' behavior and access level (more in Section 4.3.1).

4.4.2 Clustering Result

Figure 4.4 shows the resulted dendrogram after clustering users traces based on a combined distance measure (behavior and access) with both optimization variables set to one. Three main clusters are evident. The first split occurs at a dissimilarity level of 29, while the second cluster (from the left) is further split into two clusters at level 17. We can conclude that the left-most cluster is much more dissimilar than the other two clusters at level 17.

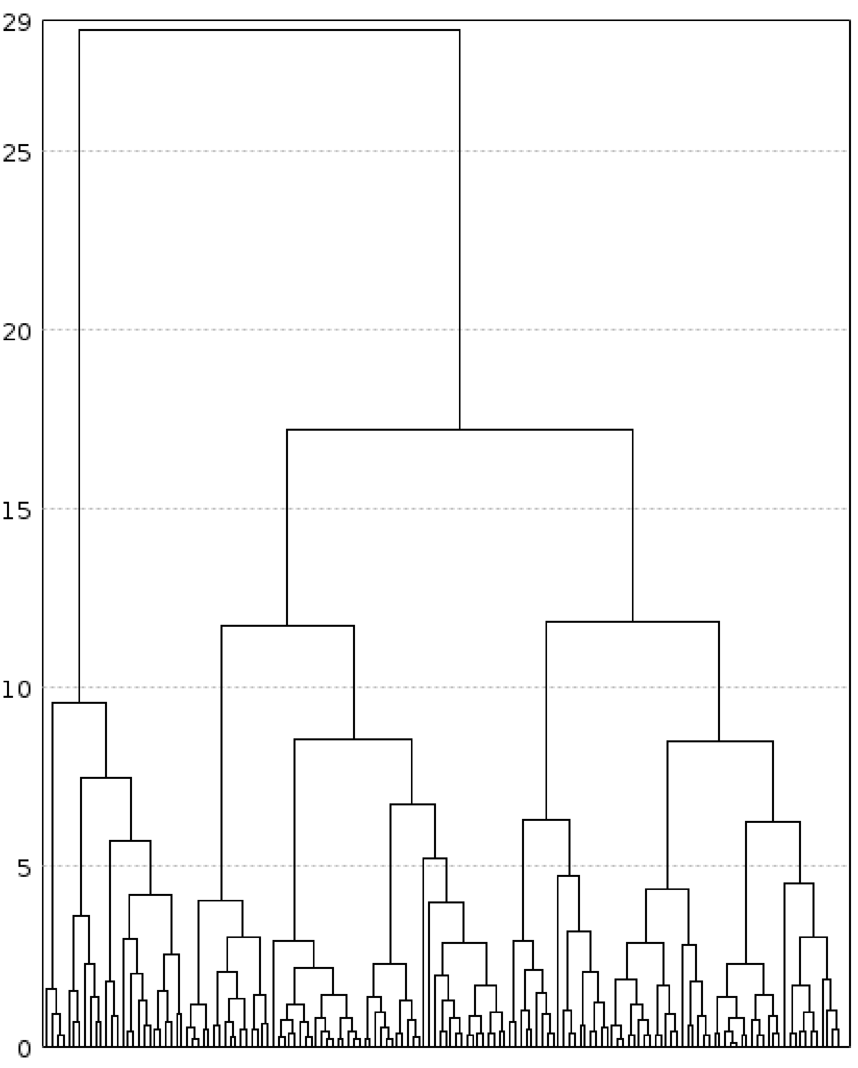


Figure 4.4: The resulted dendrogram after HAC cluster analysis

We detected the number of existing roles in the tested system. Table 4.5 shows a list of normalized Ratkowsky and Lance scores computed for different clustering configurations. The *Index Score 1* is based on the user behavior dataset whereas *Index Score 2* is based on user access dataset. The behavioral-based index indicates that the optimal number of roles occurs when $q = 3$. The access based profile, on the other hand, scored it's highest clustering index at $q = 2$. It is also noticeable that the index score for both profiles continues to decrease as the number of clusters increase.

Number of clusters (q)	**Index Score 1 ($Ratkowsky$)**	**Index Score 2 ($Ratkowsky$)**
2	0.9031	**1**
3	**1**	0.7271
4	0.7177	0.4580
5	0.4770	0.2691
6	0.2529	0.1076
7	0.1543	0.1232
8	0	0

Table 4.5: A normalized ratkowsky and lance scores based on two datasets

To detect the optimal number of existing roles, a weighed combined index score was computed with the optimization variable (γ) set to three. Figure 4.5 compares between different indices, where the y-axis represents the index score and the x-axis represents the number of clusters. Based on the unweighted combined index score ($\gamma = 1$) two clusters are the optimal number of roles. However, the weighted index score ($\gamma = 3$) shows that three is the optimal number of roles, which match the actual number of supported roles in OsCommerce.

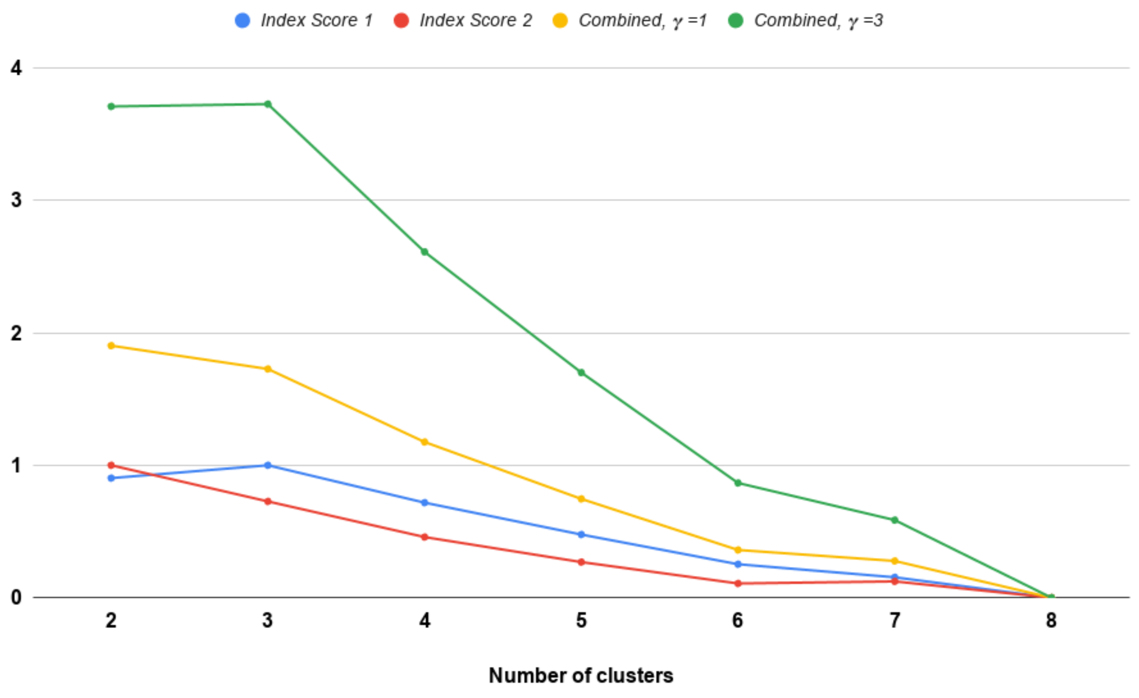


Figure 4.5: Comparing clustering indices scores

4.4.3 Clustering Quality

We employed an external validation measure to evaluate the quality of the detected clusters. External indices rely on the availability of a dataset with pre-specified labels, which is used to measure the extent to which the clustered data match with actual labels [114]. In our case, figuring out the right labels for the captured traces was possible, since roles were collected independently. We used the Maximum-Matching measure (an external Matching-Based index) to calculate the quality of the detected clusters because it finds the maximum sum of pairs in the matching matrix between clusters and classes where a class can be mapped to only one cluster at a time [115].

C$\backslash$T	**Admin**	**Registered**	**Guest**	**Sum**
Cluster 1	27	0	0	27
Cluster 2	0	1	59	60
Cluster 3	0	60	0	60
$\mathbf{m}_j$	27	61	59	147

Table 4.6: Matching matrix

Table 4.6 shows the generated matching matrix based on the detected clusters and the actual roles. The matrix shows that all instances in clusters 1 and 3 belong to the Admin and Registered role respectively, while nearly all instances in cluster 2 relate to the guest role. Based on the matching matrix the computed Maximum-Matching quality score was 0.993, as only one instance was miss clustered out of the 147 traces. This indicates that our clustering technique was able to cluster user traces and discover high-quality clusters.

4.4.4 Clustering Stability

We further tested the stability of our proposed solution by clustering and evaluating randomly shuffled sub-samples (d) based on the original dataset (D). We conducted four different clustering and evaluation iterations to compute a final average evaluation score. In each iteration the dataset was randomly shuffled and split into four sub-samples d_i, where $d_i \subset D$ and $i \in \{1, 2, 3, 4\}$. Each sub-sample (d_i) was then clustered and evaluated separately through Ratkowsky and Lance index. The average of all compactness scores per cluster configuration was computed and normalized to be compared to the same index score based on the entire dataset.

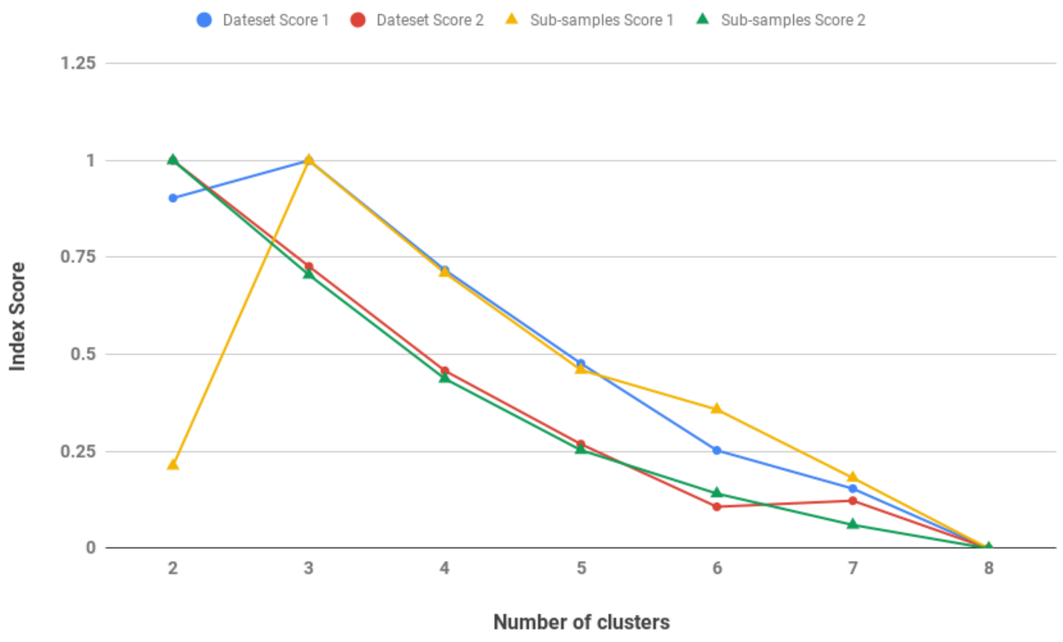


Figure 4.6: Dataset index scores compared to sub-samples average index scores

Figure 4.6 shows a comparison between two cluster compactness scores of different clustering configurations for the entire dataset and an average of four sub-samples. The figure shows how a four iteration of a randomly shuffled sub-samples scored a very similar compactness score when compared to an equivalent score computed based on the entire dataset. The results indicate that our solution was stable and managed to cluster sub-samples that, when evaluated, match the trend of the entire dataset.

Chapter 5

Discovering IF-Then Business Rules

The complexity of the decision-making process increases along with the rapid growth of information systems. Developing new features, editing existing ones, and removing others must always confirm with existing system business rules. More importantly, maintaining the business rules along with code updates is a key factor here, as they formalize how the system should behave. Neglecting or failing to maintain them in course of time will degrade the confidence in the business rules, which opens the door for potential Business Logic Vulnerabilities (BLV) that can be discovered and exploited by an attacker. With outdated business rules, the code itself becomes the only source of system documentation available [116, 117, 118].

In the domain of web applications, BLVs are hard to detect as they are unlike traditional injection attacks uniquely tailored to the targeted system. The difficulty in detecting them comes from the fact that each application has its own custom business logic that is hard to generalize. To detect such unique vulnerabilities one must understand the underlying application business logic in advance to be able to compare the observed behavior with the expected one.

Without a formal specification to describe the expected behavior, detecting BLVs becomes infeasible. The absence of such specifications can be addressed to some degree by recovering some of the business rules or system specifications using heuristic approaches. A static approach is a white-box method that analyzes the source code to discover business rules. Dynamic based approaches do not require the source code and, instead, rely on information generated during application execution. The quality of the discovered rules in the static approach depends on how well the business rules are implemented in the code, while the dynamic approach relies on the reachability level and the amount of dynamic data captured. Recovering business rules in the domain of web applications using the mentioned approaches continues to be a challenging problem due to lack of code and rapid changes in complexity, markets, and technologies.

In this chapter, we propose a novel mining approach for detecting dependency-based if-then business rules in web applications. The solution is based on a black-box approach that captures HTTP traces during normal execution. It requires no code or code modification that limits the overhead on the application performance. The remainder of the chapter is organized as follows: Section 5.1 explains if-the business rules in the context of web applications. Section 5.2 provide more details on the nature of these dependencies and their types. Section 5.3 defines and highlights some of the relations used in our solution. Section 5.4 presents our proposed solution. Lastly, Section 5.5 evaluates the proposed solution and assesses its performance.

5.1 IF-Then Business Rules and Web Applications

As discussed earlier in 2.3.1, the Business Rules Group categorizes business rules based on three types: structural assertion rules, action assertion rules, and derivation business rules[42]. As this research is mainly interested in discovering business rules based on the dynamic behavior of the system under test, our focus is on action assertion business rules. This type of business rules is used to control business behavior by defining constraints and conditions when implementing organization business rules. Action assertion business rules are further classified into authorization rules, integrity constraint rules, and condition business rules. Our focus in this chapter is on discovering condition business rules.

Condition business rules are assertion rules in the form of if-then statements [42] which, when resolved to true, suggest activation of some sort of business logic. In the web application domain, such business rules are very common and are enforced through business logic. Encoding condition business rules in the application business logic can be achieved using various forms of data such as direct user input, indirect user input (inferred values), and the state of a user session based on the usage behavior. A promotion code submitted by the user is an example of direct input, which is handled based on the guideline of the application business rules (e.g., one use only? excluded items? limit on the discount? ...etc). The decision whether to make an order eligible for free shipping is usually determined based on the total value of the order (i.e., indirect user's input).

Many decisions are made based on the current user state with respect to the business rules of the system. The usage behavior defines and indirectly alters the user state since the accessibility to many functionalities in web applications will depend on such state changes that are triggered by user behavior. For example, once a user's authenticity is confirmed, the state of the user changes along with how the system deals with the upcoming user's requests.

If-then business rules can be encoded in the system business logic using different techniques. In this research, we focus on analyzing the usage behavior to test the following hypothesis. Dependency mining can be applied on the usage behavior to rediscover if-then business rules that were used to define them.

5.2 Types of Dependencies

Two types of dependencies need to be taken into account when mining processes: explicit and implicit [119]. Most process mining algorithms rely on explicit dependency as a starting point to discover a process model (see [94], [97], and [96]). Explicit dependencies (also referred to as as local dependencies) are defined based on the number of times a task follows another over all the traces in the event log. Implicit dependencies, on the other hand, represent relations that indirectly exist between tasks, where the execution of one task can depend on the occurrences of one or more tasks in the same trace. The nature of such dependency makes them much more challenging to discover. In attempting to tackle this challenge, researchers in [119, 95, 120] proposed different heuristics approaches for detecting one-to-one implicit dependencies. However, implicit relations continue to be a challenging problem in process mining, especially when trying to detect one-to-many or even one-to-one relations in real-world applications event logs.

5.2.1 Positive and Negative Implicit Relations

Implicit dependency in one-to-one or one-to-many relations can be further classified into positive or negative relations [121]. A relation is considered positive if the dependency relies on the presence of one or more tasks, while relying on the absence of one or more tasks implies a negative implicit dependency relation. For example, editing a user account depends on the presence of a successful login before it can happen. The relation between both events/tasks is a positive implicit dependency. In contrast, the relation between editing an account page and logout page is negative since a logged out user can not edit her account. In this case, the dependency holds in the absence of another task.

In the context of web applications, implicit dependencies can be defined based on either navigation or business logic relations. Navigation pages in web applications consist of links that define all possible transitions from the current page. The order in which these pages are activated reflects the business logic enforcing the system business rules. Considering both kinds of relations when mining implicit dependencies help in discovering a more complete picture of what the developers are trying to enforce.

5.2.2 Dependencies Based on Substitute Relations

One of the challenges when mining for implicit dependencies is detecting those that are defined based on substitute relations. In such a case the implicit dependency is distributed across two or more tasks, where the occurrence of one task and one task only satisfies the dependency. For example, If D substitutes E, and A implicitly depends on E, then A implicitly depends on D too. Such dependencies are detected using frequency-based metrics (see [119, 95]), resulting in identifying only a segment of the implicit dependency or missing the dependency altogether.

In this Chapter, we present our proposed solution for mining positive implicit dependencies relations (we will refer to them as positive long distance dependency relations) with the presence of substitute relations. Our focus will be on discovering one-to-one and one-to-many implicit dependencies in web applications, with substitute relations of length two (pairs).

5.3 Preliminaries

This research expands on and improves the capability of Flexible Heuristic Miner (FHM) in detecting long-distance dependencies [95]. FHM is a mining algorithm that constructs a dependency graph as the means by which to discover the most likely process. We reuse (items 1 and 2 below) and extend some (items 3 and 4 below) of the relation definitions employed by FHM for constructing different causal dependency tables [95].

Let W be a workflow log over T, where T^* is all of the traces in the log composed of tasks T, such that $W \subseteq T^*$ and $\delta \in T^*$:

1. $a >_W b$ iff there is a trace $\delta = t_1t_2t_3...t_n$ and i $\in \{1, ..., n-1\}$ such that $\delta \in W$ and $t_i = a \, and \, t_{i+1} = b$ (direct successor)

2. $a >>>_W b$ iff there is a trace $\delta = t_1t_2t_3...t_n$ and $i < j$ and $i, j \in \{1, ..., n\}$ such that $\delta \in W$ and $t_i = a$ and $t_j = b$ (forward direct or indirect successor).

3. $a <<<_W b$ is defined as $b >>>_W a$ (backward direct or indirect successor).

4. $a <<<_{W'} b$ iff there is a trace $\delta = t_1t_2t_3...t_n$ and $i > j$ and $i, j \in \{1, ..., n\}$ such that $\delta \in W$ and $t_i = a$ and t_i is the first occurrence of a and $t_j = b$ (backward unique direct or indirect successor).

CASE ID	Events
Case 1	t_2,t_1,t_3,t_4,t_2,t_1
Case 2	t_3,t_2,t_3,t_1,t_4,t_1
Case 3	t_1,t_1,t_2,t_3,t_4,t_2
Case 4	t_3,t_4,t_1,t_2,t_3,t_1

Table 5.1: A workflow log example

$a >_W b$	$\boldsymbol{t_1}$	$\boldsymbol{t_2}$	$\boldsymbol{t_3}$	$\boldsymbol{t_4}$
$\boldsymbol{t_1}$	1	2	1	1
$\boldsymbol{t_2}$	2	0	3	0
$\boldsymbol{t_3}$	2	1	0	3
$\boldsymbol{t_4}$	2	2	0	0

Table 5.2: Direct relation

$a >>>_W b$	$\boldsymbol{t_1}$	$\boldsymbol{t_2}$	$\boldsymbol{t_3}$	$\boldsymbol{t_4}$
$\boldsymbol{t_1}$	4	6	4	4
$\boldsymbol{t_2}$	6	2	4	3
$\boldsymbol{t_3}$	8	4	2	5
$\boldsymbol{t_4}$	4	3	1	0

Table 5.3: Forward long distance relation

$a <<<_W b$	$\boldsymbol{t_1}$	$\boldsymbol{t_2}$	$\boldsymbol{t_3}$	$\boldsymbol{t_4}$
$\boldsymbol{t_1}$	4	6	8	4
$\boldsymbol{t_2}$	6	2	4	3
$\boldsymbol{t_3}$	4	4	2	1
$\boldsymbol{t_4}$	4	3	5	0

Table 5.4: Backward long distance relation

$a <<<_{W'} b$	$\boldsymbol{t_1}$	$\boldsymbol{t_2}$	$\boldsymbol{t_3}$	$\boldsymbol{t_4}$
$\boldsymbol{t_1}$	0	2	3	1
$\boldsymbol{t_2}$	3	0	2	1
$\boldsymbol{t_3}$	3	2	0	0
$\boldsymbol{t_4}$	4	3	5	0

Table 5.5: Backward unique long distance relation

The workflow log in Table 5.1 illustrates the four relations. It consists of four cases over $T = \{t_1, t_2, t_3, t_4\}$, based on different relations among tasks. The direct relation ($>_w$) shown in Table 5.2 counts the number of times a task directly followed another task. Table 5.3 shows the forward long distance relation, indicating a count of the number of times a task is directly or indirectly followed by another task. The backward long distance relation ($<<<_W$), shown in Table 5.4 presents to converse by counting the number of time a task directly or indirectly happened before another task. Lastly, Table 5.5 gives the backward unique long distance relation ($<<<_{W'}$), which is defined as the $<<<_W$ with the additional condition of only considering the first occurrence of a task. For example, $Case2$ ($\{t_3$,t_2,t_3,$\boldsymbol{t_1}$,t_4,$\boldsymbol{t_1}\}$) task t_1 happened twice but only the first occurrence is considered unique.

5.4 Methodology

In this section, we discuss our proposed solution for detecting implicit long distance dependencies. As mentioned earlier, implicit dependencies can be positive or negative. Our focus is on the former. The rest of the section is organized as follows: Section 5.4.1 illustrates the different relations used to build a dependency graph. Section 5.4.2 describes identification of substitute relations. Section 5.4.3 explains discovery of long distance dependency.

5.4.1 Step1: Build the dependency graph

The first step in FHM is building the dependency graph (DG). Graph nodes represent tasks that can be connected with other tasks through edges based on their causal dependency. Because the number of relations between tasks can be large, a frequency-based metric is used to measure the strength of the relationship to control which edges are considered. FHM defines several different dependency measures. We use some [95] and introduce another as follows:

$$a \Rightarrow_W b = \left(\frac{|a >_W b| - |b >_W a|}{|a >_W b| + |b >_W a| + 1} \right), \ \ if\ a \neq b \tag{5.1}$$

$$a \Rightarrow^l_W b = \left(\frac{2\ (|a >>>_W b|)}{|a| + |b| + 1} \right) - \left(\frac{2\ Abs(|a| - |b|)}{|a| + |b| + 1} \right) \tag{5.2}$$

The direct dependency measure defined in Eq. (5.1) quantifies the direct dependency relation between two tasks a and b such that $|a >_W b|$ is the number of times a was directly followed by b in $W \subseteq T^*$. The term, $\Rightarrow^l_W$, in Eq. (5.2) calculates the directly- or indirectly-followed-by relation between tasks (usually referred to as long distance dependency) such that $|a >>>_W b|$ is the number of times a was directly or indirectly followed by b in $W \subseteq T^*$, and $|a|$ is the count of time a was executed in $W \subseteq T^*$. Duplicate patterns are considered when constructing the forward long distance relation ($>>>_W$) table, as tasks can be performed more than once in a single trace. Such patterns will also contribute to the final value of $|a >>>_W b|$ and $|a|$.

We assume that a long distance pattern, when repeated, will continue to match the original detected pattern limits the discovery of long distance relations as in Eq. (5.2). In real-world applications such as websites, usually, the dependency must be completely satisfied once per session. For example, a user only has to log in to the system once per session to have access to other functionality available to logged-in users; fulfilling that dependency means that the user can execute these functionalities without needing to log in every time. To address this, we introduce a new long distance dependency measure capable of discovering a wide range of long distance relations.

$$a \Rightarrow^{l}_{W'} b = \left(\frac{2\ (|a <\!<\!<_{W'} b|)}{|a'| + |b'| + 1} \right) \tag{5.3}$$

The new measure ($\Rightarrow^{l}_{W'}$) defined in Eq. (5.3) is a direct and indirect backward long distance dependency measure defined based on the first occurrence of the target task, where only the tasks that happened before it is considered. $|a <\!<\!<_{W'} b|$ is the number of times b directly or indirectly preceded a based on only the first occurrence of a, and $|a'|$ is the number of traces in which a occurred at least once. The main idea here is to avoid duplicate long distance relations per trace, allowing us to focus on only the initial dependency of each task.

5.4.2 Step2: Detect substitute relations

A long distance dependency is not always a straightforward one-to-one relation, as a task can depend on the execution of one task from multiple tasks. For example, some web applications allow users to access the account page after either creating a new account or successfully logging to the system, but not both since logged-in users have no need to create new accounts. We term this type of relationship a long distance with substitute dependency relationship.

FHM uses frequency-based metrics when computing dependencies and, in the case of long distance dependency defined over a substitute relation, the frequency is divided among two or more substitute tasks. Substitute relations that involve dominant tasks with high frequency are likely to be considered when mining for long distance relations, whereas less frequent tasks are likely to be dropped. Also, tasks that are observed at roughly the same frequency are not likely to be considered because the dependency is equally distributed among them. To solve this problem, we detect the substitute relations at an early stage and resolve them so they can be easily recognized at a subsequent stage when mining for long distance relations.

Detecting substitute relations between pairs of tasks is accomplished in three steps. First, we detect candidate tasks that can be part of a substitute relation and preceded the target task. Then, we pair these candidates based on their negative impact on the presence of one another. Finally, we filter out the detected pairs to include real substitute relations only. For illustration purposes, our focus is on detecting pairs of tasks with substitute relations, however, the same approach can be applied to detect longer substitute relations, such as a triple.

Algorithm 3: Detect Candidate Events for Substitute Relations

```
Result: A list of candidate substitute events for $e$
Input: Event of interest ($e$); Unique event counter of $e$ (UEC_e)
FindCandidatesOF ($e, UEC_e$)
    C = $\emptyset$ ; // Candidate events.
    foreach $c \in E$ do
        if $c \neq e$ then
            LRSCFO$_{ce}$ = $getLongRangeSuccessionCountFirstOcc(c, e)$
            if (LRSCFO$_{ce}$ $> 0$) $\wedge$ (LRSCFO$_{ce}$ $<$ UEC$_e$) then
                C.$add(c)$
    return C
```

In Algorithm 3, we search for tasks that precede task e and have the potential to participate in a substitute relation. Lines 3-8 iterate over all events in E looking for candidate tasks, such that $E \subseteq T$. For a task to be considered a candidate (lines 5-7), the number of times it is directly or indirectly executed before e based on the first occurrence ($|e <<<_{W'} c|$) must be less than $|e'|$. Otherwise, it can not be considered, because substitute relation can not exist if the candidate task matches or exceeds $|e'|$. The algorithm culminates following the loop by returning a (possibly empty) list of candidate tasks.

Algorithm 4: Detect Candidate Pairs for Substitute Relations

Result: A list of potential substitute events e may depend on

Input: Event of interest (e); Unique event counter of e (`UEC_e`) ; Candidate substitute events of e (C)

```
FindSubstituteCandidatesOF ($e, UEC_e, C$)
    S = ∅ ; // Substitute candidate events.
    foreach $c_1 \in C$ do
        foreach $c_2 \in C$ do
            if $c_2 \neq c_1$ then
                UEC$_{c_1}$ = $getUniqueEventCount(c_1)$
                UEC$_{c_2}$ = $getUniqueEventCount(c_2)$
                LRSCFO$_{c_1e}$ = $getLongRangeSuccessionCountFirstOcc(c_1, e)$
                LRSCFO$_{c_2e}$ = $getLongRangeSuccessionCountFirstOcc(c_2, e)$
                LRSCFO$_{c_1c_2}$ = $getLongRangeSuccessionCountFirstOcc(c_1, c_2)$
                LRSCFO$_{c_2c_1}$ = $getLongRangeSuccessionCountFirstOcc(c_2, c_1)$
                // Compute support score
                Supp$_{c_1}$ = $\frac{\text{UEC}_{c_1}}{|W|}$; Supp$_{c_2}$ = $\frac{\text{UEC}_{c_2}}{|W|}$
                Supp$_{c_1c_2}$ = $\frac{\text{LRSCFO}_{c_1c_2}}{|W|}$; Supp$_{c_2c_1}$ = $\frac{\text{LRSCFO}_{c_2c_1}}{|W|}$
                // Compute lift score
                Lift$_{c_1c_2}$ = $\frac{\text{Supp}_{c_1c_2}}{\text{Supp}_{c_1} \times \text{Supp}_{c_2}}$; Lift$_{c_2c_1}$ = $\frac{\text{Supp}_{c_2c_1}}{\text{Supp}_{c_2} \times \text{Supp}_{c_1}}$
                // Compute dependency score
                LRD$_{c_1e}$ = $\frac{2 \times \text{LRSCFO}_{c_1e}}{\text{UEC}_{c_1} + \text{UEC}_e + 1}$; LRD$_{c_2e}$ = $\frac{2 \times \text{LRSCFO}_{c_2e}}{\text{UEC}_{c_1} + \text{UEC}_e + 1}$
                if $(LRD_{c_1e} \geq LRD\_THR) \wedge (LRD_{c_2e} \geq LRD\_THR)$ then
                    SUB_RATIO = $\frac{\text{LRSCFO}_{c_1e} + \text{LRSCFO}_{c_2e}}{\text{UEC}_e}$
                    if $(SUB\_RATIO \geq MINEM) \wedge (SUB\_RATIO \leq MAXEM)$ then
                        if $(Lift_{c_1c_2} \leq LIFT\_THR) \wedge (Lift_{c_2c_1} \leq LIFT\_THR)$ then
                            S.$addIfNotExists(c_1)$
                            S.$addIfNotExists(c_2)$
    return S
```

After detecting all candidate tasks (C) for $e \in T$, Algorithm 4 creates pairs of tasks with potential substitute relations. In lines 3-24, all possible pairs are tested and only those that meet the selection criteria are considered. Two tasks (a and b) are considered to have a potential substitute relation with respect to e only if the following conditions are met:

- As defined in Eq. (5.4), both a and b must have a long distance dependency score with e greater or equal to the long distance dependency threshold (default = 0.1). This condition is checked in lines 18-19.

- The number of times a and b happened before e based on their first occurrence must be equal or very close to the number of times e' was executed. As shown in Eq. (5.5), the substitute ratio is computed and evaluated with an error margin of ± 0.1. Lines 20-21 compute and test the substitute ratiod.

- The lift score of $a \Rightarrow^{l}_{Lift'} b$ and $b \Rightarrow^{l}_{Lift'} a$ should be less than 1 and as close as possible to zero. A lift score ¡ 1 indicates a negative dependency or a substitute effect, thus the closer the score is to zero the stronger the effect. In Eq. (5.6), the lift score ($\Rightarrow^{l}_{Lift'}$) is computed based on the first occurrence, then compared to the lift threshold (default = 0.15). Lines 6-16 compare lift scores and the lift scores are tested in line 22 to determine if they are sufficiently close to the threshold for the candidate pair to be considered further.

$$e \Rightarrow^{l}_{W'} a \geq \textbf{LRD\_THR} \;\wedge\; e \Rightarrow^{l}_{W'} b \geq \textbf{LRD\_THR},\ \textit{such that } a \neq b \tag{5.4}$$

$$\textbf{Substitute\_Ratio}^{e}_{ab} = \frac{|e <<<_{W'} a| \,+\, |e <<<_{W'} b|}{|e'|}$$

$$\textbf{MAXEM} \geq \textbf{Substitute\_Ratio}^{e}_{ab} \geq \textbf{MINEM},\ \textit{such that } a \neq b \tag{5.5}$$

$$a \Rightarrow^{l}_{Lift'} b = \left(\frac{support_{a<<<_{W'}b}}{(support_{a'})(support_{b'})}\right) = \left(\frac{\left(\frac{|a<<<_{W'}b|}{|W|}\right)}{\left(\frac{|a'|}{|W|}\right)\left(\frac{|b'|}{|W|}\right)}\right)$$

$$\left(a \Rightarrow^{l}_{Lift'} b \;\leq \textbf{LIFT\_THR}\right) \;\wedge\; \left(b \Rightarrow^{l}_{Lift'} a \leq \textbf{LIFT\_THR}\right),\ \textit{such that } a \neq b \tag{5.6}$$

Algorithm 5: Detect Actual Substitute Relations

```
Result: A map of substitute events in pairs per event
Input: A list of unique events $(E \in T)$ exist in the traces
DetectSubstituteRelations $(E)$
    R = $\emptyset$ ; // Result.
    foreach $e \in E$ do
        P = $\emptyset$ ; // Events with a substitute relation for e
        UEC$_e$ = $getUniqueEventCount(e)$
        C = $FindCandidatesOF(e)$ ; // Candidate events.
        S = $FindSubstituteCandidatesOF(e, \texttt{UEC}_e, C)$
        foreach $s_1 \in S$ do
            Conflict = $False$; UEC$_{s_1}$ = $getUniqueEventCount(s_1)$
            foreach $s_2 \in S$ do
                if $s_1 \neq s_2$ then
                    UEC$_{s_2}$ = $getUniqueEventCount(s_2)$
                    // Compute direct dependency
                    DSC$_{s_1 s_2}$= $getDirectSuccessionCount(s_1, s_2)$
                    DSC$_{s_2 s_1}$= $getDirectSuccessionCount(s_2, s_1)$
                    DD$_{s_1 s_2}$ = $\frac{\mathbf{DSC}_{s_1 s_2} - \mathbf{DSC}_{s_2 s_1}}{\mathbf{DSC}_{s_1 s_2} + \mathbf{DSC}_{s_2 s_1}}$
                    // Compute long range dependency
                    LRSC$_{s_1 s_2}$ = $getLongRangeSuccessionCount(s_1, s_2)$
                    LRD$_{s_1 s_2}$ = $\frac{2 \times \mathbf{LRSC}_{s_1 s_2}}{\mathbf{UEC}_{s_1} + \mathbf{UEC}_{s_2} + 1}$
                    if $(DD_{s_1 s_2} \geq DD\_TRH) \vee (LRD_{s_1 s_2} \geq LRD\_TRH)$ then
                        Conflict = $True$; Break
            if Conflict = $False$ then
                P.$add(s_1)$
        R.$add((e, P))$
```

Algorithm 5 assesses all the candidate pairs with substitute relations and generates a final list per task of the actual substitute relations. Lines 5-7 use both Algorithms 3 and 4 to create a list of candidates with potential substitute relations for each task. Lines 8-24 evaluate each candidate with all other candidates in the same list to make sure their direct and long distance dependency scores were below the threshold (default direct=0.6, long= 0.6). The main goal here is to eliminate any candidate pairs that directly or indirectly depend on one another, as tasks associated with a substitute relation should not appear at the same time and trace.

5.4.3 Step3: Mine positive long distance dependencies

As shown in the previous section, detecting positive long distance dependencies with the presence of substitute relations represents a challenge. To overcome that, we propose a heuristic approach to help in detecting substitute relations at early stages in the mining process. In this section, we show how the detected substitute relations are handled and employed to improve the mining process of long distance dependencies.

Early detection and handling of involved substitute relations is a critical step to successfully detect long distance dependency relations. The strength of such relations that are defined based on substitutionally associated tasks will be lower, due to the effect of substitute relations among tasks. Using a measure such as $\Rightarrow^{l}_{W}$ will fail to detect some or all of dependencies due to a low score. To prevent distributing the frequency in the first place, we resolve the substitute relations by merging tasks with such relations to form one single task. Take, for example, $W = \{\mathrm{A\underline{B}F\mathbf{E}A}^5, \mathrm{G\underline{C}G\mathbf{E}H}^8, \mathrm{D\underline{B}HD\mathbf{E}}^7\}$, where **E** depend on the occurs of B or C, such that B and C have a substitute relation. To resolve the substitute relation in the previous example, both B and C are merged together to form a new task "BC", and W will become =$\{$A"BC"F**E**A^5, G"BC"G**E**H^8, D"BC"HD**E**$^7\}$.

Algorithm 6: Detect Positive Long Distance Dependency Relations

```
Result: A map of positive long distance dependencies per event
Input: A list of events ($(E')$) with resolved substitute relations
1  FindPositiveLongDistanceDependencies $(E')$
2      R = $\emptyset$
3      foreach $e \in E'$ do
4          P = $\emptyset$
5          UEC$_e$ = $getUniqueEventCount(e)$
6          foreach $c \in E'$ do
7              if $c \neq e$ then
8                  LRSCFO$_{ce}$ = $getLongRangeSuccessionCountFO(c, e)$
9                  if $LRSCFO_{ce} \geq UEC_e$ then
10                     P.$add(c)$
11         R.$add((e, P))$
12     return R
```

After resolving all the substitute relations in the event log (W), Algorithm 6 is used to mine the positive long distance dependency relations. The algorithm iterates overall tasks in $E' \in T$ and searches for such relations between tasks based on the first occurrence of the target task (e). As shown in line 9, the relation selection criterion is based on the existence of a backward long distance dependency relation between a and b, which is determined by the number of times b happened before the first occurrence of a and ($|a <<<_{W'} b|$) is greater or equal to $|a'|$. Because a task can have more than one long distance dependency relation with other tasks, a list of relations is created in lines 4 & 10. Finally, a map of all detected relations per task is returned in line 12.

5.5 Evaluation

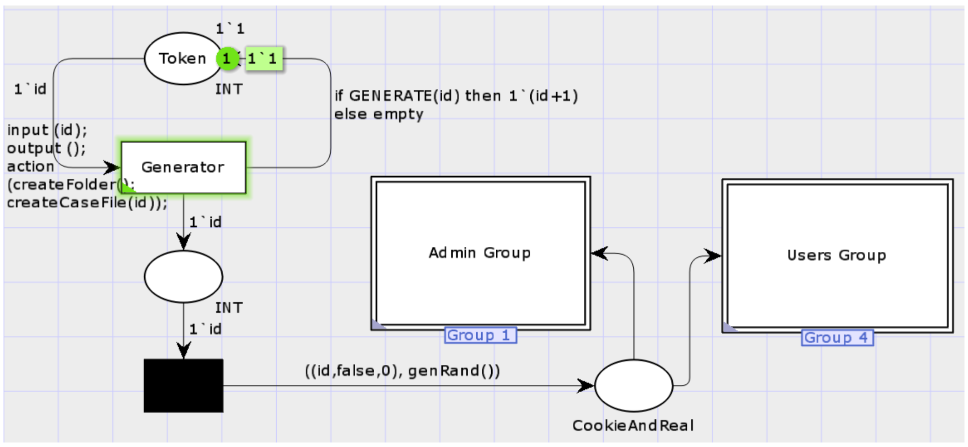
To evaluate our proposal, we created a Petri net model based on the specification of a real-world e-commerce web application (osCommerce). We used a Petri net simulator to automatically generate synthetic workflow logs satisfying modeled specifications and simulating real users' behavior. With the specifications of the tested model in hand, we evaluated the capability of our proposal in re-discovering if-then business rules based on only dynamic behavior generated rom workflow logs. We also compared our solution with other dependency-based mining algorithms such Flexible Heuristics Miner(FHM).

5.5.1 Dataset

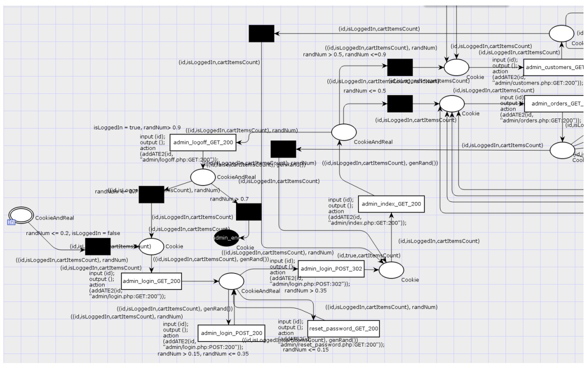
The synthetic event logs used in this evaluation were automatically generated using CPN Tools [122], which is a software for modeling, simulating, and analyzing Colored Petri nets (CP-nets). We also used LogRec scripts to export the simulated event logs in the XES format[123]. The generated event logs are the result of executing the provided process model through the simulator. As shown in Figure 5.1, the Petri net model we used was designed based on the specification of a popular open-source e-commerce shopping website named OsCommerce (version 2.3.4.1). The following aspects were prioritized when developing the model:

- Including all the three supported roles by OsCommerce (admin, guest, and registered users).
- Enforcing business rules through guard inscription associated with transitions.
- Allowing a more realistic behavior through weighted random selection.
- Maintaining users' state during the simulation.

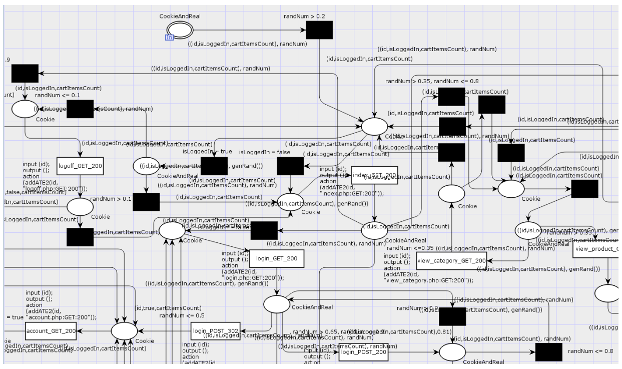
The total number of positive long distance dependency relations that existed in the Petri net model was 34, 13 for the admin role, 3 for the guest, and 18 for the registered user role. A total of 153 dependees in all roles. A dependee is what defines a long distance dependency, such that a relation can depend on one or more (in our model at least two). For example, the login page task is a very common dependee, since many long distance relations depend on it in both the admin and registered user roles.



(a) A simplified high-level view of our petri net model



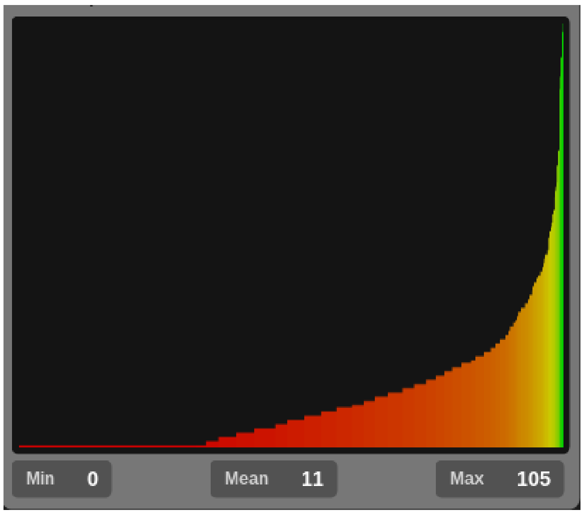
(b) Part of the admin petri net



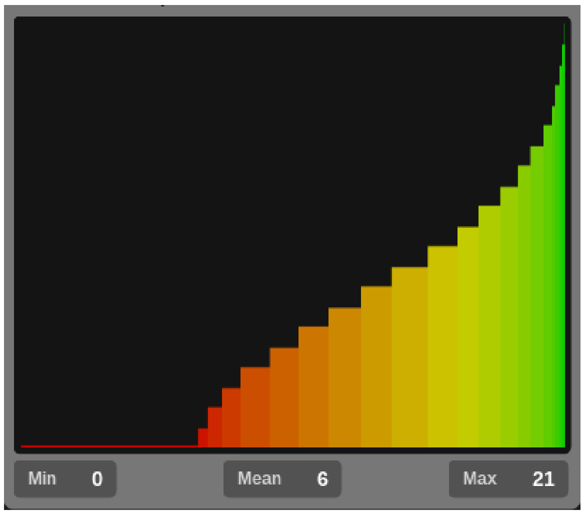
(c) Part of the guest & registered users petri net

Figure 5.1: A petri net model based on OsCommerce

Simulating the execution of the OsCommerce-based process model yielded an XES event log of 910 cases. Around 10,500 events from 56 classes were distributed among the cases. As shown in Figure 5.2-a, the events count per case ranged from 0 to 105 with a mean value of 11. Figure 5.2-b shows the average number of event classes per case, which was around 6. The generated event log reflected realistic behaviors for all the supported roles by assigning weights to the simulator transition decisions: 10% of customers placed an order, admin traffic represented 15% of the overall, customers did not usually enter long loops (e.g., add/remove from cart, login/logout), and customers followed links within the website as intended.



(a) Events per case



(b) Event classes per case

Figure 5.2: Data distribution of the synthetic event logs

5.5.2 Substitute Relations

Because substitute relations represent a challenge when mining for long distance relations, detecting them at early stages in the mining process was crucial. We defined the Petri net model to generate traffic illustrative of a substitute relation between two tasks and evaluated the ability of our proposed approach to detected such substitute relations.

The substitute relation defined in the model was based on the transition from one role to another. OsCommerce offers two ways in which a user can log in to the website, either by signing in via the login page or signing up by creating a new account. Only one scenario can take place at a time, even if one is most likely to occur more than the other. In both cases, the user's role change form guest to registered, which means both tasks deliver the same effect, therefore a substitute relation exists between them.

Role	Page	Observed Substitute Relations
Registered	account.php:GET:200	login.php:POST:302 ⇔ create_account_success.php:GET:200
	edit_account.php:GET:200	
	edit_account.php:POST:200	
	edit_account.php:POST:302	
	logoff.php:GET:200	
	my_address_book.php:GET:200	
	my_address_book.php:POST:200	
	my_address_book.php:POST:302	
	orders_history.php:GET:200	
	payment_methods.php:GET:200	
	payment_methods.php:POST:200	
	payment_methods.php:POST:302	

Table 5.6: The discovered substitute relations

Table 5.6 shows that one substitute relation was discovered belonging to the registered user role and defined between two tasks the "login.php:POST:302" and the "create_account_success .php:GET:200". A total of 12 unique tasks observed the same relation at different times of the execution. If we compare them to the total number of tasks' classes (18) under the registered user role, we found out that this relation was supported by more than half of the classes in the role. This shows that our approach was able to discover all the substitute relations in the event log by detecting all the tasks that observed these relations.

5.5.3 Positive Long Distance Dependency Relations

The possible behavior in OsCommerce and, hence in our Perti net model, was based on three roles: admin, guest, registered users. Each role enforced some restrictions to guarantee that only allowed behavior was performed. Restrictions were due to transition/navigation purposes (e.g., login page can be reached the through index page) or actual business rules of the application (e.g., only registered users can place orders). Both kind of restrictions were considered in the evaluation as they defined how a system could be used. Worth mentioning, our focus was only on long distance relations. Direct relations based on such restrictions were not considered.

Table 5.7 shows the discovered positive long distance dependency relations per role. All detected tasks with a long distance relation are highlighted under the page column. TP (True Positive) column indicates the number of correctly detected dependees (tasks depend on), while FP (False Positive) counts the number of falsely detected dependees. A total of 13 long distance relations were discovered in the admin role with an average of number of dependees of 4. For the registered user role 18 relations were detected and a dependee average of 5. A total of 3 relations were detect for the guest role with a mean of truly detected dependee around 3. The result shows that our proposed solution was able to discover all positive long distance relations along with all the dependees tasks these relations depend on.

One of the discovered positive long distance dependency relations is shown in Figure 5.3. The relation is based on the "checkout_success.php:GET:200" task, with a total of 10 dependees. This task represents the last step in a checkout process as defined in OsCommerce. The relation successfully reconstructed the if-then business rule associated with the checkout process. As can be seen in the figure, a user must have at least one item in the cart and logged in the website (two substitute tasks detected) to be able to start the checkout. To successfully check out the user must select a payment ("checkout_payment.php:GET:200") and a shipping method ("checkout_shipping.php:GET:200") then confirm the order ("checkout_confirmation.php:GET:200") to be placed.

Role	Long Distance Relations				
	Page	**TP**	**FP**	**Actual # Dependees**	**ATPR**
Admin	admin/customers.php:GET:200	3	0	3	100%
	admin/delete_customer.php:GET:200	4	0	4	
	admin/delete_customer.php:POST:302	5	0	5	
	admin/delete_order.php:GET:200	4	0	4	
	admin/delete_order.php:POST_302	5	0	5	
	admin/edit_customer.php:GET:200	4	0	4	
	admin/edit_customer.php:POST:302	5	0	5	
	admin/edit_order.php:GET:200	4	0	4	
	admin/edit_order.php:POST:302	5	0	5	
	admin/invoice.php:GET:200	4	0	4	
	admin/logoff.php:GET:200	3	0	3	
	admin/orders.php:GET:200	3	0	3	
	admin/packingslip.php:GET:200	4	0	4	
Registered	checkout.php:GET:200	6	0	6	100%
	checkout_confirmation.php	9	0	9	
	checkout_payment.php:GET:200	8	0	8	
	checkout_promo_code.php:GET:200	10	0	10	
	checkout_shipping.php:GET:200	7	0	7	
	checkout_success.php:GET:200	10	0	10	
	create_account_success.php:GET:200	3	0	3	
	edit_account.php:GET:200	3	0	3	
	edit_account.php:POST:200	4	0	4	
	edit_account.php:POST:302	4	0	4	
	logoff.php:GET:200	3	0	3	
	my_address_book.php:GET:200	3	0	3	
	my_address_book.php:POST:200	4	0	4	
	my_address_book.php:POST:302	4	0	4	
	orders_history.php:GET:200	3	0	3	
	payment_methods.php:GET:200	3	0	3	
	payment_methods.php:POST:200	4	0	4	
	payment_methods.php:POST:302	4	0	4	
Guest	cart.php:GET:200	2	0	2	100%
	checkout.php:GET:302	3	0	3	
	remove_item.php:POST:302	3	0	3	

Table 5.7: The detected positive long distance dependency relations

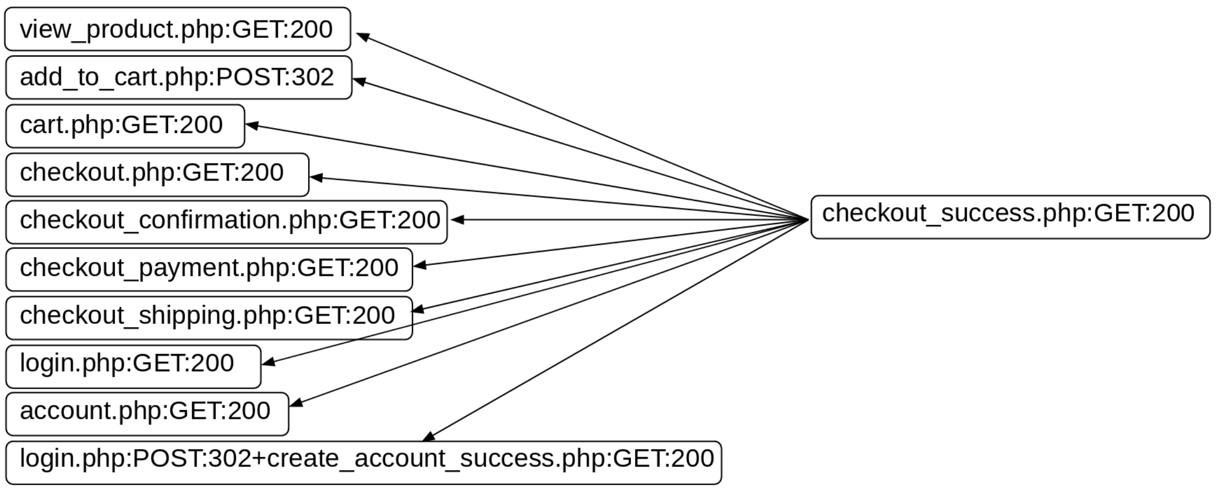

view_product.php:GET:200
add_to_cart.php:POST:302
cart.php:GET:200
checkout.php:GET:200
checkout_confirmation.php:GET:200
checkout_payment.php:GET:200
checkout_shipping.php:GET:200
login.php:GET:200
account.php:GET:200
login.php:POST:302+create_account_success.php:GET:200
checkout_success.php:GET:200

Figure 5.3: A long distance relation with all the tasks it depends on

Lastly, we compared our approach with the Flexible Heuristics Miner (FHM) in detecting long distance dependency relations using the same synthetic event logs. As can be seen in Table 5.8, FHM only detected a single long distance relation with the long distance threshold set to as low as 10%, while using our approach all long distance relations were detected along with all their dependees. This indicates that our novel approach was able to discover positive long distance dependency relations and largely improved when compared to the FHM.

	Total # of Detected Long Distance Relations	**Total # of Detected Dependees**	**Total # of Actual Long Distance Relations**	**Total # of Actual Dependees**
Our Approach	34	153	34	153
FHM	1*	2		

Table 5.8: Comparing our approach vs the flexible heuristic miner

Chapter 6

Conclusions and Future Work

The advancement in systems and techniques for detecting and preventing web injection attacks shifted the attack surface of web applications toward more advanced application-specific attacks by exploiting faulty business logic. The lack of formal specifications defining expected system behavior presents a significant challenge when detecting or preventing attacks that exploit business logic vulnerabilities (BLV). In this dissertation, we present a novel black-box approach for mining several types of business rules toward detecting and preventing BLV in web applications.

6.1 Summary of Contributions

We present a novel framework for converting HTTP traffic into high fidelity event logs that conform to the IEEE 1849-2016 XES standard, which complies with the data requirements of process mining. This allows users and developers to build and utilize many process mining algorithms in the area of process discovery, conformance checking, and performance mining. The new framework presents an easy-to-integrate solution that requires no modification to the source code and with a minimal overhead on the application performance, allowing it to passively monitoring the application's dynamic artifact (HTTP traffic). To the best of our knowledge, this is the first solution to precisely encode web applications dynamic behavior in event logs, which was achieved through several advanced mining techniques. These techniques include web content mining, identifying unique tasks/activities using structural dissimilarity, and largely expanding the state-space of each task through a multidimensional naming scheme allowing for higher fidelity in the generated models.

Using the new framework, we propose a novel automated approach for discovering authorization business rules, which are widely used for defining what users can and can not do. Many black-box solutions rely on user behavior to discover such rules but poorly perform when users from different roles share access to common functionalities. The proposed solution employs different profiling techniques considering not only what users did, but also what they can do. Our approach uses agglomerative hierarchical clustering on a combination of behavioral and reachability profiling techniques to discover organizational roles. Furthermore, we automate the process of detecting the optimal number of existing roles of the test system. Our results show the quality and stability of the proposed solution, which continues to perform well even with smaller datasets.

We illustrate our novel approach for detecting If-Then Business Rules defined based on implicit dependencies (or what we call positive long distance dependency relations). This type of business rules is very common in web applications, which controls when certain rules are applied based on one or more conditions. Our initial analysis showed that current frequency-based metrics are very limited when detecting implicit dependencies and fail to discover long distance relations that depend on tasks with substitute relations. Our proposed approach presentes a solution capable of discovering if-then business rules defined over one-to-one and one-to-many implicit dependency relations. The solution also mitigates the effect of substitute relations by detecting and resolving them at early stages in the discovery process. The results of our evaluation indicate a high precision in recovering positive long distance dependency relations from the observed behavior even with the presence of substitute relations. The detection rate of our new approach provides a 99% improvement when compared to other mining algorithms for detecting implicit dependencies.

6.2 Future Work

Capturing Ajax calls in web applications

More and more web applications are adapting and employing Ajax web technology to create asynchronous applications. Ajax allows exchanging data with a web server without interfering with the displayed page through background calls triggered automatically or through user action. However, capturing the behavior of such web applications in the standard event log format can be a challenge. For example, the transition between tasks/pages in traditional web applications is straightforward, as it is directly triggered based on the previous page. Ajax-enabled web applications can have N calls before a transition takes place from one page to another. Complicating things further, these calls can result in altering the HTML elements of a page, which can add, remove, or modify page behavior. This creates a new level of behavior at the page level itself. The solution should be able to capture the main transitions, and at the same time be able to represent pages internal behavior.

Detecting negative long distance relations

As this research focuses on discovering positive long distance relations, another equally important type of implicit dependencies is negative long distance relations. In contrast to positive relations, negative long distance relations are defined based on the absence of one or more tasks. For example, there exists a negative implicit dependency relation between pages that require authorization and the $logout$ page, since users are not supposed to edit their accounts after they logged out. The discovered If-Then business rules can be expanded using the mined negative relations and can be further used to detect access control violations.

Automate the process of detecting BLV

The lack of formal specifications defining the expected system behavior is one of the primary challenges when it comes to automating the detection of Business Logic Vulnerabilities (BLV). Without them, it is difficult to make mature decisions to determine if the observed behavior confirms with the expected behavior. However, one of the main objectives of this work was toward bridging the gap between BLV and automated detection approaches, which was achieved by mining and discovering different types of system business rules. The next step is to build an automated BLV detection system, which stress test the application guided with the discovered system specifications to detect any diverge in the perceived behavior

References

[1] Duchene, F., Rawat, S., Richier, J. L., & Groz, R. (2013, October). LigRE: Reverse-engineering of control and data flow models for black-box XSS detection. In 2013 20th Working Conference on Reverse Engineering (WCRE) (pp. 252-261). IEEE.

[2] Sindre, G., & Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. Requirements engineering, 10(1), 34-44.

[3] Haley, C., Laney, R., Moffett, J., & Nuseibeh, B. (2008). Security requirements engineering: A framework for representation and analysis. IEEE Transactions on Software Engineering, 34(1), 133-153.

[4] Shostack, A. (2014). Threat modeling: Designing for security. John Wiley & Sons.

[5] Shostack, A. (2008, September). Experiences threat modeling at microsoft. In Modeling Security Workshop. Dept. of Computing, Lancaster University, UK.

[6] Burns, S. F. (2005). Threat modeling: A process to ensure application security. GIAC Security Essentials Certification (GSEC) Practical Assignment.

[7] Verdon, D., & McGraw, G. (2004). Risk analysis in software design. IEEE Security & Privacy, 2(4), 79-84.

[8] Allen, J. H. (2008). Software security engineering: a guide for project managers. Upper Saddle River, NJ: Addison-Wesley.

[9] Huang, Y. W., Tsai, C. H., Lin, T. P., Huang, S. K., Lee, D. T., & Kuo, S. Y. (2005). A testing framework for Web application security assessment. Computer Networks, 48(5), 739-761.

[10] Huang, Y. W., Yu, F., Hang, C., Tsai, C. H., Lee, D. T., & Kuo, S. Y. (2004, May). Securing web application code by static analysis and runtime protection. In Proceedings of the 13th international conference on World Wide Web (pp. 40-52). ACM.

[11] Ciampa, A., Visaggio, C. A., & Di Penta, M. (2010, May). A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications. In Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems (pp. 43-49). ACM.

[12] Duchene, F., Rawat, S., Richier, J. L., & Groz, R. (2014, March). KameleonFuzz: evolutionary fuzzing for black-box XSS detection. In Proceedings of the 4th ACM conference on Data and application security and privacy (pp. 37-48). ACM.

[13] Duchene, F., Rawat, S., Richier, J. L., & Groz, R. (2013, October). LigRE: Reverse-engineering of control and data flow models for black-box XSS detection. In Reverse Engineering (WCRE), 2013 20th Working Conference on (pp. 252-261). IEEE.

[14] Rathore, S., Sharma, P. K., & Park, J. H. (2017). XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs. Journal of Information Processing Systems, 13(4).

[15] Wassermann, G., & Su, Z. (2008, May). Static detection of cross-site scripting vulnerabilities. In Proceedings of the 30th international conference on Software engineering (pp. 171-180). ACM.

[16] Li, X., & Xue, Y. (2011, December). BLOCK: a black-box approach for detection of state violation attacks towards web applications. In Proceedings of the 27th Annual Computer Security Applications Conference (pp. 247-256). ACM.

[17] Felmetsger, V., Cavedon, L., Kruegel, C., & Vigna, G. (2010, August). Toward automated detection of logic vulnerabilities in web applications. In USENIX Security Symposium (Vol. 58).

[18] Doupé, A., Cavedon, L., Kruegel, C., & Vigna, G. (2012, August). Enemy of the state: A state-aware black-box web vulnerability scanner. In USENIX Security Symposium (Vol. 14).

[19] Cova, M., Balzarotti, D., Felmetsger, V., & Vigna, G. (2007, September). Swaddler: An approach for the anomaly-based detection of state violations in web applications. In International Workshop on Recent Advances in Intrusion Detection (pp. 63-86). Springer, Berlin, Heidelberg.

[20] Pellegrino, G., & Balzarotti, D. (2014, February). Toward Black-Box Detection of Logic Flaws in Web Applications. In NDSS.

[21] Deepa, G., Thilagam, P. S., Praseed, A., & Pais, A. R. (2018). DetLogic: A black-box approach for detecting logic vulnerabilities in web applications. Journal of Network and Computer Applications.

[22] Deepa, G., & Thilagam, P. S. (2016). Securing web applications from injection and logic vulnerabilities: Approaches and challenges. Information and Software Technology, 74, 160-180.

[23] Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2010). CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. ACM Transactions on Information and System Security (TISSEC), 13(2), 14.

[24] Thomas, S., & Williams, L. (2007, May). Using automated fix generation to secure SQL statements. In Proceedings of the Third International Workshop on Software Engineering for Secure Systems (p. 9). IEEE Computer Society.

[25] Scholte, T., Robertson, W., Balzarotti, D.,& Kirda, E. (2012, July). Preventing input validation vulnerabilities in web applications through automated type analysis. In Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual (pp. 233-243). IEEE.

[26] Skrupsky, N., Bisht, P., Hinrichs, T., Venkatakrishnan, V. N., & Zuck, L. (2013, February). TamperProof: a server-agnostic defense for parameter tampering attacks on web applications. In Proceedings of the third ACM conference on Data and application security and privacy (pp. 129-140). ACM.

[27] Krishnamurthy, A., Mettler, A., & Wagner, D. (2010, April). Fine-grained privilege separation for web applications. In Proceedings of the 19th international conference on World wide web (pp. 551-560). ACM.

[28] Dalton, M., Kozyrakis, C., & Zeldovich, N. (2009). Nemesis: Preventing Authentication & [and] Access Control Vulnerabilities in Web Applications.

[29] Son, S., McKinley, K. S., & Shmatikov, V. (2013, February). Fix Me Up: Repairing Access-Control Bugs in Web Applications. In NDSS.

[30] Lee, I., Jeong, S., Yeo, S., & Moon, J. (2012). A novel method for SQL injection attack dete‘ction based on removing SQL query attribute values. Mathematical and Computer Modelling, 55(1-2), 58-68.

[31] Halfond, W. G., & Orso, A. (2005, November). AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (pp. 174-183). ACM.

[32] Jang, Y. S., & Choi, J. Y. (2014). Detecting SQL injection attacks using query result size. Computers & Security, 44, 104-118.

[33] Grabowski, R., Hofmann, M., & Li, K. (2011, September). Type-based enforcement of secure programming guidelines—code injection prevention at SAP. In International Workshop on Formal Aspects in Security and Trust (pp. 182-197). Springer, Berlin, Heidelberg.

[34] Juillerat, N. (2007, October). Enforcing code security in database web applications using libraries and object models. In Proceedings of the 2007 Symposium on Library-Centric Software Design (pp. 31-41). ACM.

[35] Johns, M., Beyerlein, C., Giesecke, R., & Posegga, J. (2010, February). Secure code generation for web applications. In International Symposium on Engineering Secure Software and Systems (pp. 96-113). Springer, Berlin, Heidelberg.

[36] Deepa, G., Thilagam, P. S., Khan, F. A., Praseed, A., Pais, A. R., & Palsetia, N. (2018). Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications. International Journal of Information Security, 17(1), 105-120.

[37] Appelt, D., Nguyen, C. D., Briand, L. C., & Alshahwan, N. (2014, July). Automated testing for SQL injection vulnerabilities: an input mutation approach. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (pp. 259-269). ACM.

[38] Byrne, D. (2012). Anatomy of a Logic Flaw. OWASP. `https://www.owasp.org/images/e/eb/AppSecEU2012_Anatomy_of_a_Logic_Flaw.pdf`

[39] Watson, C., Melton, D. G. J., Michael, J. A. Z. R. B., Reynolds, C. C. M. J., & Coates, M. (2008). AppSensor Guide.

[40] Suto, L. (2010). Analyzing the accuracy and time costs of web application security scanners. San Francisco, February.

[41] Martínez, S., Cosentino, V., & Cabot, J. (2017). Model-based analysis of Java EE web security misconfigurations. Computer Languages, Systems & Structures, 49, 36-61.

[42] Hay, D., Healy, K. A., & Hall, J. (2000). Defining business rules-what are they really. The Business Rules Group, 400.

[43] Crerie, R., Baião, F. A., & Santoro, F. M. (2009). Discovering business rules through process mining. In Enterprise, Business-Process and Information Systems Modeling (pp. 136-148). Springer, Berlin, Heidelberg.

[44] Neuer M. (2008, May). Context is King: A Practical Approach to Rule Mining. Business Rules Journal Vol. 9, No. 5, URL: `http://www.brcommunity.com/a2008/b415.html`

[45] Chaparro, O., Aponte, J., Ortega, F., & Marcus, A. (2012, October). Towards the automatic extraction of structural business rules from legacy databases. In Reverse Engineering (WCRE), 2012 19th Working Conference on (pp. 479-488). IEEE.

[46] Normantas, K., & Vasilecas, O. (2013). A Systematic Review of Methods for Business Knowledge Extraction from Existing Software Systems. Baltic Journal of Modern Computing (BJMC), 1(1-2), 29-51.

[47] Wang, R., Chen, S., Wang, X., & Qadeer, S. (2011, May). How to Shop for Free Online–Security Analysis of Cashier-as-a-Service Based Web Stores. In Security and Privacy (SP), 2011 IEEE Symposium on (pp. 465-480). IEEE.

[48] Wen, S., Xue, Y., Xu, J., Yuan, L. Y., Song, W. L., Yang, H. J., & Si, G. N. (2017). Lom: Discovering logic flaws within MongoDB-based web applications. International Journal of Automation and Computing, 14(1), 106-118.

[49] Bisht, P., Hinrichs, T., Skrupsky, N., & Venkatakrishnan, V. N. (2011, October). WAPTEC: whitebox analysis of web applications for parameter tampering exploit construction. In Proceedings of the 18th ACM conference on Computer and communications security (pp. 575-586). ACM.

[50] Bisht, P., Hinrichs, T., Skrupsky, N., Bobrowicz, R., & Venkatakrishnan, V. N. (2010, October). NoTamper: automatic blackbox detection of parameter tampering opportunities in web applications. In Proceedings of the 17th ACM conference on Computer and communications security (pp. 607-618). ACM.

[51] Alkhalaf, M., Choudhary, S. R., Fazzini, M., Bultan, T., Orso, A., & Kruegel, C. (2012, July). Viewpoints: differential string analysis for discovering client-and server-side input validation inconsistencies. In Proceedings of the 2012 International Symposium on Software Testing and Analysis (pp. 56-66). ACM.

[52] Gombotz, R., & Dustdar, S. (2005, September). On web services workflow mining. In International Conference on Business Process Management (pp. 216-228). Springer, Berlin, Heidelberg.

[53] Cooley, R., Mobasher, B., & Srivastava, J. (1999). Data preparation for mining world wide web browsing patterns. Knowledge and information systems, 1(1), 5-32.

[54] Eckersley, P. (2010, July). How unique is your web browser?. In International Symposium on Privacy Enhancing Technologies Symposium (pp. 1-18). Springer, Berlin, Heidelber

[55] Mowery, K., & Shacham, H. (2012). Pixel perfect: Fingerprinting canvas in HTML5. Proceedings of W2SP, 1-12.

[56] Günther, C. W., & Verbeek, E. (2014). XES Standard Definition v2. 0.

[57] Van der Aalst, W. M. (2016). Process mining: data science in action. Springer.

[58] Poggi, N., Muthusamy, V., Carrera, D., & Khalaf, R. (2013). Business process mining from e-commerce web logs. In Business process management (pp. 65-80). Springer, Berlin, Heidelberg.

[59] De Leoni, M., & van der Aalst, W. M. (2013, March). Data-aware process mining: discovering decisions in processes using alignments. In Proceedings of the 28th annual ACM symposium on applied computing (pp. 1454-1461). ACM.

[60] Earls, A. B., S. M. Embury, and N. H. Turner. "A method for the manual extraction of business rules from legacy source code." BT Technology Journal 20.4 (2002): 127-145.

[61] Chiang, C. C. (2006, April). Extracting business rules from legacy systems into reusable components. In System of Systems Engineering, 2006 IEEE/SMC International Conference on (pp. 6-pp). IEEE.

[62] Shekar, S., Hammer, J., Schmalz, M., & Topsakal, O. (2003). Knowledge extraction in the seek project part ii: Extracting meaning from legacy application code through pattern matching. tech. rep., University of Florida.

[63] ProM [Computer software]. (2019). Retrieved from `http://www.processmining.org/prom/start`

[64] Disco [Computer software]. (2019). Retrieved from `https://fluxicon.com/disco/`

[65] XESame [Computer software]. (2019). Retrieved from `http://www.processmining.org/xesame/start`

[66] OpenXES [Computer software]. (2019). Retrieved from `http://www.xes-standard.org/openxes/start`

[67] w3af [Computer software]. (2019). Retrieved from `https://github.com/andresriancho/w3af/`

[68] VEGA Vulnerability Scanner [Computer software]. (2019). Retrieved from `https://subgraph.com/vega/`.

[69] Nikto web server scanner [Computer software]. (2019). Retrieved from `https://cirt.net/Nikto2`

[70] OWASP Zed Attack Proxy (ZAP) [Computer software]. (2019). Retrieved from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[71] Burp Suite Web Scanner [Computer software]. (2019). Retrieved from http://customizedsynergy.com

[72] Netsparker Web Application Security Scanner [Computer software]. (2019). Retrieved from https://www.netsparker.com/

[73] IBM Security AppScan [Computer software]. (2019). Retrieved from https://www.ibm.com/us-en/marketplace/appscan-standard

[74] Acunetix WVS [Computer software]. (2019). Retrieved from https://www.acunetix.com/

[75] SQLMap [Computer software]. (2019). Retrieved from http://sqlmap.org

[76] osCommerce [Computer software]. (2019). Retrieved from https://www.oscommerce.com/

[77] Burp Extender [Computer software]. (2019). Retrieved from https://portswigger.net/burp/documentation/desktop/tools/extender

[78] Apache HTTP Server [Computer software]. (2019). Retrieved from https://httpd.apache.org/

[79] Apache Tomcat [Computer software]. (2019). Retrieved from http://tomcat.apache.org/

[80] IIS Web Server [Computer software]. (2019). Retrieved from https://www.iis.net/

[81] JBoss Web Server [Computer software]. (2019). Retrieved from https://www.redhat.com/en/technologies/jboss-middleware/web-server

[82] Righetto, D. (2014). Log Requests to SQLite [Computer software]. Retrieved from https://github.com/portswigger/log-requests-to-sqlite

[83] Zonk. (2005). Cross-Site Scripting Worm Floods MySpace. Retrieved from https://it.slashdot.org/story/05/10/14/126233/cross-site-scripting-worm-floods-myspace

[84] Rational Asset Analyzer 6.1.0 documentation. (2019). Business rule mining process . Retrieved from https://www.ibm.com/support/knowledgecenter/en/SS3JHP_6.1.0/com.ibm.raa.analyze.doc/common/brm_process.html

[85] XES Working Group. (2016). IEEE standard for eXtensible Event Stream (XES) for achieving interoperability in event logs and event streams. IEEE Std 1849, 1-50.

[86] Object Management Group (OMG). Semantics of Business Vocabulary and Business Rules (SBVR), Version 1.4, May 2017.

[87] Berendt, B., Mobasher, B., Nakagawa, M., & Spiliopoulou, M. (2002, July). The impact of site structure and user environment on session reconstruction in web usage analysis. In International workshop on mining web data for discovering usage patterns and profiles (pp. 159-179). Springer, Berlin, Heidelberg.

[88] Eickhoff, C., Teevan, J., White, R., & Dumais, S. (2014, February). Lessons from the journey: a query log analysis of within-session learning. In Proceedings of the 7th ACM international conference on Web search and data mining (pp. 223-232).

[89] Ortega, J. L., & Aguillo, I. (2010). Differences between web sessions according to the origin of their visits. Journal of Informetrics, 4(3), 331-337.

[90] Gowda, T., & Mattmann, C. A. (2016, July). Clustering web pages based on structure and style similarity (application paper). In 2016 IEEE 17th International conference on information reuse and integration (IRI) (pp. 175-180). IEEE.

[91] Pawlik, M., & Augsten, N. (2015). Efficient computation of the tree edit distance. ACM Transactions on Database Systems (TODS), 40(1), 1-40.

[92] Pawlik, M., & Augsten, N. (2016). Tree edit distance: Robust and memory-efficient. Information Systems, 56, 157-173.

[93] Buijs, J. C., Van Dongen, B. F., & van Der Aalst, W. M. (2012, September). On the role of fitness, precision, generalization and simplicity in process discovery. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (pp. 305-322). Springer, Berlin, Heidelberg.

[94] Van der Aalst, W., Weijters, T., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 16(9), 1128-1142.

[95] Weijters, A. J. M. M., & Ribeiro, J. T. S. (2011, April). Flexible heuristics miner (FHM). In 2011 IEEE symposium on computational intelligence and data mining (CIDM) (pp. 310-317). IEEE.

[96] Günther, C. W., & Van Der Aalst, W. M. (2007, September). Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In International conference on business process management (pp. 328-343). Springer, Berlin, Heidelberg.

[97] Leemans, S. J., Fahland, D., & van der Aalst, W. M. (2013, June). Discovering block-structured process models from event logs-a constructive approach. In International conference on applications and theory of Petri nets and concurrency (pp. 311-329). Springer, Berlin, Heidelberg.

[98] van Eck, M. L., Buijs, J. C., & van Dongen, B. F. (2014, September). Genetic process mining: Alignment-based process model mutation. In International Conference on Business Process Management (pp. 291-303). Springer, Cham.

[99] Leemans, S. J. (2017). Robust process mining with guarantees.

[100] Suriadi, S., Andrews, R., ter Hofstede, A. H., & Wynn, M. T. (2017). Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Information Systems, 64, 132-150.

[101] Verhulst, R. (2016). Evaluating quality of event data within event logs: an extensible framework (Doctoral dissertation, Master's thesis, Eindhoven University of Technology).

[102] Gower, J. C., & Legendre, P. (1986). Metric and Euclidean properties of dissimilarity coefficients. Journal of classification, 3(1), 5-48.

[103] Choi, S. S., Cha, S. H., & Tappert, C. C. (2010). A survey of binary similarity and distance measures. Journal of Systemics, Cybernetics and Informatics, 8(1), 43-48.

[104] Ruan, M. M. (2010). Timing and Frequency Synchronization in Practical OFDM Systems.

[105] Murtagh, F. (1983). A survey of recent advances in hierarchical clustering algorithms. The computer journal, 26(4), 354-359.

[106] Hands, S., & Everitt, B. (1987). A Monte Carlo study of the recovery of cluster structure in binary data by hierarchical clustering techniques. Multivariate Behavioral Research, 22(2), 235-243.

[107] Tamasauskas, D., Sakalauskas, V., & Kriksciuniene, D. (2012, December). Evaluation framework of hierarchical clustering methods for binary data. In 2012 12th International Conference on Hybrid Intelligent Systems (HIS) (pp. 421-426). IEEE.

[108] Meunier, B., Dumas, E., Piec, I., Bechet, D., Hebraud, M., & Hocquette, J. F. (2007). Assessment of hierarchical clustering methodologies for proteomic data mining. Journal of proteome research, 6(1), 358-366.

[109] Ratkowsky, D. A., & Lance, G. N. (1978). Criterion for determining the number of groups in a classification.

[110] Dimitriadou, E., Dolničar, S., & Weingessel, A. (2002). An examination of indexes for determining the number of clusters in binary data sets. Psychometrika, 67(1), 137-159.

[111] Charrad, M., Ghazzali, N., Boiteau, V., Niknafs, A., & Charrad, M. M. (2014). Package 'nbclust'. Journal of statistical software, 61, 1-36.

[112] Desgraupes, B. (2013). Clustering indices. University of Paris Ouest-Lab Modal'X, 1, 34.

[113] Milligan, G. W., & Cooper, M. C. (1985). An examination of procedures for determining the number of clusters in a data set. Psychometrika, 50(2), 159-179.

[114] Halkidi, M., Batistakis, Y., & Vazirgiannis, M. (2001). On clustering validation techniques. Journal of intelligent information systems, 17(2-3), 107-145.

[115] Wagner, S., & Wagner, D. (2007). Comparing clusterings: an overview (pp. 1-19). Karlsruhe: Universität Karlsruhe, Fakultät für Informatik.

[116] Huang, H., Tsai, W. T., Bhattacharya, S., Chen, X. P., Wang, Y., & Sun, J. (1996, August). Business rule extraction from legacy code. In Proceedings of 20th International Computer Software and Applications Conference: COMPSAC'96 (pp. 162-167). IEEE.

[117] de Souza, S. C. B., Anquetil, N., & de Oliveira, K. M. (2006). Which documentation for software maintenance?. Journal of the Brazilian Computer Society, 12(3), 31-44.

[118] Kajko-Mattsson, M. (2001, November). The state of documentation practice within corrective maintenance. In Proceedings IEEE International Conference on Software Maintenance. ICSM 2001 (pp. 354-363). IEEE.

[119] Wen, L., Wang, J., & Sun, J. (2006, January). Detecting implicit dependencies between tasks from event logs. In Asia-Pacific Web Conference (pp. 591-603). Springer, Berlin, Heidelberg.

[120] Chabrol, M., Dalmas, B., Norre, S., & Rodier, S. (2016, June). A process tree-based algorithm for the detection of implicit dependencies. In 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS) (pp. 1-11). IEEE.

[121] Wan, Q., & An, A. (2003, June). Efficient mining of indirect associations using hi-mine. In Conference of the Canadian Society for Computational Studies of Intelligence (pp. 206-221). Springer, Berlin, Heidelberg.

[122] De Medeiros, A. A., & Günther, C. W. (2005). Process mining: Using CPN tools to create test logs for mining algorithms. In Proceedings of the sixth workshop on the practical use of coloured Petri nets and CPN tools (CPN 2005) (Vol. 576).

[123] SIMBAD. (2014, March 18). LogRec - Exploiting Process Logs for Activity Recommendation. Retrieved April 2, 2020, from http://www-inf.it-sudparis.eu/SIMBAD/tools/LogRec/

Appendices

Appendix A

A sample of the generated XES event log

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file has been generated with the OpenXES library. It conforms -->
<!-- to the XML serialization of the XES standard for log storage and -->
<!-- management. -->
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<!-- OpenXES is available from http://www.openxes.org/ -->
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="
   1.0RC7">
      <extension name="Organizational" prefix="org" uri="http://www.xes-
         standard.org/org.xesext"/>
      <extension name="Time" prefix="time" uri="http://www.xes-standard.org
         /time.xesext"/>
      <extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-
         standard.org/lifecycle.xesext"/>
      <extension name="Concept" prefix="concept" uri="http://www.xes-
         standard.org/concept.xesext"/>
      <global scope="trace">
            <string key="concept:name" value="name"/>
      </global>
      <global scope="event">
            <string key="concept:name" value="name"/>
            <string key="org:resource" value="resource"/>
            <string key="time:timestamp" value="timestamp"/>
            <string key="lifecycle:transition" value="transition"/>
      </global>
      <classifier name="Event Name" keys="concept:name"/>
      <string key="lifecycle:model" value="standard"/>
      <string key="concept:name" value="Test transaction"/>
      <trace>
            <string key="concept:name" value="127.0.0.1--Mozilla/5.0 (X11;
               Linux x86_64; rv:64.0) Gecko/20100101 Firefox/64.0;-0"/>
            <event>
                  <string key="concept:name" value="||start||"/>
            </event>
            <event>
                  <string key="set-cookie" value="True"/>
                  <string key="status-code" value="200"/>
                  <string key="method" value="GET"/>
                  <string key="lifecycle:transition" value="complete"/>
                  <date key="time:timestamp" value="2019-04-07T12
                     :55:14.742-05:00"/>
                  <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
                     X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
                     /64.0;"/>
                  <string key="concept:name" value="_oscom_index.
                     php_GET_200"/>
            </event>
            <event>
                  <string key="query_products_id" value="24"/>
                  <string key="referer" value="/oscom/"/>
```

```
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :55:23.249-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_product_info.
            php_GET_200"/>
</event>
<event>
        <string key="query_products_id" value="24"/>
        <string key="referer" value="/oscom/product_info.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :55:25.276-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_product_reviews.
            php_GET_200"/>
</event>
<event>
        <string key="query_products_id" value="24"/>
        <string key="referer" value="/oscom/product_reviews.php"/
            >
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="302"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :55:26.922-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="
            _oscom_product_reviews_write.php_GET_302"/>
</event>
<event>
        <string key="referer" value="/oscom/product_reviews.php"/
            >
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :55:27.009-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_login.
            php_GET_200"/>
</event>
<event>
        <string key="referer" value="/oscom/login.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :55:34.451-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_login.
            php_GET_200"/>
```

```
</event>
<event>
      <string key="referer" value="/oscom/login.php"/>
      <string key="set-cookie" value="False"/>
      <string key="status-code" value="200"/>
      <string key="method" value="GET"/>
      <string key="lifecycle:transition" value="complete"/>
      <date key="time:timestamp" value="2019-04-07T12
         :55:37.969-05:00"/>
      <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
         X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
         /64.0;"/>
      <string key="concept:name" value="_oscom_index.
         php_GET_200"/>
</event>
<event>
      <string key="query_y" value="0"/>
      <string key="query_x" value="0"/>
      <string key="query_search_in_description" value="1"/>
      <string key="query_keywords" value="car"/>
      <string key="referer" value="/oscom/index.php"/>
      <string key="set-cookie" value="False"/>
      <string key="status-code" value="200"/>
      <string key="method" value="GET"/>
      <string key="lifecycle:transition" value="complete"/>
      <date key="time:timestamp" value="2019-04-07T12
         :55:43.069-05:00"/>
      <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
         X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
         /64.0;"/>
      <string key="concept:name" value="
         _oscom_advanced_search_result.php_GET_200"/>
</event>
<event>
      <string key="query_products_id" value="1"/>
      <string key="referer" value="/oscom/
         advanced_search_result.php"/>
      <string key="set-cookie" value="False"/>
      <string key="status-code" value="200"/>
      <string key="method" value="GET"/>
      <string key="lifecycle:transition" value="complete"/>
      <date key="time:timestamp" value="2019-04-07T12
         :55:46.171-05:00"/>
      <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
         X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
         /64.0;"/>
      <string key="concept:name" value="_oscom_product_info.
         php_GET_200"/>
</event>
<event>
      <string key="body_id[4]" value="2"/>
      <string key="body_id[3]" value="6"/>
      <string key="query_action" value="add_product"/>
      <string key="body_products_id" value="1"/>
      <string key="query_products_id" value="1"/>
      <string key="referer" value="/oscom/product_info.php"/>
      <string key="set-cookie" value="False"/>
      <string key="status-code" value="302"/>
      <string key="method" value="POST"/>
      <string key="lifecycle:transition" value="complete"/>
      <date key="time:timestamp" value="2019-04-07T12
         :55:57.282-05:00"/>
      <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
         X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
         /64.0;"/>
      <string key="concept:name" value="_oscom_product_info.
         php_POST_302"/>
</event>
<event>
      <string key="referer" value="/oscom/product_info.php"/>
      <string key="set-cookie" value="False"/>
```

```
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :55:57.373-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_shopping_cart.
            php_GET_200"/>
</event>
<event>
        <string key="query_products_id" value="1{4}2{3}6"/>
        <string key="referer" value="/oscom/shopping_cart.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :56:00.251-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_product_info.
            php_GET_200"/>
</event>
<event>
        <string key="body_id[4]" value="1"/>
        <string key="body_id[3]" value="5"/>
        <string key="query_action" value="add_product"/>
        <string key="body_products_id" value="1"/>
        <string key="query_products_id" value="1{4}2{3}6"/>
        <string key="referer" value="/oscom/product_info.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="302"/>
        <string key="method" value="POST"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :56:05.163-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_product_info.
            php_POST_302"/>
</event>
<event>
        <string key="referer" value="/oscom/product_info.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :56:05.235-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_shopping_cart.
            php_GET_200"/>
</event>
<event>
        <string key="referer" value="/oscom/shopping_cart.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
            :56:15.541-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
            X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
            /64.0;"/>
        <string key="concept:name" value="_oscom_index.
           php_GET_200"/>
    </event>
    <event>
        <string key="query_manufacturers_id" value="7"/>
        <string key="referer" value="/oscom/index.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
           :56:22.255-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
           X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
           /64.0;"/>
        <string key="concept:name" value="_oscom_index.
           php_GET_200"/>
    </event>
    <event>
        <string key="query_cPath" value="3"/>
        <string key="referer" value="/oscom/index.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
           :56:28.948-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
           X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
           /64.0;"/>
        <string key="concept:name" value="_oscom_index.
           php_GET_200"/>
    </event>
    <event>
        <string key="query_cPath" value="3_15"/>
        <string key="referer" value="/oscom/index.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
           :56:32.907-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
           X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
           /64.0;"/>
        <string key="concept:name" value="_oscom_index.
           php_GET_200"/>
    </event>
    <event>
        <string key="query_cPath" value="3_15"/>
        <string key="query_products_id" value="16"/>
        <string key="referer" value="/oscom/index.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2019-04-07T12
           :56:40.998-05:00"/>
        <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
           X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
           /64.0;"/>
        <string key="concept:name" value="_oscom_product_info.
           php_GET_200"/>
    </event>
    <event>
        <string key="query_cPath" value="3_15"/>
        <string key="query_products_id" value="16"/>
        <string key="referer" value="/oscom/product_info.php"/>
        <string key="set-cookie" value="False"/>
        <string key="status-code" value="200"/>
        <string key="method" value="GET"/>
```

```
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
                :56:45.204-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
                X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
                /64.0;"/>
            <string key="concept:name" value="_oscom_product_reviews.
                php_GET_200"/>
      </event>
      <event>
            <string key="referer" value="/oscom/product_reviews.php"/
                >
            <string key="set-cookie" value="False"/>
            <string key="status-code" value="200"/>
            <string key="method" value="GET"/>
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
                :56:47.339-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
                X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
                /64.0;"/>
            <string key="concept:name" value="_oscom_index.
                php_GET_200"/>
      </event>
      <event>
            <string key="referer" value="/oscom/index.php"/>
            <string key="set-cookie" value="False"/>
            <string key="status-code" value="302"/>
            <string key="method" value="GET"/>
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
                :56:58.983-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
                X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
                /64.0;"/>
            <string key="concept:name" value="_oscom_account.
                php_GET_302"/>
      </event>
      <event>
            <string key="referer" value="/oscom/index.php"/>
            <string key="set-cookie" value="False"/>
            <string key="status-code" value="200"/>
            <string key="method" value="GET"/>
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
                :56:59.066-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
                X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
                /64.0;"/>
            <string key="concept:name" value="_oscom_login.
                php_GET_200"/>
      </event>
      <event>
            <string key="referer" value="/oscom/login.php"/>
            <string key="set-cookie" value="False"/>
            <string key="status-code" value="200"/>
            <string key="method" value="GET"/>
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
                :57:12.766-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
                X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
                /64.0;"/>
            <string key="concept:name" value="_oscom_products_new.
                php_GET_200"/>
      </event>
      <event>
            <string key="query_page" value="2"/>
            <string key="referer" value="/oscom/products_new.php"/>
            <string key="set-cookie" value="False"/>
            <string key="status-code" value="200"/>
            <string key="method" value="GET"/>
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
               :57:18.739-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
               X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
               /64.0;"/>
            <string key="concept:name" value="_oscom_products_new.
               php_GET_200"/>
      </event>
      <event>
            <string key="query_page" value="3"/>
            <string key="referer" value="/oscom/products_new.php"/>
            <string key="set-cookie" value="False"/>
            <string key="status-code" value="200"/>
            <string key="method" value="GET"/>
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
               :57:22.023-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
               X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
               /64.0;"/>
            <string key="concept:name" value="_oscom_products_new.
               php_GET_200"/>
      </event>
      <event>
            <string key="query_page" value="3"/>
            <string key="query_action" value="buy_now"/>
            <string key="query_products_id" value="7"/>
            <string key="referer" value="/oscom/products_new.php"/>
            <string key="set-cookie" value="False"/>
            <string key="status-code" value="302"/>
            <string key="method" value="GET"/>
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
               :57:30.094-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
               X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
               /64.0;"/>
            <string key="concept:name" value="_oscom_products_new.
               php_GET_302"/>
      </event>
      <event>
            <string key="query_page" value="3"/>
            <string key="referer" value="/oscom/products_new.php"/>
            <string key="set-cookie" value="False"/>
            <string key="status-code" value="200"/>
            <string key="method" value="GET"/>
            <string key="lifecycle:transition" value="complete"/>
            <date key="time:timestamp" value="2019-04-07T12
               :57:30.170-05:00"/>
            <string key="org:resource" value="127.0.0.1:Mozilla/5.0 (
               X11; Linux x86_64; rv:64.0) Gecko/20100101 Firefox
               /64.0;"/>
            <string key="concept:name" value="_oscom_shopping_cart.
               php_GET_200"/>
      </event>
      <event>
            <string key="concept:name" value="||end||"/>
      </event>
</trace>
</log>
```

Appendix B

The discovered positive long distance dependency relations

Page	Dependees
checkout.php:GET:200	account.php:GET:200
	add_to_cart.php:POST:302
	cart.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	view_product.php:GET:200
checkout_confirmation.php:GET:200	account.php:GET:200
	add_to_cart.php:POST:302
	cart.php:GET:200
	checkout.php:GET:200
	checkout_payment.php:GET:200
	checkout_shipping.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	view_product.php:GET:200
checkout_payment.php:GET:200	account.php:GET:200
	add_to_cart.php:POST:302
	cart.php:GET:200
	checkout.php:GET:200

	checkout_shipping.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	view_product.php:GET:200
checkout_promo_code.php:GET:200	account.php:GET:200
	add_to_cart.php:POST:302
	cart.php:GET:200
	checkout.php:GET:200
	checkout_confirmation.php:GET:200
	checkout_payment.php:GET:200
	checkout_shipping.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	view_product.php:GET:200
checkout_shipping.php:GET:200	account.php:GET:200
	add_to_cart.php:POST:302
	cart.php:GET:200
	checkout.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	view_product.php:GET:200
checkout_success.php:GET:200	account.php:GET:200
	add_to_cart.php:POST:302
	cart.php:GET:200
	checkout.php:GET:200
	checkout_confirmation.php:GET:200
	checkout_payment.php:GET:200
	checkout_shipping.php:GET:200

	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	view_product.php:GET:200
create_account_success.php:GET:200	create_account.php:GET:200
	login.php:GET:200
	create_account.php:POST:302*
edit_account.php:GET:200	account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
edit_account.php:POST:200	account.php:GET:200
	edit_account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
edit_account.php:POST:302	account.php:GET:200
	edit_account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
logoff.php:GET:200	account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
my_address_book.php:GET:200	account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
my_address_book.php:POST:200	account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	my_address_book.php:GET:200
	account.php:GET:200

my_address_book.php:POST:302

	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	my_address_book.php:GET:200
orders_history.php:GET:200	account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
payment_methods.php:GET:200	account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
payment_methods.php:POST:200	account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	payment_methods.php:GET:200
payment_methods.php:POST:302	account.php:GET:200
	login.php:GET:200
	login.php:POST:302+create_account_success.php:GET:200
	payment_methods.php:GET:200

Table B.1: Registered user role positive long distance dependency relations

Page	Dependees
admin/customers.php:GET:200	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
admin/delete_customer.php:GET:200	admin/customers.php:GET:200
	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
admin/delete_customer.php:POST:302	admin/customers.php:GET:200
	admin/delete_customer.php:GET:200
	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
admin/delete_order.php:GET:200	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
	admin/orders.php:GET:200
admin/delete_order.php:POST_302	admin/delete_order.php:GET:200
	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
	admin/orders.php:GET:200
admin/edit_customer.php:GET:200	admin/customers.php:GET:200
	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
admin/edit_customer.php:POST:302	admin/customers.php:GET:200
	admin/edit_customer.php:GET:200

	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
admin/edit_order.php:GET:200	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
	admin/orders.php:GET:200
admin/edit_order.php:POST:302	admin/edit_order.php:GET:200
	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
	admin/orders.php:GET:200
admin/invoice.php:GET:200	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
	admin/orders.php:GET:200
admin/logoff.php:GET:200	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
admin/orders.php:GET:200	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
admin/packingslip.php:GET:200	admin/index.php:GET:200
	admin/login.php:GET:200
	admin/login.php:POST:302
	admin/orders.php:GET:200

Table B.2: Admin role positive long distance dependency relations

Page	**Dependees**
cart.php:GET:200	add_to_cart.php:POST:302
	view_product.php:GET:200
create_account.php:POST:200	create_account.php:GET:200
	login.php:GET:200
forget_password.php:POST:200	forget_password.php:GET:200
	login.php:GET:200
remove_item.php:POST:302	add_to_cart.php:POST:302
	cart.php:GET:200
	view_product.php:GET:200
checkout.php:GET:302	add_to_cart.php:POST:302
	cart.php:GET:200
	view_product.php:GET:200

Table B.3: Guest role positive long distance dependency relations