Adversarial ModSecurity: Countering Adversarial SQL Injections with Robust Machine Learning

Biagio Montaruli
biagio.montaruli@sap.com
SAP Security Research and EURECOM
Mougins, France

Luca Demetrio
luca.demetrio@unige.it
University of Genova and Pluribus One
Genova, Italy

Andrea Valenza
andrea.valuenza@prima.it
Prima Assicurazioni
Milano, Italy

Battista Biggio
battista.biggio@unica.it
University of Cagliari and Pluribus One
Cagliari, Italy

Luca Compagna
luca.compagna@sap.com
SAP Security Research
Mougins, France

Davide Balzarotti
davide.balzarotti@eurecom.fr
EURECOM
Biot, France

Davide Ariu
davide.ariu@pluribus-one.it
Pluribus One
Cagliari, Italy

Luca Piras
luca.piras@pluribus-one.it
Pluribus One
Cagliari, Italy

ABSTRACT

ModSecurity is widely recognized as the standard open-source Web Application Firewall (WAF), maintained by the OWASP Foundation. It detects malicious requests by matching them against the Core Rule Set (CRS), identifying well-known attack patterns. Each rule in the CRS is manually assigned a weight, based on the severity of the corresponding attack, and a request is detected as malicious if the sum of the weights of the firing rules exceeds a given threshold. In this work, we show that this simple strategy is largely ineffective for detecting SQL injection (SQLi) attacks, as it tends to block many legitimate requests, while also being vulnerable to adversarial SQLi attacks, i.e., attacks intentionally manipulated to evade detection.

To overcome these issues, we design a robust machine learning model, named AdvModSec, which uses the CRS rules as input features, and it is trained to detect adversarial SQLi attacks. Our experiments show that AdvModSec, being trained on the traffic directed towards the protected web services, achieves a better trade-off between detection and false positive rates, improving the detection rate of the vanilla version of ModSecurity with CRS by 21%. Moreover, our approach is able to improve its adversarial robustness against adversarial SQLi attacks by 42%, thereby taking a step forward towards building more robust and trustworthy WAFs.

KEYWORDS

web application firewalls, sql injection, adversarial training

1 INTRODUCTION

Web applications are constantly evolving and deployed at a broad scale, thus enabling organizations to offer rich services over the Internet. However, this imposes serious challenges in securing web applications against an increasing number of attacks [16]. Among these, SQL injection (SQLi) consists of injecting a malicious SQL code payload inside regular queries, causing the target web application to behave in an unintended way or expose sensitive data. Even if many countermeasures to this attack have been proposed [3, 4, 19, 22], the OWASP Foundation still classifies it as one of the top-10 most dangerous web threats [30]. To counter such attacks and protect web applications, WAFs are commonly used as a defense tool in enterprise systems [3, 5]. They work by filtering the incoming requests directed towards the protected applications, blocking suspicious connections.

ModSecurity [15] is an established open-source WAF solution that builds its defense on top of signatures of well-known attacks, collected by the OWASP Foundation and known as the Core Rule Set (CRS). If ModSecurity is largely used as WAF solution, the CRS is even more adopted, being considered the de-facto standardized set of rules in the WAFs domain. Many commercial and open-source WAF solutions make use of CRS to analyze the web traffic, including Google Cloud Armor and Microsoft Azure WAF [33]. CRS includes 249 rules, out of which 120 target critical injection attacks. SQLi is the most represented class of injection attack, with 49 rules within the CRS, while cross-site scripting (XSS) is the second one with 30 rules. All rules in the CRS are given a heuristic *severity level* and they not only capture many different attack signatures attempted in the past, but they also have been crafted by experts with the goal of detecting variants of the same attack. ModSecurity and all the other WAF solutions based on CRS evaluate whether an HTTP request is malicious or not by computing an overall score over all the rules that are activated by that request. We refer the reader to Sect. 2 for a more detailed overview of SQLi attacks, ModSecurity and the CRS project, and state-of-the-art fuzzing techniques for generating adversarial SQLi attack.

In this work, we show that the aforementioned strategy used by ModSecurity to combine the outputs of the CRS rules can be largely ineffective, for two main reasons. First, we show that ModSecurity achieves a sub-optimal trade-off between detection rate and false alarms, as its weights are selected using heuristics that only consider

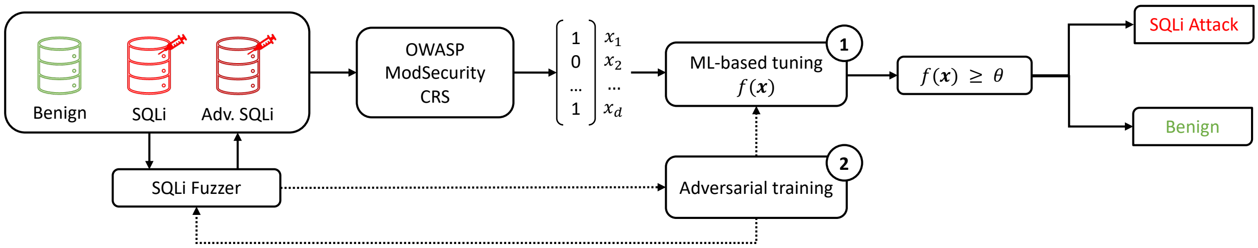

Figure 1: The main idea behind AdvModSec is twofold. (1) We start by training a machine-learning model on the CRS rules to improve the trade-off between detection rate and false alarms. This amounts to learning a model of the incoming traffic directed towards the protected web services. (2) We then improve the robustness of our model against evasive attempts using an ad-hoc adversarial training procedure, exploiting a state-of-the-art SQLi Fuzzer to generate adversarial SQLi attacks during training.

attack patterns, neglecting the nature of the incoming, legitimate traffic directed towards the protected services. Second, we demonstrate that ModSecurity is significantly vulnerable to *adversarial* SQLi attacks that consist of slightly manipulating the malicious SQLi samples to evade detection, without breaking their intrusive functionality. To this end, we optimize adversarially-manipulated SQLi attacks against ModSecurity using a well-known black-box fuzzer, named WAF-A-MoLE [12], and experiment on a publicly-available dataset [12] consisting of more than 700,000 legitimate and malicious samples. We refer the reader to Sect. 4.2 for the results of this empirical evaluation.

To overcome the aforementioned limitations, we propose a twofold approach, presented in Figure 1 and detailed in Sect. 3. The main underlying idea of our approach is to leverage machine learning (ML) to improve ModSecurity. In particular, our first contribution here amounts to training machine-learning models using the CRS rules as input features, which enables learning a model that inherently depends on the traffic directed towards the protected web services. In this way, we show that it is possible to improve the trade-off between detection and false positive rates, improving the overall performance of the ModSecurity WAF. Our second contribution aims to improve robustness against adversarial SQLi payloads. To this end, we define a custom adversarial training approach that incorporates knowledge of state-of-the-art SQLi manipulations when training our learning-based ModSecurity WAF. We will show that the resulting model, named AdvModSec, provides an unprecedented level of robustness against adversarial SQLi attacks. By inspecting the trained AdvModSec model and its predictions, we further provide a thorough understanding of why AdvModSec exhibits such a high level of adversarial robustness.

We validate our approach through an extensive experimental analysis (Sect. 4), showing that AdvModSec does not only increase the detection rate against SQLi attacks by 21% with respect to ModSecurity, but that it also improves its adversarial robustness up to 42%. Furthermore, we also show that AdvModSec is significantly (25%) more robust than other machine-learning models built on the CRS rules without adversarial training.

To summarize, we provide the following main contributions:

(1) we show that the ModSecurity WAF can be largely ineffective as (i) it does not build any model of the traffic observed by the protected web services, causing many false alarms, and (ii) it can be easily bypassed by *adversarial* SQLi attacks;
(2) we overcome these limitations by proposing AdvModSec, an ML-based approach that improves (i) the trade-off between detection and false positive rates, and (ii) robustness to adversarial SQLi attacks via adversarial training.

We conclude the paper by discussing related work (Sect. 5), and the limitations of our approach, along with promising future research directions (Sect. 6). We firmly believe that the results reported in this paper will pave the way towards inspiring novel research directions aimed at strengthening classical rule-based solutions with machine learning-based approaches, filling the gap between these two worlds.

2 BACKGROUND

We provide here an overview of SQLi attacks and introduce the OWASP Core Rule Set (CRS) project along with the SQLi rules included in the CRS to detect them. We then describe how to generate adversarial SQLi attacks to bypass WAFs by fuzzing them.

2.1 SQL Injection

SQLi is one of the most harmful web attack techniques, in which attackers can modify the original behavior of the application to retrieve sensitive information from the database, modify data without authorization, or even execute privileged operations on the database [4]. This can be achieved via specific SQL code fragments that are passed in the original request. If the application does not sanitize the user-provided input, and it simply concatenates it with the rest of the query, the SQL fragment is interpreted as if it were part of the original SQL command.

The login form of a web application is a paradigmatic example. The user's credentials are provided in two input fields (e.g., `$user` and `$passwd`) and sent along with the HTTP request. The credentials could then be checked server-side via a database query, as shown in Listing 1.

```
SELECT * FROM users WHERE username = '$user'
  AND password = '$passwd'
```

Listing 1: Example of SQL query vulnerable to SQL injection.

However, malicious users could inject SQL fragments in the `$user` parameter, *e.g.*, "`admin'-- `". As shown in Listing 2, this comments out the portion of the original SQL query that checks for a valid password, leading to a successful SQLi attack, i.e., the login is performed only using a valid username, without requiring the password.

```
SELECT * FROM users WHERE username = 'admin'
 -- ' AND password = 'x'
```

Listing 2: Example of SQL injection attack targeting the SQL query of Listing 1.

2.2 OWASP CRS Project

The Open Web Application Security Project (OWASP) Core Rule Set (CRS) project [31] is one of the most popular open-source sets of web attack detection rules targeting OWASP Top 10 security risks [30]. It is the reference rule set of several open-source WAF engines like ModSecurity [15] and Coraza[1]. It is also adopted in more than ten commercial solutions such as Google Cloud Armor, Microsoft Azure, Amazon Web Services (AWS) and Cloudflare WAFs [33].

Detection Rules. Rules included in CRS are implemented via regular expressions (regex). In the example of Listing 3, the regex (i.e., the string following `@rx` in line #1) captures several patterns of comments commonly used in SQLi attacks such as "`;--`", "`-- `", etc. Moreover, each rule is identified through a unique identifier (*id* in line #2) that, in the considered example, is *942440.* It also indicates the attack type detected by the rule. For instance, all the rules starting with *942* target SQLi attacks [35]. Finally, among all the other attributes shown in Listing 3, it is worth mentioning the Paranoia Level (line #7) and severity level (line #9) that is linked to the anomaly scoring mechanism adopted by CRS. Both are described in the following.

Paranoia Level. The Paranoia Level (PL) [34] is a configuration parameter used to select which rules are enabled to analyze the HTTP requests. CRS includes four PLs (PL1 - PL4) and each rule is assigned to a specific PL. For instance, rule in Listing 3 belongs to PL2 (line #7). Specifically, rules are grouped together by PL in a nested way: when setting a certain PL, it enables all the rules assigned to this PL as well as those assigned to lower PLs. For instance, PL3 enables all the rules related to such PL, as well as those assigned to PL1 and PL2. Consequently, setting PL4 will enable all the rules.

Not surprisingly, SQLi is the vulnerability class that received more attention from CRS and counts 49 rules (over a total of 249) grouped by PL as follows (see outer ring of Figure 2): 16 (33%) related to PL1, 24 (49%) belonging to PL2, while 7 (14%) and 2 (4%) related to PL3 and PL4, respectively.

[1] https://coraza.io

Anomaly Scoring. Each detection rule is heuristically assigned with a *severity level*, a positive integer value that quantifies how menacing a captured request is [32]. To compute a decision, ModSecurity applies the rules on incoming requests, and it sums all the severity levels of all the matches. If such a summation exceeds a threshold, the incoming request is flagged as malicious. In CRS there are four severity levels: CRITICAL (5), ERROR (4), WARNING (3) and NOTICE (2). For instance, the severity level of the rule in Listing 3 is CRITICAL (line #9), which means that its contribution to the anomaly score is equal to 5 (line #10).

As for the distribution of SQLi rules' weights (see Figure 2), 42 (86%) rules have weight set to 5 (CRITICAL), while the 7 (14%) remaining ones are weighted 3 (WARNING).

Limitations. Even if the severity level of the detection rules is assigned by domain experts, such process remains purely heuristic. We will show in Sect. 4 that this strategy tends indeed to produce a largely sub-optimal trade-off between the detection rate and the false alarm rate exhibited by the ModSecurity WAF.

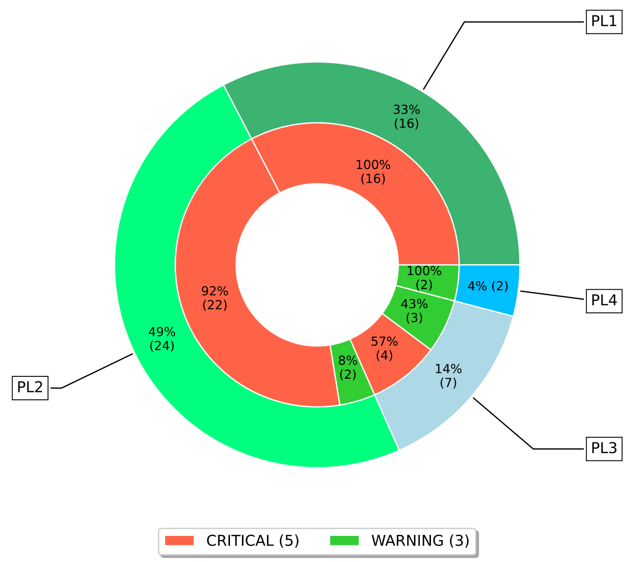

Figure 2: Analysis of the distribution of the SQLi rules by PL and severity level (i.e., weight).

2.3 Adversarial SQLi Attacks against WAFs

In the context of WAFs, the problem of finding SQLi attacks that are able to bypass the target WAF is *adversarial in nature.* Therefore, it is reasonable to expect that attackers may manipulate the source SQLi attack sample to evade detection, while preserving its malicious functionality [12, 20]. For instance, the SQLi rule (*942440*) of CRS reported in Listing 3 can detect the SQLi attack shown in Listing 2. However, by just carefully inserting a white space character ('`  `') in the original attack (cf. line #1 of Listing 4), we can generate a new semantically-equivalent SQLi attack (cf. line #2 of Listing 4) that can evade the 942440 rule.

WAF-a-MoLE. In this work, we use the state-of-the-art, open-source SQLi fuzzer known as WAF-A-MoLE [12]. It encompasses

```
SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|!REQUEST_COOKIES:/_pk_ref/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "@rx
      (?:/\*!?|\*/|[';]--|--[\s\r\n\v\f]|--[^-]*?-|[^&-]#.*?[\s\r\n\v\f]|;?\\x00)"
    "id:942440,
    block,
    msg:'SQL Comment Sequence Detected',
    logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %{MATCHED_VAR}',
    tag:'attack-sqli ',
    tag:'paranoia-level/2',
    ver:'OWASP_CRS/3.3.4',
    severity:'CRITICAL',
    setvar:'tx.anomaly_score_pl2=+%{tx.critical_anomaly_score}',
    setvar:'tx.sql_injection_score=+%{tx.critical_anomaly_score}'"
```

Listing 3: Excerpt of a SQLi detection rule (942440) of CRS. It detects patterns of comments commonly used in SQLi attacks.

```
1 admin' OR 1=1;--'
2 admin' OR 1=1;  --'
```

Listing 4: Example of SQLi payload (line #1) that is detected by 942440 SQLi rule (see Listing 3) and adversarially-manipulated one (line #2) to evade such rule.

many of the known manipulation techniques used to craft adversarial SQLi attacks against ML-based WAFs. In particular, it adopts a guided mutational fuzz testing algorithm [45] that works as follows. Starting from a given SQLi attack that is correctly classified as malicious by the target WAF, it randomly applies a set of semantic-preserving mutation operators in order to generate new mutated SQLi attacks, which are evaluated by the WAF and ranked using the confidence score returned by the WAF. The best among them is used for generating newly-mutated payloads during the next round. The above steps are repeated until the confidence score of the best payload found so far is lower than a given threshold or when a stop condition is met (i.e., the maximum number of rounds or queries to the target WAF is reached).

In this work, we will use WAF-A-MoLE not only to show that ModSecurity can be bypassed by adversarial SQLi attacks, but also to show how adversarial robustness of ML-based WAFs can be significantly improved.

3 ADVERSARIAL MODSECURITY

This section describes in detail the main contributions of our work. Specifically, we present AdvModSec, our novel methodology (see Figure 1) for training a robust ML classifier on the CRS rules. AdvModSec executes two consecutive phases. In the former, we train machine-learning models on a feature space defined by the CRS rules of the vanilla ModSecurity. For this reason, from now on we refer to such models as MLModSec.

In the latter, our methodology implements a novel adversarial training approach that leverages a state-of-the-art SQLi fuzzer in order to improve the robustness of MLModSec against adversarial SQLi attacks. Such adversarially-trained models are the final outcome of our methodology.

3.1 MLModSec: ML for ModSecurity

To surpass the shortcoming of manually-tuned severity levels, while keeping the predictive power of the CRS rules, we construct a feature space based on these rules, and we train machine-learning models on it. Hence, MLModSec comprises two components: (i) a feature extraction phase that encodes the CRS rules triggered by the input samples (i.e., SQL queries) into a vector representation, and (ii) a machine-learning model that learns how to combine the CRS rules to minimize the classification error.

Detection Rules as Features. The input space is represented by SQL queries that need to be classified as malicious or benign by a machine-learning classifier. Each SQL query is a string of readable characters, represented as $z \in \mathcal{Z}$, being $\mathcal{Z}$ the space of all possible queries. Let $\mathcal{D}$ be the set of selected SQLi rules from CRS, and $\mathrm{d} = |\mathcal{D}|$ its cardinality. We denote with $\phi : \mathcal{Z} \mapsto \mathcal{X}$ a function that maps a SQL query z to a d-dimensional feature vector $\boldsymbol{x} = (x_1, \dots, x_{\mathrm{d}}) \in \mathcal{X} = \{0,1\}^{\mathrm{d}}$, where each feature is set to 1 if the corresponding SQLi rule has been triggered by the SQL query z, and 0 otherwise.

$$\boldsymbol{x} = \phi(z) \mapsto \begin{pmatrix} \cdots \\ 0 \\ 1 \\ \cdots \\ 1 \\ 0 \\ \cdots \end{pmatrix} \begin{array}{l} \cdots \\ \texttt{942160} \\ \texttt{942170} \\ \cdots \\ \texttt{942420} \\ \texttt{942431} \\ \cdots \end{array}$$

Figure 3: Feature encoding approach adopted in our work to represent a SQL query z into a vector $\boldsymbol{x}$.

A SQL query encoded in feature space may thus look like Figure 3, where SQLi rules are represented by their ID like `942160`. We want to remark that, even if in this paper we focus on SQL queries, this feature representation can be built on top of every family of rules contained inside the CRS.

ModSecurity as a Linear Classifier. We show here that ModSecurity indeed works as a linear classifier, with fixed hard-coded parameters. Let $\boldsymbol{w}$ be the vector containing the severity levels for each detection rule, and b the value of the threshold (with negative sign) used to determine the maliciousness of input samples. Hence, given an input SQL query z, ModSecurity compute the following score: $f : \mathcal{Z} \to [0,1], f(\boldsymbol{x}) = \boldsymbol{w}^t \boldsymbol{x} + b > 0$, where $\boldsymbol{x} = \phi(z)$ is the feature-vector representation of the input as described in Figure 3. The main limitation of this approach stands in the choice of $\boldsymbol{w}$, whose entries are fixed and they do not capture the real distribution of data. Hence, we now show how these parameters can be learned by leveraging machine-learning models in order to improve performance.

Optimal Combination of CRS Rules with ML. To optimally tune the contribution of the SQLi rules towards effectively classifying the input requests we leverage two different machine-learning algorithms on the feature representation defined above: a linear Support Vector Machine (SVM), and a non-linear Random Forest (RF) classifier. Nevertheless, we remark that our methodology can be applied to any machine-learning classifier.

Linear SVM. The SVM classifier [10, 43] is a linear function trained to maximize the distance (i.e. margin) between the different classes of the input training data. It can be written as $f : \mathcal{X} \mapsto \mathbb{R}$, $f(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x}+b$, where, in our case, $\boldsymbol{x} = \phi(z)$ is the SQL query in the feature space, while $\boldsymbol{w} \in \mathbb{R}^{\mathrm{d}}$ denotes the vector of weights that in our case corresponds to the *SQLi rules' weights*, and b is the bias term. Unlike ModSecurity, the SVM learns its weights at training time, adapting them to the real traffic of the web application to defend. Ideally, such optimized weights can be directly used within the ModSecurity equipped with CRS, without any computational overhead at test time, providing a better, well-crafted model to protect our target web services (i.e., being trained specifically to classify correctly the incoming traffic directed towards them).

Random Forest. If, on the one hand, linear models have several practical advantages such as explainability, simplicity and computational efficiency, on the other hand, they may exhibit lower classification accuracy on data that is not inherently linearly separable, or when features are not independent of each other [28]. For such reasons, we also consider here a non-linear model, i.e., the RF algorithm, which is also well-renowned for its computational efficiency both at training and at test time. In particular, RF is a tree-based ensemble learning algorithm, where each tree is trained on random features in order to increase the generalization power and reduce the variance in the individual trees' output [8, 11]. The final prediction is calculated by a majority prediction vote of the decision trees or by averaging the predictions of all the trees. It is characterized by several advantages, including robustness to outliers, scalability and efficiency on high-dimensional input spaces since they can be implemented in parallel.

3.2 AdvModSec: adding Adversarial Training

As machine-learning models exhibit vulnerabilities to minimal input changes, *adversarial training* stands out to be capable of withstanding evasion attacks [7, 26]. This technique integrates the computation of adversarial examples at training time, thus giving the model the possibility of knowing in advance evasive patterns that could be computed at test time. However, adversarial training mostly works in domains like object detection and recognition, where images can be manipulated with additive perturbations. Hence, we are the first to bridge the gap, by proposing an adversarial training strategy that leverages state-of-the-art manipulations of SQL queries through WAF-A-MoLE. Formally, our methodology consists in optimizing the following min-max objective:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\delta}} \sum_{z,y \in \mathcal{S}} L(f(\phi(h(z;\boldsymbol{\delta}))), y; \boldsymbol{\theta}), \tag{1}$$

where $\mathcal{S}$ is a dataset including the pairs (z, y), z is an input SQL query and y is its true label. The function $h : \mathcal{Z} \times \mathcal{M} \rightarrow \mathcal{Z}$ (being $\mathcal{M}$ the set of operators included in WAF-A-MoLE) transforms an input SQLi payload z into a new semantically-equivalent SQLi query through a sequence of functionality-preserving mutation functions specified in $\boldsymbol{\delta}$ used in WAF-A-MoLE. The loss function $L : \mathbb{R}\times\mathcal{Y} \rightarrow \mathbb{R}$, parameterized with $\boldsymbol{\theta}$ measures how likely an input sample is classified as SQLi payload, by comparing the output of the model $f(\phi(h(z;\delta)))$ to the real class label $y = 1$ of malicious samples. Lastly, $\phi : \mathcal{Z} \rightarrow \mathcal{X}$ is the feature extraction function, that maps input SQL queries to the CRS rules they trigger as detailed in subsection 3.1.

Algorithm 1: Adversarial training of AdvModSec with WAF-A-MoLE

Input : $\mathcal{T} = (\mathcal{Z}, \mathcal{Y})$, training set with SQL queries;
f, the target ML model to train;
N, number of adversarial examples to generate

Output : $f^\star$, the re-trained model

1 $\mathcal{Z}' \leftarrow$ *random_sampling_attacks*$(\mathcal{Z}, N)$
2 **for** z **in** $\mathcal{Z}'$
3 $\quad z^\star \leftarrow$ *WAF-A-MoLE*(z, f)
4 $\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \{z^\star\}$; $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{1\}$
5 $f^\star \leftarrow$ *train*$(f, \mathcal{Z}, \mathcal{Y})$
6 return $f^\star$

In the classical definition, adversarial training computes evasive samples through gradient-based techniques, as both the manipulations and the model to harden are differentiable. The methodology we propose differs from such formulation since we deal with non-differentiable objects. Hence, we rely on a gradient-free, a.k.a. *black-box*, approach, and we detail how to apply our adversarial training procedure in algorithm 1. Given a dataset $\mathcal{Z}$ of benign queries and SQLi attacks, we first create a new set $(\mathcal{Z}')$ by randomly sampling a given amount of SQLi samples from the training dataset (line 4). Then, for each payload of this newly created set, we use WAF-A-MoLE (subsection 2.3) to generate the corresponding adversarial example (line 3) and add it to the training dataset along with its label (line 4). Finally, the model is re-trained on the new training set that includes the adversarial SQLi payloads (line 5).

We would like to point out that although adversarial training and WAF-A-MoLE are not novel on their own, none has ever proposed to use them jointly and we are also the first to apply them in the web security domain and to show that adversarial SQLi examples generated in the *problem space* [37] can be effectively employed to successfully harden ML-based WAFs. Moreover, our approach is completely different from the case of image classifiers, where adversarial training is not sufficient to achieve high levels of robustness, and adversarial examples are still an open issue.

4 EXPERIMENTAL ANALYSIS

In this section, we detail the three experiments that we run to support the contributions of our methodology. Specifically, we first evaluate the detection capabilities of the vanilla ModSecurity (see subsection 4.2) showing that, due to its naive approach of combining the CRS rules based on manually-assigned weights, which does not take into account the benign traffic, it achieves a very low detection rate (~70% at 1% FPR in the best case) and it is significantly vulnerable to adversarial SQLi attacks (~55 % detection rate at 1% FPR on the best configuration). Second, we empirically show that the ML-based tuning adopted by MLModSec allows to fill the gaps of the vanilla ModSecurity by significantly enhancing its detection rate up to 21% (see subsection 4.3). Third, we present the results of our novel adversarial training approach of AdvModSec demonstrating that it significantly improves the robustness of the vanilla ModSecurity by 42%, while it is 25% more robust than its non-hardened counterpart (see subsection 4.4). The overall experimental setup is discussed in subsection 4.1.

4.1 Experimental Setup

We now describe the setup underlying our experimental analysis, conducted on an Ubuntu 18.04.6 LTS server equipped with an Intel Xeon E7-8880 CPU (16 cores) and 64 GB of RAM.

Dataset. We adopt the same dataset[2] used by Demetrio *et al.* [12], consisting of 393,629 malicious and 345,199 benign SQL queries. As for the dataset collection, it is worth noting that although it does not reflect a real-world scenario, it is, according to our knowledge, the most recent and comprehensive dataset for training machine-learning models for SQLi detection. Specifically, the benign samples have been generated from a restricted SQL grammar, while malicious ones are generated using state-of-the-art web security testing tools such as sqlmap and OWASP ZAP [12]. We use it to build the following sub-datasets for training and evaluating the performance of ModSecurity and our models.

Training set (`train`). This dataset is composed of 20,000 samples by random sampling them from the original dataset, keeping the balance between the two classes. Thus, it contains 10,000 benign and 10,000 SQLi queries.

Test set (`test`). This dataset contains 4,000 samples chosen randomly from the original dataset, where 2,000 are benign queries, while the other 2,000 are SQLi queries. This dataset has no intersection with the training set mentioned before, and we use it to evaluate the performances of our target WAFs, i.e. vanilla ModSecurity, MLModSec and AdvModSec, at different PLs.

Adversarial training set. This dataset is built by randomly sampling 5,000 SQLi queries from the main training set (i.e., `train`), optimizing them using WAF-a-MoLE against a target WAF, and adding the manipulated SQLi queries to the main training set for adversarial training.

Adversarial test set (`test-adv`). This dataset is obtained by optimizing the 2,000 SQLi queries from the test set (i.e., `test`) with WAF-a-MoLE against each target WAF, while keeping the benign samples unchanged. This dataset is used to evaluate the adversarial robustness of the target WAFs. Finally, we would like to clarify that, although using the same SQLi fuzzer (i.e., WAF-A-MoLE), the adversarial examples generated for building the adversarial training and test sets are different. They are also re-optimized against each target model at test time, resulting in the application of different, optimal manipulation strategies. Hence, the two sets are independent, ensuring an unbiased evaluation. Using different SQLi fuzzers may result in overestimated robustness values, given that they do not optimize the manipulations as WAF-A-MoLE does, and many of them only include a subset of the same adversarial manipulations used by WAF-A-MoLE.

[2] https://github.com/zangobot/wafamole_dataset

Setup of ModSecurity. We implement the feature extractor described in subsection 3.1 with the *pymodsecurity*[3] v0.0.5 library, that implements the Python bindings to interface with ModSecurity v3.0.3. Since in this work we focus on detection of SQLi attacks, we only enabled the SQLi rules[4] of CRS.

Setup of WAF-A-MoLE. We configure WAF-A-MoLE to minimize the confidence score of the target model, by setting its *threshold* parameter to zero, and to use a maximum of 2000 queries, which in our experiments resulted to be sufficient to reach convergence when optimizing the attacks. To efficiently query and test ModSecurity, i.e., without running the attacks on an instance of the web server, we extended WAF-A-MoLE by developing a dedicated interface that uses the pymodsecurity Application Programming Interface (API).

MLModSec and AdvModSec with SVM and RF. We leverage the scikit-learn v1.1.3 [36] implementation of SVM and RF to train both MLModSec and AdvModSec, and we tune their hyperparameters using grid search [40] based on 5-fold cross-validation on the training set, i.e., `train`. Hence, the training set is split into 5 folds, and each model is trained and evaluated five times, each time using a different fold as the validation set. The final model is selected based on the average F1-score (defined as the harmonic mean of precision and recall [40]) across these five evaluations. Specifically, as for MLModSec SVM, we tune the regularization parameter C, by considering 9 values: $\{10^{-4}, 5\cdot10^{-4}, 10^{-3}, 5\cdot10^{-3}, 10^{-2}, 5\cdot10^{-2}, 10^{-1}, 5\cdot10^{-1}, 1.0\}$, while using the default values of the remaining parameters. We train one SVM for each PL, and after the cross-validation, the best parameters are $C = 10^{-4}$ as for PL1, while $C = 1.0$ as for the other PLs. Instead, as for MLModSec RF, we tune the number of decision trees used by the algorithm, by looking into the following values: {10, 40, 80, 160}, while using the default values of the remaining parameters. We train one RF for each PL, and after the cross-validation, the optimal number of estimators is 10 for PL1, while 160 for the other PLs. Regarding AdvModSec, we apply our adversarial training procedure only for PL4 considering both SVM and RF. The rationale behind this choice will be detailed in subsection 4.3, where we demonstrate that MLModSec models trained on PL4 have better performances on the test set. Finally, we use the same settings described above for estimating the best hyperparameters of AdvModSec models.

4.2 Evaluation of ModSecurity

The first goal of our experimental analysis is understanding the predictive capability of the vanilla ModSecurity based on CRS. Rather than focusing only on its default values, we experiment with it over its entire configuration space, considering both the 4 different PLs and the classification threshold that can be set in ModSecurity. Hence, for each PL, we compute the Receiver-Operating-Characteristic (ROC) curve, which reports the True Positive Rate (TPR, i.e., the fraction of correctly-detected malicious SQLi requests) against the False Positive Rate (FPR, i.e., the fraction of wrongly-classified legitimate requests) obtained by considering all possible classification threshold values.

[3] https://github.com/pymodsecurity/pymodsecurity

[4] https://github.com/coreruleset/coreruleset/blob/v3.3/master/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf

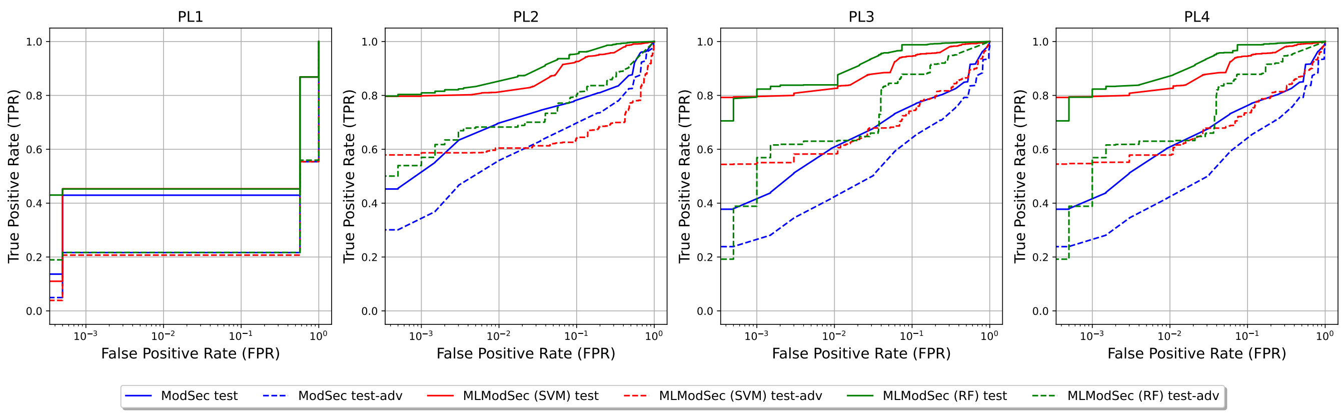

Figure 4: ROC curves of vanilla ModSecurity (ModSec) and MLModSec (SVM and RF), evaluated on `test` and `test-adv`. Each curve reports the average detection rate of SQLi attacks (i.e., the True Positive Rate) against the fraction of misclassified benign SQL queries (i.e., the False Positive Rate).

We report our findings with blue lines in Figure 4, while in Table 1 we extrapolate the TPR values at 1% FPR. We have chosen to report the results in Table 1 at 1% FPR because it is a reasonable value commonly adopted in the literature [9, 13]. Nevertheless, the ROC curves in Figure 4 already show the detection rates for each possible operating point (i.e., value of FPR). So, our experiments already report the results for any possible FPR value. We detail hereafter the key findings of our evaluations of ModSecurity against both the test set (`test`) and the adversarial test set (`test-adv`).

ModSecurity against the test set. The results of this first evaluation are indicated with the blue solid lines of Figure 4. The ROC curve of PL1 (default PL for ModSecurity) highlights its complete inability in discriminating benign and malicious SQL queries (TPR below 50% for all FPR below 50%). PL2 appears to be the best among all the PLs, with a detection rate of 69.55% at 1% FPR. Lastly, the ROC curves for both PL3 and PL4 are almost identical. This simply indicates that the two extra SQLi rules of PL4 do not contribute to improving the detection capabilities.

ModSecurity against the adversarial test set. The results on the adversarial test set highlight a more alarming trend, thus clearly showing that ModSecurity is not able to cope with adversarial attacks. As shown by the blue dashed lines in Figure 4, regardless the considered PL, the TPR drops to 55.55% when considering 1% FPR (which is slightly better than random guessing).

While these findings are aligned with the recommendations of the OWASP Foundation (stating that higher PLs usually lead to more false positives [34]), ModSecurity with CRS is unable to really defend applications and networks not only against already-known attacks, but also against adversarial examples.

4.3 Evaluation of MLModSec

We now analyze the detection capabilities of MLModSec using both SVM and RF trained on the feature set defined with the CRS rules as described in Figure 3, by testing them against both the baseline test set (`test`) and the adversarial one (`test-adv`).

MLModSec against the test set. We plot the ROC curves in Figure 4 using red and green solid lines. By comparing the results of these models with the respective ModSecurity counterpart, we can conclude that the application of machine learning improves the TPR for all PLs greater than 1. Specifically, considering the best results reported in Table 1, the TPR at 1% FPR of the linear SVM is 18.65% higher than ModSecurity. As for the RF, the TPR at 1% FPR is 25.8% higher than the vanilla ModSecurity. It is worth noting that, for PL1, the trained ML models achieve similar results to the vanilla ModSecurity. This confirms that, even by learning optimal weights, rules enabled by PL1 are inappropriate to effectively discriminate benign samples from malicious ones. Finally, we would like to remark that, unlike the vanilla ModSecurity, all ML models achieve the best detection rate with PL4 (even though they are only slightly higher than those obtained for PL2). This result underlines that, even when adding rules that may lead to more false positives, ML models are able to tune the importance of each rule, thus achieving a better trade-off between TPR and FPR.

MLModSec against the adversarial test set. As regards the robustness evaluation, both ML models suffer the presence of adversarial attacks, as shown in Figure 4 (dashed red and green lines) but they still outperform the detection capabilities of ModSecurity. Specifically, considering the best results on the adversarial test set reported in Table 1, the TPR at 1% FPR of the MLModSec linear SVM / RF is 8.8% / 22.8% better than ModSecurity. Contrary to the setup in absence of attacks, the best PL for this analysis is PL2. This effect underlines that the adversarial SQL injections are able to leverage rules of PL3 and PL4 to achieve evasion, by removing detected patterns considered important at training time.

4.4 Evaluation of AdvModSec

As we notice that models trained on PL4 achieve better performances in terms of TPR/FPR, we use this setting to re-train both the linear SVM and RF versions of MLModSec. We then evaluate the re-trained models of AdvModSec against the `test` and `test-adv`, and plot the results in Figure 5 using light blue lines. Also, we report the TPR of AdvModSec at 1% FPR as the last column of Table 1.

		PL1	PL2	PL3	PL4	
					Base	AdvModSec
ModSec vanilla	test	42.95	**69.55**	60.25	60.25	—
	test-adv	21.65	**55.55**	41.70	41.70	—
MLModSec SVM	test	42.25	81.21	82.52	**82.52**	82.02
	test-adv	20.70	60.45	58.20	57.85	**70.40**
MLModSec RF	test	45.35	84.76	83.90	**87.55**	84.45
	test-adv	21.70	68.25	63.00	63.00	**78.95**

Table 1: TPR at 1% FPR for ModSec and MLModSec (SVM and RF) evaluated on the baseline (`test`) and adversarial (`test-adv`) test sets. As for the ML models, it also reports the results for AdvModSec at PL4. The best results for each target WAF are in bold. RF achieves the best results by PL on both `test` and `test-adv`.

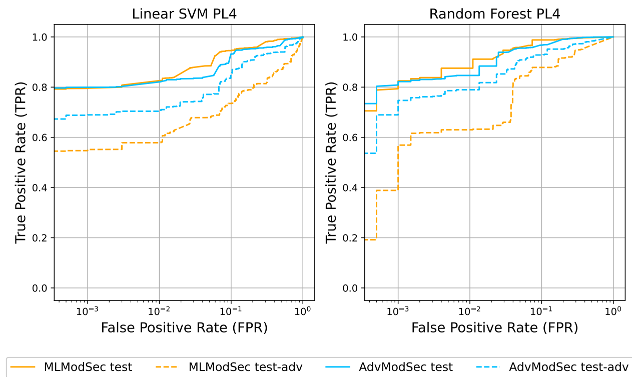

Figure 5: ROC curves for SVM and RF models of MLModSec (yellow) and AdvModSec (light blue) at PL4, evaluated on `test` (solid lines) and `test-adv` (dashed lines).

Hereafter, we first discuss the performance of AdvModSec in comparison with MLModSec (baseline ML models without adversarial training). Overall, we observe that the robustness achieved by AdvModSec clearly outperforms its non-hardened counterparts. Then, we inspect deeply such a result and we provide a detailed explanation.

AdvModSec against the test set. In absence of attacks, AdvModSec has comparable performances to its non-adversarially-trained models MLModSec, exhibiting a small decrement in TPR (solid lines in Figure 5). This is expected, as the adversarial training procedure is making the problem harder to train, hence the model pays some accuracy points to boost robustness.

AdvModSec against the adversarial test set. Over the adversarial test set, AdvModSec outperforms its non-hardened counterparts, reaching thus higher robustness (see the dashed lines of Figure 5). In percentage, with a fixed threshold at 1% FPR, the linear AdvModSec SVM improves the performance of its non-hardened counterpart by 21.7%, and AdvModSec RF by 25.3%. Furthermore, it is worth noting that, the best AdvModSec (i.e., RF at PL4) is 42% more robust than the best vanilla ModSecurity (PL2). Of course AdvModSec is still vulnerable to new adversarial examples optimized against it, but the decrement in performance is lower compared to the decrement caused to the non-hardened models.

Robustness of AdvModSec explained. Here we explain why AdvModSec achieves better robustness. To do so we consider the behavior of the linear SVM model, before and after adversarial training. Indeed, for a linear model, we can precisely compute how much each feature contributed to the score by simply inspecting the weights. As we want to understand if the model is relying on rules that WAF-A-MoLE is avoiding or triggering, we first evaluate which features are changed by the adversarial attack, by computing the *rule activation delta*, i.e., $\Delta \boldsymbol{a}_i = \boldsymbol{a}_i - \boldsymbol{a}'_i$. This captures the difference between the probability that the rule i is activated by standard attacks and the probability that is activated by its adversarial counterpart. Hence, if this quantity is positive / negative, it means that WAF-A-MoLE is bypassing / activating the signature matched by rule i, and if it is zero it means that WAF-A-MoLE has no effect on rule i. In theory, a robust classifier would attribute positive importance both to rules that are activated as side effects of attacks, thus making the attack detectable, and to those that attacks are not able to bypass, thus increasing the difficulty of achieving evasion.

With this scenario in mind, we analyze how each rule is affected by the adversarial SQLi attacks generated through WAF-A-MoLE (Figure 6), as well as how differently the baseline MLModSec SVM and the AdvModSec SVM compute weights for each rule (Figure 7). Specifically, in Figure 6 we plot the probability that a rule is active, and we display how different the distribution for the malicious (cyan) and adversarially-manipulated (yellow) queries are. When the two distribution overlap, they are colored turquoise. Furthermore, we sort the rules by their value of $\Delta \boldsymbol{a}_i$, and we divide them into three groups: rules whose matching signature is evaded by WAF-A-MoLE (left), rules that WAF-A-MoLE is unable to affect (center), and rules that are triggered only by adversarial attacks as a side effect (right). The same approach is also adopted to sort the weights learned by MLModSec and AdvModSec SVM, reported in Figure 7.

We can already observe that more than one-third of the rules are exploited by WAF-A-MoLE to avoid detection, as the first block has a drop in the probability of being active. This is also confirmed by Figure 7, where we can see that most of the positive weights (i.e., the one that increases the scores towards the malicious class) assigned by MLModSec (cyan) are all concentrated in the first group, which is exactly the one leveraged by adversarial attacks. On the contrary, AdvModSec (yellow) is more robust since it spreads the importance on more rules, prioritizing the ones belonging to the second and third groups, making attacks harder to land and more easy to detect. Of course, in this analysis we focus on a linear model, which we have shown in subsection 4.4 to still be weak against adversarial attacks. Also, by looking at Figure 7, AdvModSec attributes negative weights (i.e., decrease the score towards the benign class) to some features of the first block, hence giving space to WAF-A-MoLE in finding adversarial examples. On the other hand, AdvModSec RF is more robust because its non-linearity induces correlation between features, requiring the attacker to increase the number of affected rules, and thus increasing robustness as shown in subsection 4.4.

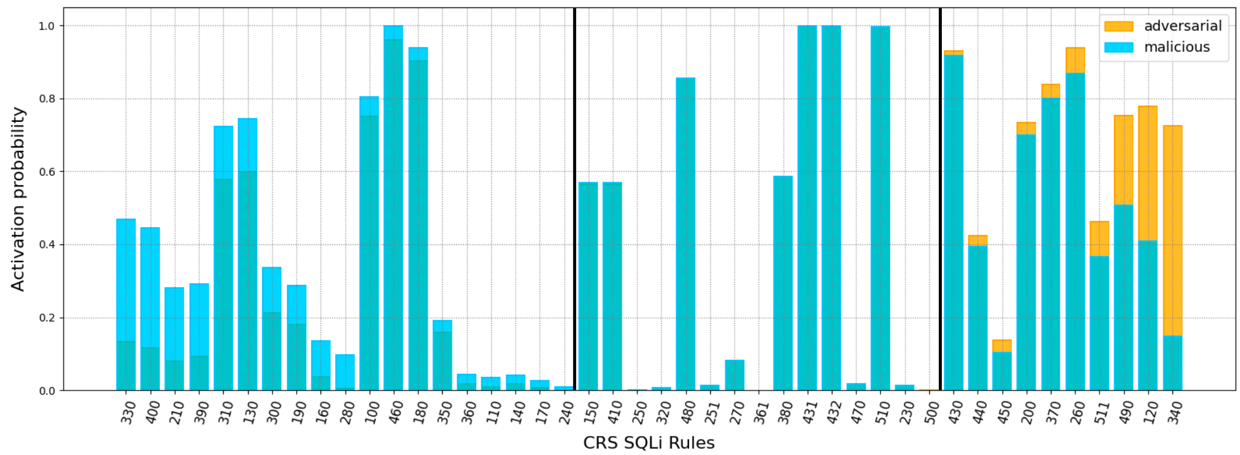

Figure 6: Activation probability of CRS rules on malicious (cyan) and adversarial (yellow) SQLi samples optimized against MLModSec (SVM). The other color in the plot (turquoise), is given by the overlapping between the yellow and cyan bars. For the sake of brevity and clarity, we only report the last three digits of the rule IDs on the x-axis since, for all the rules, the first three digits are equal to *942*.

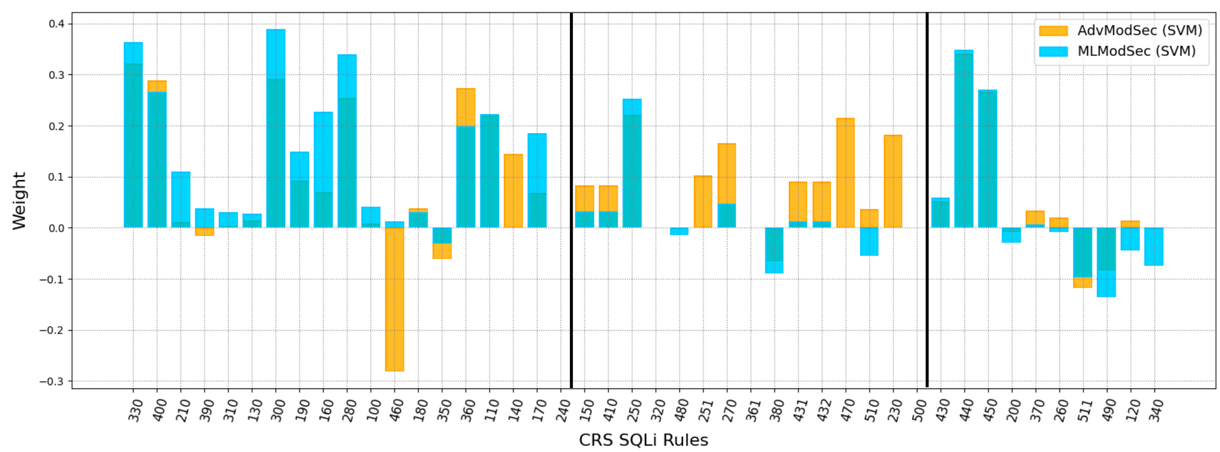

Figure 7: Weight values learned by MLModSec SVM (cyan) and AdvModSec SVM (yellow). The additional color, i.e., turquoise, is given by the overlapping of the yellow bars with the cyan ones. For brevity and clarity's sake we only report the last three digits of the rule IDs on the x-axis since, for all the rules, the first three digits are equal to *942*.

5 RELATED WORK

If ModSecurity (and its CRS) has been targeted by previous studies, its detection capabilities have been only partially evaluated. Specifically, [41, 42] considered the impact of the PL under different types of web security threats. However, unlike our work, both of them adopt a very limited number of attack samples (for instance, the authors of [41] use only 27 samples) and do not provide a complete analysis of the trade-off between TPR (True Positive Rate) and FPR (False Positive Rate). Furthermore, none of them analyzed how the detection capabilities change in an adversarial scenario (i.e., against adversarially-manipulated attacks).

Other papers go beyond that limitation and consider adversarial SQLi attacks against ModSecurity by proposing different approaches based on machine learning [3] and Reinforcement Learning (RL) [18, 20, 44], fuzzing [12, 24], as well as heuristic search algorithm like Monte Carlo Tree search [38]. However, either they do not specify which PLs they targeted [3, 18, 24] for their analysis or they just limit themselves to the default PL1 such as [20, 38]. In both cases, the results are thus just partial and do not explain precisely why ModSecurity is failing and how it could be improved.

Indeed, according to our best knowledge, there is no work that provides a thorough analysis of ModSecurity as we do in this paper (i.e., carefully evaluating the impact of its configuration settings such as the PL). In addition, no previous work has investigated the potentiality of adversarial training in this specific application domain making us the first that propose an ML-based methodology for effectively increasing the robustness of WAFs.

6 CONCLUSIONS AND FUTURE WORK

In this work we proposed AdvModSec, a novel methodology for training ML classifiers using the CRS rules as input features to learn how to optimally tune the severity levels (i.e., the weights) of CRS rules on the traffic directed towards the protected web services, hence achieving the best trade-off between detection rate and false positive rate. Additionally, it also leverages a novel adversarial training methodology that incorporates knowledge of state-of-the-art SQLi fuzzer and manipulations to counter the presence of adversarial SQLi attacks. Among the major findings, we empirically show that AdvModSec improves the detection rate of the vanilla ModSecurity by 21%, and its adversarial robustness up to 42%. Hence, we can state that our methodology provides a first, concrete example of how adversarial machine learning can be used to effectively enhance the robustness of WAFs against adversarial attacks.

We envision several promising future developments to further improve our work. First, our methodology has been evaluated on the dataset released by Demetrio et al. [12]. This collection of queries has not been captured from real-world network traffic, but they have been generated using a grammar along with well-known attack payloads from sqlmap [2, 6, 29] and OWASP ZAP [21, 27]. However, as far as we know, is the most recent and comprehensive open-source dataset including both benign and malicious SQLi samples. Indeed, other public datasets available such as CSIC-2010 [17] and ECML/PKDD 2007 [39] are outdated (i.e., do not include newly discovered web attacks) and do not reflect the complexity of real-world web applications [1, 5]. Furthermore, the synthetic dataset used in our work is probably even more challenging than real data since SQLi queries are generated by injecting malicious payloads into legitimate queries, thus making the classification problem even more challenging. While we are confident that such an artificial setting does not impact the results of our methodology, it still remains to be investigated what would happen in a real scenario, with a real network to defend.

Secondly, even though in this work we target only one type of web attack, i.e., SQLi, our methodology is general enough to tackle other web threats like XSS. Hence, as future work, we will start by building both a dataset and new manipulations for other threats, like XSS or remote code execution (RCE) attacks, and train one or more hardened ML model that can spot them in the wild. These can be accomplished by tuning AdvModSec accordingly, by increasing the number of rules that we use inside the training pipeline.

Finally, we also see future developments in evaluating other state-of-the-art ML-based WAFs. Indeed, even though in this work we focus on ML classifiers based on CRS, we think that the same promising results can be also obtained on more advanced models such as Convolutional Neural Networks (CNN) [25], as well as on different feature representation approaches [23]. This is also true for a production-grade WAFs. Indeed, since, as discussed in section 2, there are commercial solutions based on CRS, another interesting future extension of our approach may be to evaluate them in terms of transferability [14] of adversarial SQLi attacks optimized on ModSecurity.

ACKNOWLEDGMENTS

This research was partially supported by the TESTABLE project, funded by the European Union's Horizon 2020 research and innovation program (grant no. 101019206).

REFERENCES

[1] Rokia Lamrani Alaoui and El Habib Nfaoui. 2022. Deep learning for vulnerability and attack detection on web applications: A systematic literature review. *Future Internet* 14, 4 (2022), 118.

[2] Waad Almadhy, Amal Alruwaili, and Saloua Hendaoui. 2022. Using SQLMAP to Detect SQLI Vulnerabilities. *IJCSNS* 22, 1 (2022), 234.

[3] Dennis Appelt, Cu D. Nguyen, Annibale Panichella, and Lionel C. Briand. 2018. A Machine-Learning-Driven Evolutionary Approach for Testing Web Application Firewalls. *IEEE Transactions on Reliability* 67, 3 (2018), 733–757.

[4] Dennis Appelt, Annibale Panichella, and Lionel Briand. 2017. Automatically Repairing Web Application Firewalls Based on Successful SQL Injection Attacks. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. 339–350. https://doi.org/10.1109/ISSRE.2017.28

[5] Simon Applebaum, Tarek Gaber, and Ali Ahmed. 2021. Signature-based and Machine-Learning-based Web Application Firewalls: A Short Survey. *Procedia Computer Science* 189 (2021), 359–367. AI in Computational Linguistics.

[6] Mahmoud Baklizi, Issa Atoum, Nibras Abdullah, Ola A Al-Wesabi, Ahmed Ali Otoom, and Mohammad Al-Sheikh Hasan. 2022. A Technical Review of SQL Injection Tools and Methods: A Case Study of SQLMap. *International Journal of Intelligent Systems and Applications in Engineering* 10, 3 (2022), 75–85.

[7] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2154–2156.

[8] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[9] Igino Corona, Battista Biggio, Matteo Contini, Luca Piras, Roberto Corda, Mauro Mereu, Guido Mureddu, Davide Ariu, and Fabio Roli. 2017. DeltaPhish: Detecting Phishing Webpages in Compromised Websites. In *Computer Security – ESORICS 2017*. Springer International Publishing, Cham, 370–388.

[10] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995). https://doi.org/10.1023/A:1022627411411

[11] Adele Cutler, D. Richard Cutler, and John R. Stevens. 2012. *Random Forests*. Springer US, Boston, MA, 157–175. https://doi.org/10.1007/978-1-4419-9326-7_5

[12] Luca Demetrio, Andrea Valenza, Gabriele Costa, and Giovanni Lagorio. 2020. WAF-A-MoLE: Evading Web Application Firewalls through Adversarial Machine Learning. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (Brno, Czech Republic) *(SAC '20)*. Association for Computing Machinery, New York, NY, USA, 1745–1752. https://doi.org/10.1145/3341105.3373962

[13] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Dan Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2017. Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing* 16 (2017), 711–724.

[14] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 321–338. https://www.usenix.org/conference/usenixsecurity19/presentation/demontis

[15] Christian Folini and Ivan Ristic. 2017. *ModSecurity Handbook, Second Edition* (2nd ed.). Feisty Duck, London, GBR.

[16] Ouissem Ben Fredj, Omar Cheikhrouhou, Moez Krichen, Habib Hamam, and Abdelouahid Derhab. 2020. An OWASP Top Ten Driven Survey on Web Application Protection Methods. In *Risks and Security of Internet and Systems: 15th International Conference, CRiSIS 2020, Paris, France, November 4–6, 2020, Revised Selected Papers* (Paris, France). Springer-Verlag, Berlin, Heidelberg, 235–252. https://doi.org/10.1007/978-3-030-68887-5_14

[17] Carmen Torrano Giménez, Alejandro Pérez Villegas, and Gonzalo Álvarez Marañón. 2010. HTTP data set CSIC 2010. *Information Security Institute of CSIC (Spanish Research National Council)* 64 (2010).

[18] Yuting Guan, Junjiang He, Tao Li, Hui Zhao, and Baoqiang Ma. 2023. SSQLi: A Black-Box Adversarial Attack Method for SQL Injection Based on Reinforcement Learning. *Future Internet* 15, 4 (2023), 133.

[19] William G. J. Halfond and Alessandro Orso. 2006. Preventing SQL Injection Attacks Using AMNESIA. In *Proceedings of the 28th International Conference on Software Engineering* (Shanghai, China) *(ICSE '06)*. Association for Computing

Machinery, New York, NY, USA, 795–798. https://doi.org/10.1145/1134285.1134416

[20] Mojtaba Hemmati and Mohammad Ali Hadavi. 2021. Bypassing Web Application Firewalls Using Deep Reinforcement Learning. In *2021 18th International ISC Conference on Information Security and Cryptology (ISCISC)*. 35–41. https://doi.org/10.1109/ISCISC53448.2021.9720473

[21] Adam Jakobsson and Isak Häggström. 2022. *Study of the techniques used by OWASP ZAP for analysis of vulnerabilities in web applications*. Ph. D. Dissertation. Linköping University. http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-186346

[22] Anamika Joshi and V Geetha. 2014. SQL Injection detection using machine learning. In *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. 1111–1115. https://doi.org/10.1109/ICCICCT.2014.6993127

[23] Debabrata Kar, Suvasini Panigrahi, and Srikanth Sundararajan. 2016. SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM. *Computers & Security* 60 (2016), 206–225. https://doi.org/10.1016/j.cose.2016.04.005

[24] Ke Li, Heng Yang, and Willem Visser. 2022. DaNuoYi: Evolutionary Multi-Task Injection Testing on Web Application Firewalls. *CoRR* abs/2206.05743 (2022). https://doi.org/10.48550/arXiv.2206.05743 arXiv:2206.05743

[25] Ao Luo, Wei Huang, and Wenqing Fan. 2019. A CNN-based Approach to the Detection of SQL Injection Attacks. In *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*. IEEE, 320–324.

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=rJzIBfZAb

[27] Balume Mburano and Weisheng Si. 2018. Evaluation of web vulnerability scanners based on owasp benchmark. In *2018 26th International Conference on Systems Engineering (ICSEng)*. IEEE, 1–6.

[28] Christoph Molnar. 2022. *Interpretable Machine Learning* (2 ed.). Lulu.com. https://christophm.github.io/interpretable-ml-book

[29] Olajide Ojagbule, Hayden Wimmer, and Rami J. Haddad. 2018. Vulnerability Analysis of Content Management Systems to SQL Injection Using SQLMAP. In *SoutheastCon 2018*. 1–7. https://doi.org/10.1109/SECON.2018.8479130

[30] OWASP Foundation Inc. 2021. OWASP Top 10. https://owasp.org/Top10/ Available online. Accessed on 22 February 2023.

[31] OWASP Foundation Inc. 2023. OWASP Core Rule Set. https://coreruleset.org.

[32] OWASP Foundation Inc. 2023. OWASP Core Rule Set documentation. *Anomaly Scoring*. https://coreruleset.org/docs/concepts/anomaly_scoring/.

[33] OWASP Foundation Inc. 2023. OWASP Core Rule Set documentation. *Engine and integration options*. https://coreruleset.org/docs/deployment/engine_integration_options/.

[34] OWASP Foundation Inc. 2023. OWASP Core Rule Set documentation. *Paranoia Levels*. https://coreruleset.org/docs/concepts/paranoia_levels/.

[35] OWASP Foundation Inc. 2023. OWASP Core Rule Set documentation. *Rules*. https://coreruleset.org/docs/rules/rules/.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[37] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*. IEEE, 1332–1349.

[38] Zhenqing Qu, Xiang Ling, and Chunming Wu. 2022. *AutoSpear: Towards Automatically Bypassing and Inspecting Web Application Firewalls*. Black Hat Asia 2022, Singapore. https://www.blackhat.com/asia-22/briefings/schedule/#autospear-towards-automatically-bypassing-and-inspecting-web-application-firewalls-25503

[39] Chedy Raıssi, Johan Brissaud, Gérard Dray, Pascal Poncelet, Mathieu Roche, and Maguelonne Teisseire. 2007. Web analyzing traffic challenge: description and results. In *Proceedings of the ECML/PKDD*. 47–52.

[40] Sebastian Raschka. 2018. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. *CoRR* abs/1811.12808 (2018). http://arxiv.org/abs/1811.12808

[41] Jatesh Jagraj Singh, Hamman Samuel, and Pavol Zavarsky. 2018. Impact of Paranoia Levels on the Effectiveness of the ModSecurity Web Application Firewall. In *2018 1st International Conference on Data Intelligence and Security (ICDIS)*. 141–144. https://doi.org/10.1109/ICDIS.2018.00030

[42] Timilehin David Sobola, Pavol Zavarsky, and Sergey Butakov. 2020. Experimental Study of ModSecurity Web Application Firewalls. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. 209–213. https://doi.org/10.1109/BigDataSecurity-HPSC-IDS49724.2020.00045

[43] Vladimir Vapnik. 1999. *The nature of statistical learning theory*. Springer science & business media.

[44] Xianbo Wang and Han HU. 2020. Evading Web Application Firewalls with Reinforcement Learning. https://openreview.net/forum?id=m5AntlhJ7Z5

[45] Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. 2023. Mutation-Based Fuzzing. In *The Fuzzing Book*. CISPA Helmholtz Center for Information Security. https://www.fuzzingbook.org/html/MutationFuzzer.html Retrieved 2023-01-07 14:53:00+01:00.