# Exploring the Application of Process Mining Techniques to Improve Web Application Security

Marcelo Bruno, Pablo Ibañez, Tamara Techera, Daniel Calegari, Gustavo Betarte
*Instituto de Computación, Facultad de Ingeniería*
*Universidad de la República*
Montevideo, Uruguay, 11300
{marcelo.bruno, pablo.ibanez, tamara.techera, dcalegar, gustun}@fing.edu.uy

***Abstract*—Web applications are permanently being exposed to attacks that exploit their vulnerabilities. To detect and prevent misuse of the functionality provided by an application, it has become necessary to develop techniques that help discern between a valid user of the system and a malicious agent. In recent years, a technology that has been widely deployed to provide automated and non-invasive support for detecting web application attacks is Web Application Firewalls. In this work, we put forward and discuss the application of Process Mining techniques to detect deviations from the expected behavior of web applications. The objects of behavior analysis are logs generated by a widely deployed WAF called ModSecurity. We discuss experiments we have carried out applying our mining method on the well-known *e-commerce* platform Magento and using the ProM tool for the execution of the process mining techniques.**

***Keywords*- Security, web applications, process mining, web application firewall, ModSecurity, ProM.**

## I. Introduction

A web application is software based on a client-server architecture. The information flowing between the client and the application server is transmitted using the *HTTP(s)* protocol. Web applications are typically designed to be exposed to any individual or artifact with capabilities to access the Internet. Therefore they are a primary target for any attacker who wants to get unauthorized access to the information they handle or to place baits to lure honest users. Attackers exploit vulnerabilities, i.e., a hole or a weakness in the application, which can be a design flaw or an implementation bug. Web application vulnerabilities are listed and precisely described in the Open Web Security Project TOP 10 [1].

Although the natural path to provide more secure web applications is to perform vulnerability analysis and modify the application's code or the technological platform it runs, some scenarios require an alternative approach. For example, when the code is unavailable or correcting the vulnerability is expensive for the organization. In this context, it has become popular the application of a Virtual Patching strategy [2], e.g., using a *Web Application Firewall* (*WAF*) to protect the application. A WAF is software that intercepts and inspects all the traffic between the web server and its clients, searching for attacks inside the HTTP packet contents. Once recognized, the suspicious packets may be processed in a different secure way, for instance, been logged or suppressed. *ModSecurity* [3] is an open-source, widely used WAF enabling real-time web application monitoring, logging, and access control. ModSecurity undertakes actions driven by rules that specify, through regular expressions, the contents of the packets to be analyzed and eventually spotted. Trial-and-error is needed to adjust these rules discerning between valid and malicious behavior.

Process Mining (PM) [4] allows analyzing the event logs associated with the execution of a system's processes, considering a process as a set of coordinated tasks to achieve an objective. Process discovery techniques find behavioral (process) models that best describe the behavior within the event log. A large variety of discovery algorithms are available, e.g., the *Inductive Miner* [5], which copes with infrequent behavior and large event logs. Process models can be compared with other execution logs, measuring how much they fit. This procedure is called *conformance checking*. There exist several tools, e.g., the ProM framework [6], which provide automated support to perform PM-based analysis of systems behavior.

Few works exist on the application of PM techniques in the context of web applications [7]–[9], since they frequently embody loosely structured processes, or even worst, there is no prior definition of a process at all. In those cases, only a subset of the PM techniques is applicable. In [10] the authors propose a method based on PM to improve the security of web information systems. They assume the existence of UML models specifying the application's expected behavior, which is not always possible when the application is already in operation. In this context, it seems reasonable to exploit information registered from the applications' execution.

In this paper, we explore the application of PM techniques, being the object of analysis logs generated by ModSecurity, to detect deviations in the expected behavior of web applications. Our proposal elaborates on the method presented in [10], which we have adapted to focus on the use of the WAF to register the interaction among users (and artifacts) and the application server. This information allowed us to leverage, from a security point of view, the mining method.

The rest of the paper is structured as follows. In Section II, we discuss related work. In Section III, we describe our proposal for a security-oriented PM method. In Section IV, we present the experiments we have carried out on the well-known e-commerce application Magento and, in Section V, we discuss some limitations of the proposed approach. Finally, in Section VI, we conclude and describe future work.

## II. Process mining and web application security

Few works exist on the application of PM techniques for modeling and analyzing the behavior of web applications. In [7] the authors structure models of user behavior intended to provide insights, from a business perspective, on potential improvements of the analyzed application. Some works apply PM to address security issues [8], [9], but the primary target of the study is not web applications.

In [10] the authors propose a method based on PM to improve the security of web information systems. They specify the application's expected behavior using UML behavioral models. Their approach assumes that UML behavioral models are already available or can be obtained by through reverse engineering. They also assume that the behavioral specification is complete enough to generate a proper normative model by applying a model transformation. Although these assumptions are not always possible when the application is already in operation, they devise a platform-independent methodology to identify attack patterns by detecting deviations from the known behavior of the system, which a corresponding normative model gives. Their main ideas can be adapted to work with factual information from executing a web application.

As depicted in Figure 1, the method proposed in [10] consists of five steps: (Step 1) specification of the system behavior using Unified Modeling Language (UML) sequence diagrams; (Step 2) automatic generation of a formal (normative) model from the UML-based specification; (Step 3) acquisition of real behavior logs, which are obtained as the result of monitoring the web information system to get data representing its operational behavior; (Step 4) preprocessing of the registered behavior using process mining techniques to get helpful event logs; and finally, (Step 5) identification of behavioral deviations applying PM techniques to perform conformance checking between the normative model and the operative behavior logs.

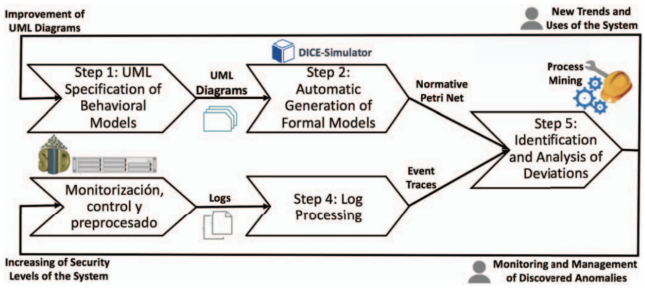


Fig. 1: Mining attack patterns in information systems [10].

We propose to adapt this method by using a WAF to register the behavior generated from the response of a web application to HTTP requests. It makes it possible to leverage the mining method in several ways, as described in what follows.

*Specification of the system behavior:* Instead of using a UML diagram to specify the application's expected behavior, we have chosen to register valid executions that can be captured using the WAF ModSecurity. The WAF logs capture the application's execution for each HTTP request directed to the protected application. ModSecurity is configured on an Apache server, pointing as a reverse proxy to the website in question. In this way, the information can be collected by merely browsing the application.

*Generation of a normative model:* As was already mentioned, valid behavior logs can be used to obtain the normative model of the system. With the help of simple software tools, it is possible to transform the files generated by ModSecurity to a single log in XES format (a standard used by the ProM framework). From the generated XES log, a Petri Net [11] can then be obtained using a discovery algorithm. Since the initial logs are the product of navigation of the website, the different activities of the Petri net correspond to the URLs of the site to which users were accessing during the (adequate) use of it. For this reason, we have decided to filter static content, such as requests for CSS files and images, since they do not provide valuable information regarding user behavior.

*Obtaining real behavior logs:* Using ModSecurity, it is possible to obtain information from navigation carried out by well-intentioned users and generated automatically by navigations belonging to potential or attempted attacks. This latter probing of the application can be performed using a dedicated tool. It is worth mentioning that each log file contains, among others, basic information about the HTTP requests and the rules that the WAF has applied in the request corresponding to the log file. These rules are incorporated as information in all the logs generated in the system.

*Pre-processing of the logs:* We had to face the granularity of the registered data since the applications generate low-level records based on HTTP requests. A model generated from the navigation through a website without any refinement is unstructured (a.k.a., *spaghetti* processes). Figure 2 shows a model discovered using a log from the example applications that is described in Section IV.

Due to the complexity of the web application and the fact that usually, there exist accessible URLs that correspond to elements that are not relevant for vulnerability analysis, we have defined filters that allow us to disregard access to URLs of that kind. Therefore, in contrast to what is described in [10], our method incorporates a data filtering stage and a strategy for identifying critical areas to narrow the analysis phase. A *critical area* is defined as a set of activities around certain parts of the web that may involve a possible risk for the system, such as a login procedure. Another point to consider is that the logs available for the study do not contain fields that refer to user sessions. Different heuristics can be considered to obtain that information from the logs using the IP values, the elapsed time, or the browser. If the system allows more than one user, this data could also identify the application user. One option is to consider that if the HTTP requests registered in ModSecurity come from the same IP, the same user interacts with the system in a work session. We followed this approach, and a deeper exploration of this problem is left for future work.

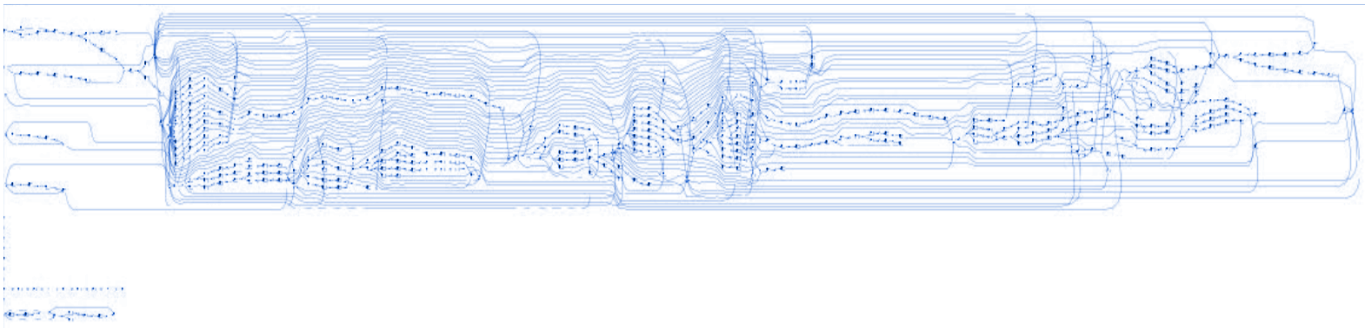

Fig. 2: Model discovered from a well-intentioned navigations on a site.

*Identification of deviations:* The conformance checking procedure identifies deviations from a normative model. It compares a normative model with an event log of the same process, checking if the actual execution of the process (i.e., the event log) conforms to the model. A widely used deviation measure is the notion of each trace's fitness value, which takes value in the range 0..1. The lowest fitness value (0) corresponds to the worst possible alignment case (the log does not conform to the model), and the highest (1) corresponds to a perfect alignment without penalties (the log fits perfectly with the model).

## III. Mining web application behavior from ModSecurity logs

As mentioned in the previous section, we make extensive use of logs generated by the WAF ModSecurity for analyzing deviations from the expected behavior of web applications. In the first place, we use logs generated from regular use to discover a corresponding normative model using process mining algorithms. That model shall be the typical behavior reference to compare when analyzing logs that might register abnormal or malicious web system uses.

In what follows, we proceed to describe the security-oriented PM method depicted in Figure 3, which constitutes an adapted formulation on the one presented in [10].

### A. Log-based application behavior modeling

On *"Step 1: Specification of the system behavior"* depicted in Figure 3, we configure the WAF ModSecurity to generate a log file for each HTTP request sent to an application. We developed the tool *ParserModSecurity* [12] a Java console application that can be used to transform a ModSecurity log into a corresponding XES log, which in turn can be handled by PM tools. In Listing 1 we show a fragment of a ModSecurity log file corresponding to a given HTTP request. Section A shows the client's IP, section B the method and resource associated with the request (in this case, a catalog search), section F the response header (200, a successful request), and section H the rules activated by ModSecurity (in this particular case there is no rule).

Listing 1: Example ModSecurity log (excerpt).

```
--4c358d0e-A--
[19/Jul/2020:17:20:08+0000]
   XxSAxawRAAQAAAAM9LEAAABL 172.17.0.1
   55218 172.17.0.4 80
--4c358d0e-B--
POST /catalogsearch/result/?q=aa
Host: modsecurity-magento
--4c358d0e-F--
HTTP/1.1 200 OK
--4c358d0e-H--
```

The tool *ParserModSecurity* transforms this event into the XES fragment shown in Listing 2. The attribute `org:resource` corresponds to the user's IP who made the request. The attribute `time:timestamp` contains the date and time in which the request was made. The attribute `concept:name` represents an activity. It corresponds to the specific web resource requested by the user. Its value provided by ModSecurity corresponds to the original URL of the requestor. However, as will be explained later, we perform a grouping transformation to simplify requests. The attribute `debug:originalUri` corresponds to the original URL of the request. Finally, `idRules` contains a comma-separated list of all the ModSecurity rules activated by the request. In the example, the H section is empty since this section generates logs with valid behavior that do not generate rules. Each trace corresponds to an execution of a user, which corresponds to a particular IP (i.e., the case id), to distinguish different users.

Listing 2: Example XES log (excerpt).

```
<event>
   <string key="org:resource" value="
      172.17.0.1"/>
   <date key="time:timestamp" value="
      2020-07-19T17:20:08"/>
   <string key="concept:name" value="
      POST:catalogSearchResult"/>
   <string key="debug:originalUri" value=
      "POST:/catalogsearch/result/?q=a"/>
   <string key="idRules" value=""/>
</event>
```

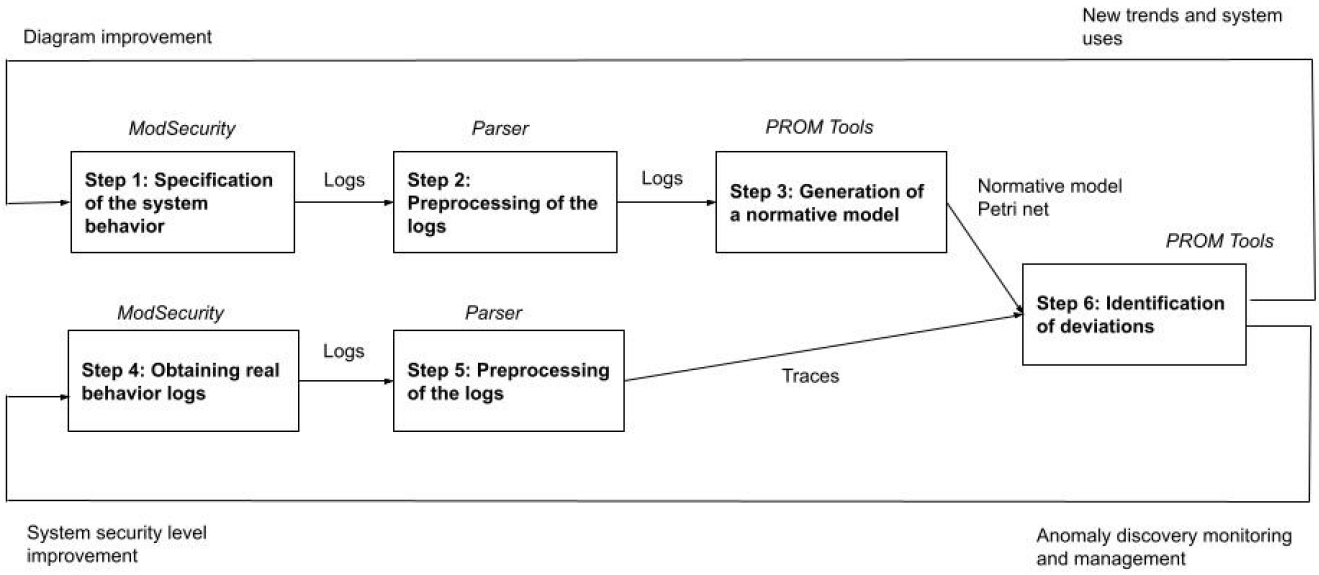


Fig. 3: Method overview.

Suppose an HTTP request activates a ModSecurity rule (within the H section). In that case, that rule is added in the `idRule` attribute of the XES event, and an event is created that represents such activation. These actions allow explicit the activated rules in the generated models and intentionally penalize one such rule when executing conformance checking against a normative model, affecting its fitness values negatively. It is assumed that a model generated with information from well-intentioned users does not contain that kind of rule.

In Listing 3 we show the event of Listing 2 representing the request of the resource `POST:catalogSearchResult` together with the activation of two rules. In such cases we also add two more rule events `RULE:Id=1` and `RULE:Id=2`.

Listing 3: Example XES log with rules (excerpt).

```
<event>
   ...
   <string key="concept:name" value="
      POST:catalogSearchResult"/>
   ...
   <string key="idRules" value="1,2"/>
</event>
<event>
   <string key="org:resource" value="
      172.17.0.1"/>
   <date key="time:timestamp" value="
      2020-07-19T14:27:38.038-0300"/>
   <string key="concept:name" value="
      RULE:Id=1"/>
   <string key="debug:originalUri" value=
      "RULE:/catalogsearch/result/?q=aaa"
      />
   <string key="idRules" value=""/>
</event>
<event>
   <string key="org:resource" value="
      172.17.0.1"/>
   <date key="time:timestamp" value="
      2020-07-19T14:27:38.038-0300"/>
   <string key="concept:name" value="
      RULE:Id=2"/>
   <string key="debug:originalUri" value=
      "RULE:/catalogsearch/result/?q=aaa"
      />
   <string key="idRules" value=""/>
</event>
```

It should be noted that each event, which is the execution of an activity by a user in a given time, is contained within a *trace* in the XES log, which represents an execution of a user (with a concrete IP), to distinguish different users. Likewise, there may be several traces corresponding to the same IP of the same user, since in our proposed method, as shall be explained later, the traces are separated by critical areas.

### *B. Security analysis-oriented log selection*

Once the logs have been generated, on *"Step 2: Preprocessing of the logs"* depicted in Figure 3 we filter information that is not relevant for the analysis, and that additionally might prevent the correct working of PM techniques. Three procedures, of different nature, are carried out: i) records that are not relevant for the security analysis are filtered out, ii) certain logs are unified into a single activity (e.g., loop) or into a single record type (e.g., collapsing a certain set of web pages), and iii) only certain parts of the web that are considered critical are considered.

Filters of different nature can be applied:

- Static web application elements such as images, CSS style sheets, and JavaScript files are filtered since they do not provide information about user behavior on the web application.
- It is interesting to filter out certain intermediate requests that do not add value to the analysis. An example of this is search fields, which are very common on AJAX websites.
- It is often of interest to filter ModSecurity rules applied to a certain dataset known to have no impact on web application security.

*ParserModSecurity* tool applies these filters by defining rules using JSON configuration files. Figure 4 shows an example of a configuration rule for static elements.

```
"extension_filter": {
        "enabled": "true",
        "parameters": [".css", ".js", ".svg", ".jpg", ".png", ".ico", ".gif"]
        },
```

Fig. 4: Filter rule example in ParserModSecurity

We then **group URLs** that are not interesting to be considered as separate events. The activities are being identified from URLs, but some have parameters that, if not previously processed, will be reflected in the model as different activities. For example, the following URLs are generated from viewing a review of two different products on one website:

/opinion/product/id/699/
/opinion/product/id/700/

ModSecurity generates two different activities, but the interest to be modeled is consulting the opinion of a product, not the specific product. We group both into a single activity to display the opinion of a product.

| Original URLs | Grouped URL |
|---|---|
| /opinion/product/id/699/<br>/opinion/product/id/700/ | productOpinion |

We finally refine the log by focusing on what we call critical areas, defined in Section II. Despite the filtering and groupings carried out, the models generated from these logs are complex. Typically, a web application has several links that can be accessed from many different places. In this case, it is impossible to identify a predefined process or user behavior. Given that the focus of our research is to detect evidence of inappropriate behavior, which could potentially have the objective of violating some security properties, we have chosen to focus on portions of the web application. We select certain traces that may involve a possible risk for the system (i.e., a critical area), such as a login procedure, access to input fields, or bank transactions. Activities can also be selected based on whether they have triggered ModSecurity rules.

We generate a trace for each critical area. For example, to analyze the behavior of users when using a functionality, such as consulting an item on the site, we take a fixed number of previous events and the same fixed number of events following the execution of the functionality mentioned above. This number is called the *window of the critical area*. For example, if we consider the trace

a1 a2 a4 a5 a7 **a3** a9 a10 a11 a12

and we define the critical area around an activity **a3** with a window of length 2, we get the trace

a5 a7 **a3** a9 a10

Repeated activities are not considered while counting the previous and subsequent activities. If we consider the trace

a1 a2 a2 a3 **a4** a5 a5 a5 a6 a7

and we define the critical area around an activity **a4** with a window of length 2, we get the trace

a2 a2 a3 **a4** a5 a5 a5 a6

In this case, the resulting trace ends up having a length of 8 instead of 5 if there were no repeated activities. This rule avoids the loss of pre or post-information due to loops.

The activated rules are not counted for this window either. Therefore, if we consider the trace

a1 a2 a3 **a4** a5 r1 a6 a7

and we define the critical area around an activity **a4** with a window of length 2, being `r1` an activated rule, we get the trace

a2 a3 **a4** a5 r1 a6

Using these rules for filtering critical areas, we can generate an XES file for each such area. Notice that we can extract multiple critical area logs from the ModSecurity log registered from a user since the user can cross an area many times. It generates more extensive but focused event logs.

### C. Log-based generation of normative models

On *"Step 3: Generation of a normative model"* depicted in Figure 3 we use the ProM framework to apply PM techniques to discover process models from the logs and perform conformance checkings. More precisely, we use the *Inductive Miner Infrequent* [5] algorithm. This variant of the classical *Inductive Miner* is recommended because, in addition to guaranteeing the generation of a sound model, it filters infrequent behavior in every step whose result is a flower model, which significantly improves its limitation to generate adequate models and its precision. The precision metric could also be used together with fitness on the normative model after generating it to have a more strong quality assessment.

To model behavior within a critical area, it is necessary to define the corresponding window. If the window is too small, the appropriate behavior before and after the critical event might not be observed. If the window is too high, the logged events might occur too far from the critical event and do not contribute to understanding that event. Thus, we used an empirically inspired criterion to select those values: the window chosen is the one that obtains the best fitness value concerning the normative model.

### D. Obtaining (almost)real behavior logs

Once the normative model has been generated, we can compare it against the actual observed behavior of the web application and identify possible deviations concerning the expected behavior. If there are deviations, they must be analyzed in greater detail since they could imply a potential attack on the system. Using the WAF ModSecurity, on *"Step 4: Obtaining real behavior logs"* depicted in Figure 3, we generate logs of everyday use of the system by users. It is enough to have the WAF correctly configured.

We propose to generate two sets of logs to perform conformance checking against the same normative model and study the results. The first one is a set of logs obtained from the well-intentioned use of the site, a different one from the one used in the discovery stage. We can verify that traces are different from those who allow the normative model discovery, with high fitness values when performing conformance checking against it. The second one is a set of logs obtained from attacks. With this objective in mind, we use the OWASP Zap [13] tool, an open-source security scanner that provides various functionalities and modes of operation. In particular, its attack mode allows specifying a website URL, and the tool scans the site and executes malicious requests, testing different types of attacks. Then, the logs generated by ModSecurity constitute an attack log.

### E. Preprocessing of the logs and identification of deviations

On *"Step 5: Preprocessing of the logs"* depicted in Figure 3, the tool *ParserModSecurity* takes as input ModSecurity logs and returns as output XES logs, which are suitable for applying PM techniques within ProM. The parser also uses the filtering and grouping techniques described above according to the configuration that is set. The next step is to perform conformance checking of these logs against the normative model. Thus, the parser must have the same configuration, i.e., the same filtering and grouping strategies.

After having generated and processed the logs, on *"Step 6: Identification of deviations"* depicted in Figure 3, we use ProM to perform conformance checking using alignments [14]. It allows us to see the deviations of the logs according to the normative model. It should be considered that there will be a log per user, group of users, or critical area, expanding the type of studies that can be carried out.

We consider the following data to interpret the results and identify potentially malicious behavior:

- Average fitness value of traces.
- Standard deviation of the fitness of traces.
- Minimum fitness value of traces.
- Length of traces.

The first value to consider is the average fitness of the log. An average fitness value can be high (higher than 0.8) and its standard deviation low, and it can also be a log that implies an attack or attempted attack on the site. Here are some reasons why this can happen:

- Many attacks are carried out using the HTTP request body, and that information is not used in the generation of the logs, so these requests are not properly reflected.
- The attacks may involve many requests to the same activity or set of activities, represented as loops in the model. Still, the model is not detected as something wrong, although it may be suspicious behavior.
- By grouping URLs, malicious URLs can be grouped into valid URLs, and, in this way, those traces have their correspondence in the normative model, even if this is incorrect.
- There can be only some traces with invalid behavior. As these are only a few, this may not affect the average fitness. For these cases, it is necessary to see each trace's fitness detail or consider the log's minimum fitness to detect the traces with undesirable behavior.

The fitness will be low in the case of an attack where ModSecurity rules are activated. As they are added in the traces as events, the value of fitness will be low since, in the normative model, there are no modeled rules. Likewise, if the average fitness is low, it does not necessarily imply a log with invalid behavior. It is possible to have a low fitness value because the behavior of the log was not modeled in the normative model, but it is proper behavior.

The following steps must be followed to interpret the results obtained from performing the conformance checking of the log against the model:

- If the minimum fitness value:
  - Is low, so the trace with minimal fitness should be analyzed, and look for more traces with low fitness, to analyze if they are attack cases.
  - Is high (close to 1), it may or may not be an attack. A more in-depth analysis of the cases should be carried out to ensure that it is not a malicious log.
- If there are very long traces, it should be verified if it is due to an excessive amount of loops or rules activation. It can be malicious behavior in both cases, so they should be analyzed in more depth.

## IV. Experimentation

We present the results of experiments we have carried out concerning applying the proposed mining method on a concrete web application. The experiment's target is the *Magento* e-commerce platform [15], an open-source platform that offers a flexible shopping cart system. It is written in PHP and uses a MySQL relational database.

### A. Generation of the normative model

For the generation of the normative model, we have deployed and made accessible on the Internet an instance of the Magento application. Approximately 60 users accessed the site, performing non-malicious browsing of the application and generating about 22.500 Modsecurity audit log files that correspond to HTTP requests made by those users.

Then, using ParserModSecurity, the generated logs were transformed into XES format, filtered out, and grouped.

In the first place, URLs involving requests for CSS, JavaScript, and image files were filtered out. Additionally, each URL associated to a GUI-specific element that does not provide information about user behavior on the site, was also filtered. Example of that kind of resource are static elements that start with either of the following formats:

/pub/static/
/customer/section/
page_cache/block/render/

URLs representing equivalent actions over the system were grouped into a single abstract activity representing those actions. This grouping allows us to reduce spaghetti models, like the one depicted in Figure 2, which would prevent reliable results out of a conformance checking. As an example, one of the groupings performed relates to a particular kind of query on the site menu. In that menu, product categories are exposed and classified into subcategories, as shown in Figure 5. The corresponding URLs were grouped into two abstract activities: *menuCategory* and *menuSubcategory*, for URLs providing access to categories and subcategories, respectively.

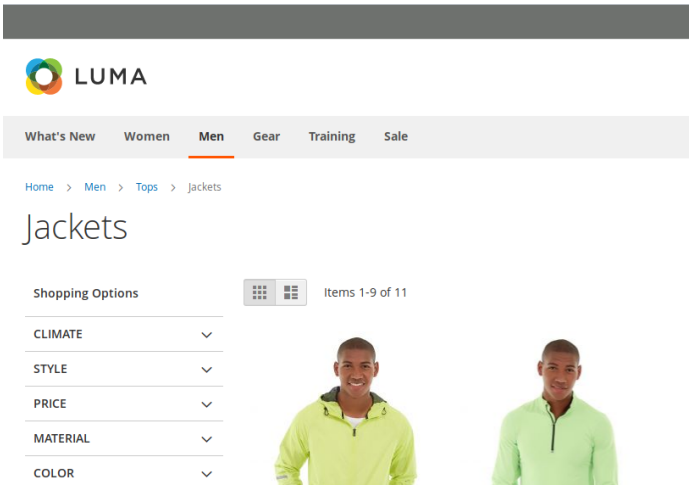


Fig. 5: Categories present in the Magento site, e.g., within the Men category there is Tops category, and within it there is Jackets category.

As the final step of the preprocessing step, we have defined several **critical areas** that were considered especially vulnerable to possible attacks. We have used the OWASP ZAP [13] tool to generate attacks on the site. It is possible to spot activities that significantly triggered ModSecurity rules, like the *addProductToCart* related to adding a product to the site's shopping cart *catalogSearchResult* concerning searches in the products catalog. We decided to study critical areas surrounding both activities. For this, it was necessary to define the window of activities that make up these critical areas. The objective is to have a window value that is not too small for the model to have a single activity, nor too large that the critical zone becomes meaningless. The larger is the trace, the fewer related events are recorded with critical functionalities.

Table I shows the lengths of the traces for the *addProductToCart* functionality using different windows. The greater the window, the greater the average and maximum length of the traces. For example, with a window of length 10, the maximum trace length is 40 events, quite far from the mean of 22, close to the maximum number of different activities in a log with a window of length 10. We took 10 as the maximum length since trace length can reach high values with larger windows.

TABLE I: Lengths of the traces used for the *addProductToCart* functionality w.r.t. different window length.

| | Trace lenght | | |
|---|---|---|---|
| **Window** | **Avg** | **Min** | **Max** |
| 1 | 3.47 | 3 | 9 |
| 2 | 5.81 | 4 | 11 |
| 3 | 8.1 | 5 | 20 |
| 4 | 10.36 | 6 | 26 |
| 5 | 12.47 | 7 | 29 |
| 6 | 14.42 | 8 | 21 |
| 7 | 16.31 | 9 | 24 |
| 8 | 18.16 | 9 | 36 |
| 9 | 19.9 | 9 | 38 |
| 10 | 21.55 | 9 | 40 |

Fitness values in Table II have been obtained from conformance checking performed between the normative model generated for each window and the same log that generated it, bounded to that same window. We considered the three window values with the highest average fitness: 6, 8, and 9 with values 0.94, 0.96, and 0.94, respectively. These three values are very similar, and it would be acceptable to use any of them. In this case, the model of a window of length 6 was chosen because it had smaller traces and a greater number of cases with a perfect fitness of 1. Figure 6 shows the Petri net model obtained for a window of length 6, using the Inductive Mining Infrequent algorithm executed in ProM.

TABLE II: Fitness values of the traces used for the *addProductToCart* functionality w.r.t. different window length.

| | Trace Fitness | | | | | |
|---|---|---|---|---|---|---|
| **Window** | **Avg/ Case** | **Max** | **Min** | **Std Deviation** | **Cases with value 1.0** | **Total cases** |
| 0 | 0.94 | 1 | 0.50 | 0.14 | 112 | 137 |
| 1 | 0.87 | 1 | 0.40 | 0.17 | 74 | 137 |
| 2 | 0.72 | 1 | 0.38 | 0.15 | 15 | 137 |
| 3 | 0.92 | 1 | 0.56 | 0.11 | 80 | 137 |
| 4 | 0.87 | 1 | 0.50 | 0.13 | 46 | 137 |
| 5 | 0.84 | 1 | 0.50 | 0.14 | 39 | 137 |
| 6 | 0.94 | 1 | 0.57 | 0.08 | 73 | 137 |
| 7 | 0.91 | 1 | 0.47 | 0.10 | 47 | 137 |
| 8 | 0.96 | 1 | 0.59 | 0.07 | 72 | 137 |
| 9 | 0.94 | 1 | 0.57 | 0.08 | 60 | 137 |
| 10 | 0.82 | 1 | 0.45 | 0.15 | 30 | 137 |

Although there is no formal definition for differentiating structured from unstructured processes, a widely used rule is to verify that the conformance checking is greater than 0.8 [16]. In the case of *addProductToCart*, its fitness value is 0.94, so by this rule, it could be assumed that the resulting model is structured enough. In the case of *catalogSearchResult*, the average fitness for all windows is less than 0.8.

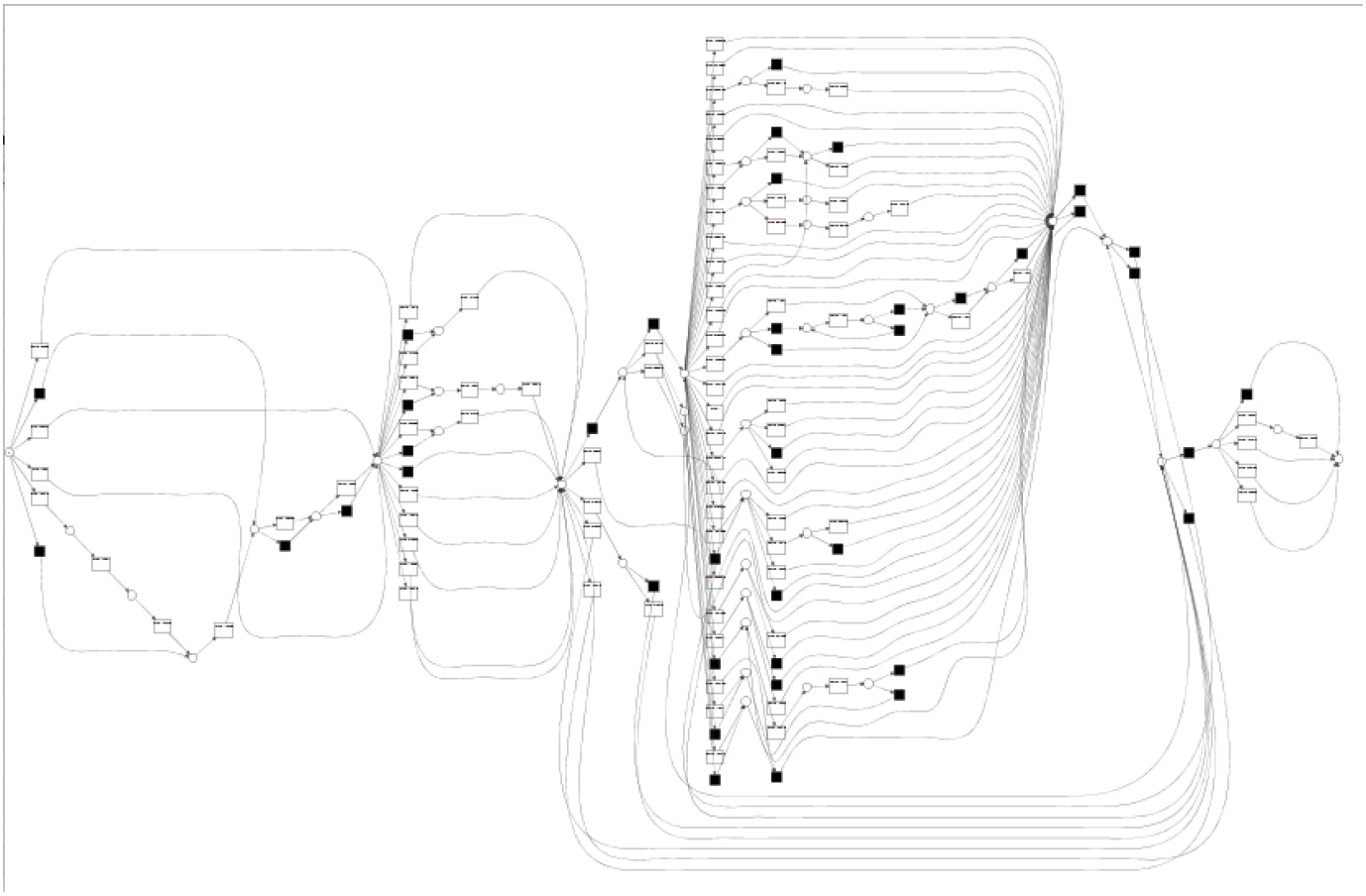

Fig. 6: Petri net model corresponding to the *addProductToCart* log with window of length 6

When considering the number of traces that repeat the same path in the model, it can be noted that access to the *addProductToCart* functionality is more restricted than *catalogSearchResult* since to add a product to the cart, certain preconditions must be met. With a sufficient number of traces, which varies depending on the expected behavior, more heterogeneous traces can indicate habitual behaviors that are not recorded. Also, more homogeneous traces give the guideline that if other behaviors exist, they will correspond to exceptional cases. In the case of *addProductToCart*, 132 traces out of 137 have a common path, while in the case of *catalogSearchResult*, 52 traces out of 63 have a unique path. As a result, the lower average fitness value of *catalogSearchResult* does not ensure reliable results. The problem is that it does not has a process-like behavior but single and heterogeneous events that are not easily evaluated through PM. In this case, another kind of security analysis should be carried out, which is left as future work. We omit the analysis of *catalogSearch* in the rest of the sections.

### *B. Identifying deviations from the normative model*

As already mentioned, we have performed several executions of the OWASP ZAP [13] for injecting malicious requests. Those requests were audited and registered using ModSecurity, giving rise to what we call the *ZAP* log.

We have also generated what we call a *Valid* log. It was obtained from registering navigations of the site that make regular and straightforward use of it. These navigations are similar to those carried out by most of the users who visited the site from which logs were generated to create the normative model (called the *Users* log).

Moreover, the *Test* log from navigations related to unusual behavior within the website, such as browsing without following the order indicated by the user interface, pretending to be a testing user. These navigations do not involve an attack on the site. Still, as they represent behavior that is unlikely to be included in its entirety by the mandatory model, they could be identified as incorrect and potentially malicious behavior.

An additional log (*Valid with Attack*) was created from the Valid log, adding an attack trace extracted from the ZAP logs. In the case of *addProductToCart*, this trace contains 97 events, of which 50 correspond to activated rules. This log is used to evaluate how sensible the proposed method is, i.e., if it can detect outliers within a large log with valid behavior.

Table III shows the different logs, their total number of traces, and the maximum number of events within a trace. All the logs mentioned above were then processed using ParserModSecurity, using precisely the same configuration settings as the generation of the normative model.

TABLE III: Max and total length of the traces of each log.

| Log | Max. events in a trace | Total traces |
|---|---|---|
| Valid | 17 | 7 |
| ZAP | 141 | 323 |
| Test | 17 | 6 |
| Valid with Attack | 97 | 8 |
| Users | 31 | 137 |

We performed conformance checking using alignments of the normative model against the logs. To identify whether the logs present attacks on the site or not, we consider their fitness value and standard deviation. These values for each log corresponding to the *addProductToCart* functionality are shown in Figure 7 and Figure 8

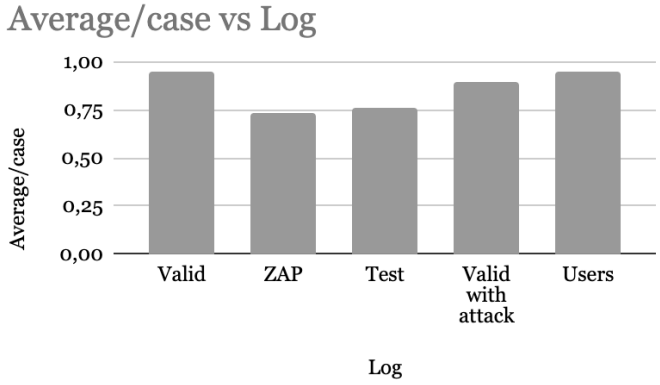


Fig. 7: Average/case fitness value of each log.

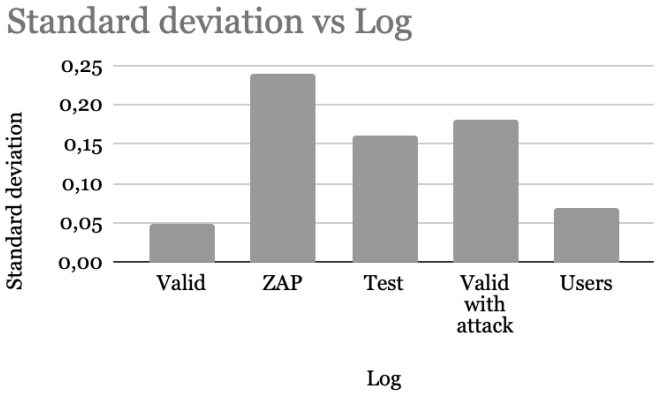


Fig. 8: Fitness standard deviation of each log.

As shown in Figure 7, the minimum fitness of the Valid log is high, and its standard deviation (in figure 8) is very low. It indicates the proper behavior of the activity registered by that log. As shown in the table III, no long traces could imply activation of rules or iterations on the same activity.

On the other hand, the minimum fitness is very low for the ZAP log, so the particular trace and the traces with low fitness should be better analyzed, as this is likely an attack on the site. In addition, its standard deviation is higher than the rest of the logs, indicating varied behavior. That is, the fitness values are not uniformly distributed concerning the average. It implies possible suspicious behavior. Table III also shows that the maximum length of a ZAP trace is much longer than the others. Looking for the longest trace, it is easy to see that it corresponds to ModSecurity rules events.

The remaining logs (Test and Valid with Attack), have a minimum fitness between 0.47 and 0.59, so must be analyzed further. The Valid with Attack log has a maximum trace length greater than the User or Valid logs. Furthermore, as a window of length 6 activities was defined, presenting a trace of length 97 could be because it contains many iterations of activities, which may be suspicious, or ModSecurity rules were activated since neither the loops nor the rules count for the window. In this case, there is a single trace longer than the others in which many ModSecurity rules were activated. The Test log has a maximum length of trace with a value of 17, which indicates no large loops or a trace that could activate many rules. It may be grasped as behavior not identified in the normative model. In that case, the decision is whether or not such behavior will be allowed on the site. Loops represent repeated access on the same activity or set of activities, where specific parameters of the request may vary. It could imply a possible attack on the site since it is not usually a typical site user's typical behavior.

The logs we have just presented and discussed might be separated into logs that contain attacks (ZAP and Valid with Attack) and logs that do not (Users, Valid, and Test). One question is whether that separation could be defined as the analysis of those logs' fitness values. The average fitness value is not helpful in logs with valid mixed traces and a few traces with an attack. They can even have an average fitness greater than some log without attacks but with unusual behavior. For example, the average value of Valid with Attack is greater than the one of the Test log. On the other side, the logs Users, Valid, and Test have a minimum fitness value of 0.57, 0.86, and 0.59, respectively, while the logs ZAP and Valid with Attack have a minimum fitness value of 0.34 and 0.47, respectively. It seems to suggest a differentiation criterion.

## V. Discussion

Although PM techniques are promising in detecting malicious behavior, some limitations need to be further studied.

The experience shows that web applications are subjected to corrective and preventive maintenance throughout their lifespan. Even if what we are presenting in this work are techniques that help spot vulnerabilities of running applications, we are convinced that security must be present in applications by design. In particular, the use of PM techniques to perform vulnerability analysis of applications may greatly benefit from the use of design criteria that helps to avoid the generation of spaghetti models like the ones illustrated in this paper. For instance, Web applications allow accessing some functionality from many places simultaneously, giving rise to that kind of behavior, making it challenging to extract a common behavior pattern. For this reason, before exposing this kind of functionality, the developers must consider the balance between

usability and security. For example, a designer could consider structuring a process-like behavior around critical areas, e.g., requiring login before adding a product to a cart. Nevertheless, a challenge is identifying critical areas in advance, in which it must be considered common security risks and the main business interest for the organization.

Without proper preprocessing, we are convinced that a discovered reference model can provide no helpful information. It is essential, for instance, to correctly group the URLs in the log preprocessing stage since this can be a critical point in determining whether some attacks are detected. On the other side, the grouping of URLs may give rise to problems since, in some cases, information about specific attacks might be lost. A complementary approach could be moving event log activities from low-level to high-level, following the approach in [17].

As illustrated with the radius of a critical area, there must be an adjustment phase to define the most appropriate granularity for grouping events. Another issue is that, as was the case in [10], we have assumed that the IP determines the user session (i.e., the case id within an event log). However, a common strategy of attackers is to change such IPs to hide their identity. In real environments, it requires more advanced heuristics to identify the user session considering, for instance, whether the request comes from the same browser, the time elapsed between events, or the parameters of a request. It is an open problem within the community of PM [18].

Concerning the identification phase, not every attack can be identified from the URLs that the user access. For example, attacks are carried out on the bodies of HTTP messages that present valid URLs, and therefore, they are considered valid activities. Moreover, loops on activities are not detected by analyzing the fitness of models. However, some attacks try to exploit some vulnerabilities by performing multiple requests to the same resource. For this reason, the length of the loops must be considered in the log analysis since it is a way to detect suspicious behavior. These aspects can be indirectly seen in both cases by adding ModSecurity triggered rules as events within a log. When a log presents rules as events, the conformance checking presents the worst fitness results.

The techniques we have explored so far do not make it possible to provide decisive arguments to identify whether a log contains attacks or not. However, we are convinced that they constitute an excellent means to approximate the correct characterization of the application´s use and behavior. If the logs do not fit the model, further analysis must help identify an attack or unspecified behavior. If they are well suited, it is more likely that they represent inappropriate behavior.

## VI. Conclusion and further work

In this work, we have presented initial results regarding the application of PM techniques to improve the security of web applications. Our working thesis is that the well-known benefit of applying PM to discover the usual behavior of application users can be used to identify possible vulnerabilities and malicious activity directed towards the same application.

We have adapted previous work by integrating a Web Application Firewall that makes it possible to generate security-enriched logs that register interactions with a web application. In particular, we provide the basis to define a normative model using a discovery algorithm from the logs generated by the WAF ModSecurity. We also put forward criteria that guide to perform a security-oriented selection of the activity registered by ModSecurity. We identified critical areas to focus the security analysis, and we also incorporated into the logs the ModSecurity rules that were activated. These suspicious interactions with the application allow considering potential behavioral deviations during conformance checking.

As discussed in the previous section, there are several research problems to pursue as future work. Most notably, the identification of user sessions from attack logs and enhancing the methodology for analyzing attacks that are not detected by just using fitness as a metric.

## References


[1] OWASP and P. R. P. 73, *OWASP Top 10: The Top 10 Most Critical Web Application Security Threats Enhanced with Text Analytics and Content by PageKicker Robot Phil 73*. CreateSpace, 2014.

[2] OWASP, "Virtual patching best practices," http://owasp.org/www-community/Virtual_Patching_Best_Practices, accessed: 2021-04-10.

[3] C. Folini and I. Ristic, *ModSecurity Handbook, Second Edition*, 2nd ed. London, GBR: Feisty Duck, 2017.

[4] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[5] Sander J.J. Leemans, Dirk Fahland, Wil M.P. van der Aalst, "Discovering Block-Structured Process Models From Event Logs Containing Infrequent Behaviour," Eindhoven University, Tech. Rep., 2009.

[6] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, "The prom framework: A new era in process mining tool support," in *Applications and Theory of Petri Nets 2005*. Springer, 2005, pp. 444–454.

[7] N. Poggi, V. Muthusamy, D. Carrera, and R. Khalaf, "Business process mining from e-commerce web logs," in *Business Process Management*. Springer, 2013, pp. 65–80.

[8] W. M. P. van der Aalst and A. K. A. de Medeiros, "Process mining and security: Detecting anomalous process executions and checking process conformance," *ENTCS*, vol. 121, pp. 3–21, 2005.

[9] R. Accorsi, T. Stocker, and G. Müller, "On the exploitation of process mining for security audits: The process discovery case," in *Proc. of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. ACM, 2013, p. 1462–1468.

[10] S. Bernardi, R. P. Alastuey, and R. T. Lado, "Using process mining and model-driven engineering to enhance security of web information systems," in *2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops*. IEEE, 2017, pp. 160–166.

[11] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[12] T. Techera, P. Ibañez, and M. Bruno, "*ParserModSecurity*," https://gitlab.fing.edu.uy/open-coal/ParserModSecurity, accessed: 2021-04-10.

[13] OWASP, "Owasp zap," https://www.zaproxy.org/, accessed: 2021-04-10.

[14] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley*, vol. 2, no. 2, pp. 182–192, 2012.

[15] Magento, "Official site," https://magento.com/, accessed: 2021-04-10.

[16] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[17] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, and P. J. Toussaint, "From low-level events to activities - A pattern-based approach," *EMISA Forum*, vol. 37, no. 1, pp. 47–48, 2017.

[18] R. Pérez-Castillo, B. Weber, I. G.-R. de Guzmán, M. Piattini, and J. Pinggera, "Assessing event correlation in non-process-aware information systems," *SoSym*, vol. 13, no. 3, pp. 1117–1139, 2014.