ModSec-AdvLearn: Countering Adversarial SQL Injections with Robust Machine Learning

Giuseppe Floris[§], Christian Scano[§], Biagio Montaruli[§], Luca Demetrio[*], Andrea Valenza, Luca Compagna, Davide Ariu, Luca Piras, Davide Balzarotti, and Battista Biggio[*], *Fellow, IEEE*

***Abstract*—Many Web Application Firewalls (WAFs) leverage the OWASP Core Rule Set (CRS) to block incoming malicious requests. The CRS consists of different sets of rules designed by domain experts to detect well-known web attack patterns. Both the set of rules and the weights used to combine them are manually defined, yielding four different default configurations of the CRS. In this work, we focus on the detection of SQL injection (SQLi) attacks, and show that the manual configurations of the CRS typically yield a suboptimal trade-off between detection and false alarm rates. Furthermore, we show that these configurations are not robust to adversarial SQLi attacks, i.e., carefully-crafted attacks that iteratively refine the malicious SQLi payload by querying the target WAF to bypass detection. To overcome these limitations, we propose (i) using machine learning to automate the selection of the set of rules to be combined along with their weights, i.e., customizing the CRS configuration based on the monitored web services; and (ii) leveraging adversarial training to significantly improve its robustness to adversarial SQLi manipulations. Our experiments, conducted using the well-known open-source ModSecurity WAF equipped with the CRS rules, show that our approach, named ModSec-AdvLearn, can (i) increase the detection rate up to 30%, while retaining negligible false alarm rates and discarding up to 50% of the CRS rules; and (ii) improve robustness against adversarial SQLi attacks up to 85%, marking a significant stride toward designing more effective and robust WAFs. We release our open-source code at https://github.com/pralab/modsec-advlearn.**

***Index Terms*—web application firewalls, machine learning, sql injection, adversarial training**

I. 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 [1]. Among these, 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 [2–5], the Open Web Application Security Project (OWASP) Foundation still classifies it as one of the top-10 most dangerous web threats [6].

Web Application Firewalls (WAFs) are commonly used as a defense tool in enterprise systems to counter such attacks and protect web applications [5, 7]. They work by filtering the incoming requests directed towards the web applications and blocking suspicious connections. To this end, many available WAF solutions leverage the OWASP Core Rule Set, i.e., a collection of signatures designed to detect well-known web attack patterns. The CRS rules have all been developed by experts in the domain of web security in the last decade, helping to withstand a vast plethora of web-based attacks. The CRS v4.0.0 (one of the latest stable versions) used in this work includes 319 rules, out of which 170 target critical injection attacks [8]. Within this set, SQLi is the most represented class of injection attack counting 62 rules.

The CRS rules are sub-divided into four sets, each identified by a specific *Paranoia Level (PL)*. These sets are constructed such that PL1 $\subset \ldots \subset$ PL4, i.e., increasing the PL amounts to include more CRS rules, with PL4 including all of them. In practice, higher PLs tend to exhibit a higher detection rate but also cause a higher number of false alarms. Within each PL, rules are assigned a specific weight, referred to as their *severity level*. The severity level of each rule is assigned by domain experts, based on their subjective evaluation of the potential impact of the attack that such a rule aims to prevent. Then, the score associated with each incoming request is computed as the sum of the severity levels associated with the firing rules. If such a score exceeds a given threshold, the incoming request is blocked. More details on how the PL1-PL4 CRS configurations work are provided in Sect. II.

While the domain knowledge poured in developing the CRS rules is extremely valuable, in this work we use the well-known ModSecurity WAF [9] equipped with the CRS rules to show that the heuristic choices made to select and combine such rules can lead to: (i) a suboptimal trade-off between detection rate and false alarms; and (ii) a substantial lack of robustness to *adversarial* SQLi attacks, i.e., functionality-preserving manipulations of the SQLi attack payload aimed to evade detection [10, 11]. To overcome these limitations, we then propose a novel robust machine learning approach to selecting and combining the CRS rules, named ModSec-AdvLearn, which is conceptually represented in Fig. 1 and detailed in Sect. III. This approach is built upon two main

G. Floris, C. Scano and B. Biggio are with the Dept. of Electrical and Electronic Engineering, University of Cagliari, 09124 Cagliari, Italy e-mail: (name.surname@unica.it), C. Scano is also with the Department of Computer, Control and Management Engineering, Sapienza University, Rome, Italy.

B. Montaruli and D. Balzarotti are with the Dept. of Digital Security, EURECOM, 06410 Biot, France, e-mail: (name.surname@eurecom.fr).

Luca Demetrio is with the Department of Informatics, Bioengineering, Robotics and Systems Engineering (DIBRIS), University of Genova, 16146 Genova, Italy e-mail: (luca.demetrio@unige.it).

Andrea Valenza is with Prima Assicurazioni, 20131 Milano, Italy e-mail: (andrea.valenza@prima.it).

Luca Compagna is with Endor Labs, e-mail: (lcompagna@endor.ai).

Davide Ariu and Luca Piras are with Pluribus One, 09128 Cagliari, Italy e-mail: (name.surname@pluribus-one.it).

[§] means equal contribution, while [*] refers to corresponding authors.

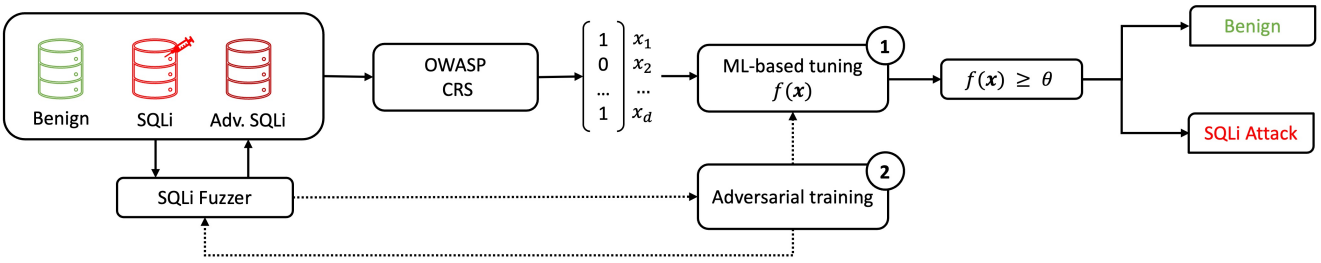

Fig. 1: Conceptual representation of ModSec-AdvLearn. A machine-learning model is trained using the CRS rules as input features, and leveraging our novel adversarial training approach to improve robustness against adversarial SQLi attacks.

contributions. First, we propose using machine learning (ML) to automate both rule selection and weighting, adjusting the CRS configuration based on the traffic data collected from the monitored web services, and building on our preliminary findings in [11]. The underlying idea is to train a linear ML model using *all* the CRS rules as input features, aiming to improve the trade-off between detection and false alarm rates, by specializing the model to the specifics of the traffic data collected from the monitored applications. Furthermore, enforcing a sparse regularization during training enables the selection of an effective subset of rules as a by-product, avoiding the need for manual selection of the CRS rules in each PL. The second main contribution of this work is the definition of a novel adversarial training scheme to improve model robustness against adversarial SQLi attacks. To craft these attacks, we leverage WAF-A-MoLE [10], i.e., a black-box mutational fuzzer [12] that iteratively selects the best combination of random manipulations of SQLi payloads to reduce their probability of being detected by the targeted WAF. We also show in Sect. III-B that using ℓ_∞-norm regularization on a linear model yields equivalent robust solutions, avoiding the computational burden of optimizing attacks during training.

Through our experiments, reported in Sect. IV and conducted on two publicly-available datasets [10, 11], we show that ModSec-AdvLearn overcomes the limitations of the current CRS, by detecting 30% attacks more with much fewer rules. ModSec-AdvLearn also provides an unprecedented level of robustness against adversarial SQLi attacks, i.e., 85% more than the default CRS configurations. By deepening the investigation of this result, we discover that ModSec-AdvLearn gives more emphasis to rules that are (i) less affected by adversarial SQLi attacks and (ii) accidentally triggered by side-effect artifacts introduced by the adversarial SQLi manipulations.

To conclude, we remark that our work is the first to demonstrate the effectiveness of adversarial training in the WAFs domain (specifically, for detecting SQLi attacks) when leveraging state-of-the-art input-space SQLi manipulations. This is completely different from other domains like image classifiers, where adversarial training is not sufficient to achieve a high level of robustness, and the effective mitigation of the risks presented by adversarial examples is still an open issue. For this reason, we firmly believe that our work provides interesting and novel insights on how to design robust machine learning models for cybersecurity. We discuss these aspects along with related work in Sect. V, and the limitations of our approach as well as the corresponding future research directions in Sect. VI. We have also publicly released our code to foster reproducibility of our work.[1]

II. Background

We introduce here SQLi attacks and the OWASP CRS project. We then describe how to generate adversarial SQLi attacks using state-of-the-art fuzzing techniques.

A. SQL Injection (SQLi)

These attempts to retrieve or alter sensitive information from a target database, modifying 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 input and simply concatenates it with the query, the SQL fragment is interpreted as part of the original SQL query. The login form of a web application is a paradigmatic example (Listing 1). The credentials of a user are provided in two input fields (e.g., `$user` and `$passwd`) and sent via an HTTP request. The credentials are then checked server-side via a database query. However, a malicious user injects SQL fragments in the `$user` parameter, *e.g.*, "`admin'-- `". As shown in Listing 2, this bypasses the original SQL query's password check, resulting in a successful SQLi attack, allowing login with just a valid username.

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

Listing 1: Example of SQL query vulnerable to injection.

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

Listing 2: Example of SQL Injection on Listing 1.

[1] https://github.com/pralab/modsec-advlearn

B. The OWASP Core-Rule-Set (CRS) Project

This open-source initiative is one of the most widely-used sets of detection rules targeting OWASP Top 10 web security risks [6]. It is not only the reference rule set of several open-source WAF solutions like ModSecurity [9] and Coraza,[2] but is also adopted in many commercial solutions including Google Cloud Armor, Microsoft Azure, and Cloudflare WAFs [8].

Detection Rules. These are regular expressions (regex) that match specific byte patterns in requests. For instance, line 3 of Listing 3 captures several patterns of comments commonly used in SQLi attacks such as "**;--**" and "**--** ". Each rule is denoted by a unique identifier (`id` in line 4), whose suffix also indicates the type of attack it is designed to identify (those starting with 942 target SQLi attacks [8]). Notable among configuration settings are the Paranoia Level (line 8) and Severity Level (line 9), which are explained below.

Paranoia Level. It defines the set of rules that are enabled to analyze the incoming HTTP requests [8]. The CRS includes four PLs (PL1 - PL4) and each rule is assigned to a specific PL; e.g., the rule in Listing 3 belongs to PL1 (line 8). Moreover, rules are grouped together by PL in a nested way: setting a certain PL enables all the rules assigned to that 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, PL4 will enable all the rules.

Severity Level. Each CRS rule is heuristically given a *severity level*, i.e., a positive integer that quantifies how severe the corresponding attack could be [8]. The WAF applies the rules on each incoming request, and sums the severity levels of the firing rules. If the aggregated score exceeds a predefined threshold, the request is flagged as malicious. The CRS defines four severity levels: CRITICAL (5), ERROR (4), WARNING (3) and NOTICE (2); e.g., the severity level of the rule in Listing 3 is CRITICAL (line 9), so it contributes to the aggregated score with a value of 5.

```
1  SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/
2  |REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/*
3  "@rx (?i)/\*[\s\v]*?[!\+](?:[\s\v\(-\)\-0-9=A-Z_a-z]+)
       ?\*/" \
4    "id:942500,
5    block,
6    msg:'MySQL in-line comment detected',
7    tag:'attack-sqli',
8    tag:'paranoia-level/1',
9    severity:'CRITICAL',
10   setvar:'tx.sql_injection_score=+
11     %{tx.critical_anomaly_score}',
12   setvar:'tx.inbound_anomaly_score_pl1=+
13     %{tx.critical_anomaly_score}'"
```

Listing 3: CRS rule detecting typical comments in SQLi.

C. Adversarial SQLi Attacks against WAFs

In the context of WAFs, the problem of finding SQLi attacks that can bypass the target WAF is *adversarial in nature*. To this end, the attacker may manipulate SQLi attack payload to evade detection while preserving its malicious functionality [10, 13]. For instance, the SQLi rule reported in Listing 3 can detect the following SQLi payload: **admin'** `OR 1=1; --'`. However, by inserting a white space character (`' '`) in the original attack payload, we generate a semantically-equivalent SQLi attack: **admin'** `OR 1=1; --'`, that can evade the rule.

[2]https://coraza.io

TABLE I: Manipulation functions applied by WAF-A-MoLE.

Manipulation	Effect on payload
Case Swapping	CS(`admin' OR 1=1`) → `ADmIn' oR 1=1`
Whitespace Substitution	WS(`admin' OR 1=1`) → `admin'\n OR 1=1`
Comment Injection	CI(`admin' OR 1=1`) → `admin'/**/OR 1=1`
Comment Rewriting	CR(`admin'/**/OR 1=1`) → `admin'/*abc*/OR 1=1`
Integer Encoding	IE(`admin' OR 1=1`) → `admin' OR 0x1=1`
Operator Swapping	OS(`admin' OR 1=1`) → `admin' OR 1 LIKE 1`
Logical Invariant	LI(`admin' OR 1=1`) → `admin' OR 1=1 AND 2<>3`

WAF-a-MoLE. Our methodology builds upon WAF-A-MoLE [10], a state-of-the-art, open-source SQLi guided mutational fuzzer [12], aimed at finding semantically-equivalent SQLi attacks that evade detection through the application of functionality-preserving manipulations. These manipulations, detailed in Table I, can be encoded as a function $h(\boldsymbol{z}, \boldsymbol{\delta})$, where $\boldsymbol{z}$ is the SQLi query to be modified, and $\boldsymbol{\delta}$ are the parameters defining the perturbation. For instance, WAF-A-MoLE can include new comments into the SQLi, adding always-true or always-false statements, converting numbers to a different base, or replacing them with SQL commands that, once evaluated, produce the same number. Such choice is controlled by $\boldsymbol{\delta}$, which specifies the type of manipulation and the content that should be injected or replaced. WAF-A-MoLE then iteratively refines the choice of $\boldsymbol{\delta}$ to decrease the confidence of the targeted WAF into classifying the modified SQLi payload as an attack. This is achieved by generating several candidates through random choices of $\boldsymbol{\delta}$ in each iteration, and retaining only those that successfully reduce the confidence score attributed by the WAF. In this work, we will use WAF-A-MoLE (i) to show that the standard configurations of the CRS can be bypassed by optimizing adversarial SQLi attacks against them, and (ii) to generate adversarial SQLi queries for our novel *problem-space* adversarial training approach.[3]

III. Robust Machine Learning against Adversarial SQLi Attacks

We detail here how we design our robust ML approach to (i) improving the tradeoff between detection and false alarm rates by optimizing the selection and combination of the CRS rules (Sect. III-A), and (ii) improving robustness to adversarial SQLi attacks (Sect. III-B).

A. ModSec-Learn: Machine Learning for CRS

We start by discussing the building block developed from our initial findings, i.e., ModSec-Learn (step 1 of Fig. 1). It consists of two components: (i) a feature extraction phase that encodes the CRS rules into a vector representation; and (ii) a ML model that learns how to optimally combine the CRS rules, avoiding the manual tuning of their severity scores.

[3]We refer to our approach as problem-space adversarial training to differentiate it from approaches that simulate the effect of attacks by modifying only their feature vectors, without even producing the actual samples [14].

Detection Rules as Features. The input space is represented by SQL queries that are classified as malicious or benign by a ML model. Each SQL query is a string of readable characters, represented as $\boldsymbol{z} \in \mathcal{Z}$, being $\mathcal{Z}$ the space of all possible queries. Let $\mathcal{D}$ be the set of selected SQLi rules from CRS, and $\mathsf{d} = |\mathcal{D}|$ its cardinality. We denote with $\phi : \mathcal{Z} \mapsto \mathcal{X} = \{0,1\}^{\mathsf{d}}$ a function that maps a SQL query $\boldsymbol{z}$ to a d-dimensional Boolean feature vector $\boldsymbol{x} = (\phi_1(\boldsymbol{z}), \ldots, \phi_{\mathsf{d}}(\boldsymbol{z}))$, where each $\phi_j(\boldsymbol{z})$ corresponds to evaluating the *j*-th SQLi rule on the input query $\boldsymbol{z}$. Each $\phi_j(\boldsymbol{z})$ returns 1 if the corresponding rule has been triggered by the SQL query $\boldsymbol{z}$, and 0 otherwise.

Optimal Combination of CRS Rules with ML. To optimally tune the contribution of the CRS rules towards effectively classifying the input requests we leverage three different ML algorithms on the feature representation defined above: two linear models, i.e., Support Vector Machines (SVMs) [15] and Logistic Regression (LR) [16], both with ℓ_1 and ℓ_2 regularization; and a non-linear Random Forest (RF) model [17].

Linear models are particularly valuable as they can automatically adjust the severity score assigned to each rule, rather than relying on default CRS values, whereas non-linear models may give us an indication of the best performance achievable. Furthermore, when sparse (ℓ_1) regularization is applied, linear models can effectively select an optimal subset of CRS rules, potentially eliminating the need for predefined PLs. Although our approach is applicable to both linear and non-linear models, integrating a non-linear model within the existing CRS rules may pose additional complexity and scalability issues, while also worsening the interpretability of the WAF decisions. Conversely, the weights learned by any linear model can be directly plugged-in into the CRS rules without any significant disadvantage. Let us finally remark that, compared to our previous work in [11], we extend the evaluation of ModSec-Learn models by assessing their robustness against adversarial attacks, and also considering an additional dataset [10].

B. ModSec-AdvLearn: Robustness against Adversarial SQLi

While ModSec-Learn can learn to automatically select and combine the CRS rules from the training data, yielding a better tradeoff between detection and false alarm rates, it may not guarantee a sufficient degree of robustness against adversarial SQLi attacks. Hence, we detail here how we extend ModSec-Learn to improve robustness against adversarial SQLi attacks. We refer to this approach as ModSec-AdvLearn (step 2 of Fig. 1). The underlying idea is to leverage *adversarial training* (AT) [18, 19] to include adversarial SQLi examples during training, thereby giving the model the ability to withstand the corresponding evasive attack patterns at test time.

In the common setting of image classification, AT leverages gradient-based attacks to craft adversarial examples, as the corresponding optimization problem is end-to-end differentiable, and the considered perturbations are simply additive. This is not directly applicable in the case of SQLi attacks, as well as in other domains [20, 21], in which models are not end-to-end differentiable (given the presence of non-differentiable pre-processing and feature extraction steps) and perturbations are not additive. We thus consider here two distinct approaches to performing AT: *feature-space* and *problem-space* AT.

Feature-space AT. In this setting, we make the naïve assumption that each adversarial SQLi attack can enable or disable up to a number λ of CRS rules to evade the target WAF. This amounts to optimizing the following min-max objective:

$$\min_{\boldsymbol{w}} \max_{\|\boldsymbol{\delta}_i\|_1 \leq \lambda} \sum_i \mathcal{L}(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i + \boldsymbol{\delta}_i)) \,, \tag{1}$$

where (i) $\boldsymbol{x}_i = \phi(\boldsymbol{z}_i)$ is the d-dimensional Boolean vector representing the activations of the CRS rules for the given SQL sample $\boldsymbol{z}_i$; (ii) $y_i \in \mathcal{Y} = \{-1, +1\}$ is its label; (iii) $f_{\boldsymbol{w}} : \mathcal{X} \to \mathbb{R}$ is the ML model, parameterized by $\boldsymbol{w}$, which classifies a sample as positive if $f_{\boldsymbol{w}}(\boldsymbol{x}) \geq 0$, and as negative otherwise; (iv) $\mathcal{L} : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ is the loss function to be minimized; and (v) $\boldsymbol{\delta}_i$ is the manipulation that switches on or off at maximum λ rules from the CRS (expressed as an ℓ_1 norm constraint). Furthermore, it must also hold that $\boldsymbol{x}_i + \boldsymbol{\delta}_i \in \mathcal{X} = \{0,1\}^{\mathsf{d}}$, as the activations have to remain Boolean also after perturbation. In practice, the min-max problem is solved iteratively. In each iteration, the inner problem amounts to finding adversarial examples against the given model, while the outer problem adjusts the model parameters $\boldsymbol{w}$ to re-classify them correctly.[4]

In this work, we do not solve the problem given in Eq. 1 directly, as it is too computationally demanding. Instead, we derive an equivalent formulation for linear SVMs based on solving the inner problem in closed form, which simply amounts to using a different regularization term.

Robustness through Regularization with SecSVM. As originally shown by by Xu *et al.* [22], and subsequently adapted in [20] to the case of Android malware detection, the inner problem in Eq. 1 can be solved in closed form when the loss function $\mathcal{L}$ is *linear*. This is the case, e.g., when using the hinge loss and a linear model $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + b$, as in linear SVM training. In this case, the inner problem in Eq. 1 for each sample can be rewritten as:

$$\max_{\|\boldsymbol{\delta}\|_1 \leq \lambda} \mathcal{L}(y, f_{\boldsymbol{w}}(\boldsymbol{x})) + \boldsymbol{\delta}^\top \nabla \mathcal{L}(y, f_{\boldsymbol{w}}(\boldsymbol{x})) \,, \tag{2}$$

where the gradient $\nabla \mathcal{L}(y, f_{\boldsymbol{w}}(\boldsymbol{x}))$ corresponds to the weight vector $\boldsymbol{w}$. The above problem then amounts to maximizing a scalar product over an ℓ_1-norm constraint, i.e., $\max_{\|\boldsymbol{\delta}\|_1 \leq \lambda} \boldsymbol{\delta}^\top \boldsymbol{w}$, whose solution is proportional to the dual norm of the weight vector, given as $\lambda \|\boldsymbol{w}\|_\infty$ [22]. This means that we can rewrite the robust (min-max) optimization problem given in Eq. 1 as a much simpler regularized problem:

$$\min_{\boldsymbol{w}} \sum_i \mathcal{L}(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) + \lambda \|\boldsymbol{w}\|_\infty \,. \tag{3}$$

This finding sheds light on the role of the regularization term, showing that its choice should be based on the type of noise that affects the input data. In particular, it tells us that ℓ_∞ is

[4]Note that, for simplicity, in the given formulation we consider adversarial modifications of both benign and malicious training samples, but this can be easily adjusted to consider only manipulations of malicious SQLi queries.

the optimal regularizer for training a robust model against ℓ_1-norm (sparse) perturbations. It is also not difficult to see that, analogously, the standard ℓ_2-norm SVM is optimal against ℓ_2-norm (dense) perturbations [20, 22].

The above problem can be equivalently re-parameterized by (i) replacing the regularization parameter λ with the hyperparameter t, i.e., bounding the weight values $\boldsymbol{w}$ in $[-t, t]$; and (ii) introducing the slack variables $\boldsymbol{\xi}$ to measure how far each sample $\boldsymbol{z}_i$ is from being correctly classified:

$$\min_{\boldsymbol{w},\xi} \quad \sum_i \xi_i \tag{4}$$

$$\text{s.t.} \quad \xi_i \geq 1 - y_i f_{\boldsymbol{w}}(\boldsymbol{x}_i), \quad \forall i \in 1, \ldots, n \tag{5}$$

$$\xi_i \geq 0, \quad \forall i \in 1, \ldots, n \tag{6}$$

$$-t \leq w_j \leq t, \quad \forall j \in 1, \ldots, d\,. \tag{7}$$

The given formulation corresponds to a linear programming problem in its canonical form, which can be solved using standard, off-the-shelf solvers, such as the simplex algorithm or interior-point methods. In this work, the optimization problem is solved using the linear programming solver provided by the SciPy library[5]. We refer to this robust learning approach as Secure Support Vector Machine (SecSVM) in our experiments.[6]

Problem-space AT with WAF-A-MoLE. Let us now define a different approach to learning robust models directly against problem-space perturbations. The reason is that hardening models via feature-space AT or with ad-hoc regularization methods may provide an overly pessimistic approach to feature manipulation that does not consider the specific constraint of realizable, semantic- and functionality-preserving SQLi attacks. In particular, practical SQLi manipulations may not enable switching on or off individual CRS rules independently, and may inadvertently trigger certain rules even if the attack aims to bypass the detection. Furthermore, such manipulations cannot be modeled as additive perturbations, and computing them typically requires inverting a complicated, non-differentiable feature extraction step; in our case, this would amount to reversing the inner workings of each CRS rule. For this reason, feasible SQLi attacks, as many other adversarial perturbations aimed to bypass ML models for cybersecurity-related tasks [21], cannot be typically optimized via gradient descent directly. To overcome these issues, we consider here a more general problem-space AT procedure, defined as:

$$\min_{\boldsymbol{w}} \max_{\boldsymbol{\delta}_i \in \Delta} \sum_i \mathcal{L}(y_i, f_{\boldsymbol{w}}(\phi(h(\boldsymbol{z}_i, \boldsymbol{\delta}_i)))\,, \tag{8}$$

where $h(\boldsymbol{z}, \boldsymbol{\delta})$ is a manipulation function that modifies the input SQL query $\boldsymbol{z}$ and returns a semantic-preserving SQL query $\boldsymbol{z}'$, based on the choice of its input parameters $\boldsymbol{\delta} \in \Delta$. The set Δ constrains the input manipulations described in Table I to produce valid samples, e.g., picking only visible characters when adding or re-writing comments, produce always-true or always-false conditions that do not change the original evaluation of the payload, or encoding integers in a specific base different from the original one.

[5]https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html

[6]Note that, even if we keep the same name of the approach proposed in [20], our SecSVM implementation is different, as we are neither using custom bounds on each feature value nor any ℓ_2 regularization.

Algorithm 1: Adversarial training of ModSec-AdvLearn with WAF-A-MoLE

Input : $\mathcal{D} = (\boldsymbol{z}_i, y_i)_{i=1}^{M}$, the training set of SQL samples with labels; f, the ML model; $\mathcal{L}$, the loss function; N, the number of adversarial SQLi attacks to be added to the initial training set.

Output: $f_{\boldsymbol{w}^\star}$, the model with re-trained parameters $\boldsymbol{w}^\star$

1 $\mathcal{Z}' \leftarrow \{\boldsymbol{z}_i\}_{i=1}^{N}$ with $\boldsymbol{z}_i \sim \mathcal{D}$ s.t. $y_i = +1$

2 **for** $\boldsymbol{z}$ **in** $\mathcal{Z}'$

3 $\quad \boldsymbol{z}^\star \leftarrow$ *WAF-A-MoLE*$(\boldsymbol{z}, f)$

4 $\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \{\boldsymbol{z}^\star\}$; $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{+1\}$;

5 $\boldsymbol{w}^\star \leftarrow \arg\min_{\boldsymbol{w}} \frac{1}{|\mathcal{Z}|} \sum_{i=0}^{|\mathcal{Z}|} \mathcal{L}(y_i, f_w(\boldsymbol{z}_i))$

6 return $f_{\boldsymbol{w}^\star}$

To solve the problem given in Eq. 8, we consider a gradient-free *black-box* optimization achieved through WAF-A-MoLE as shown in Alg. 1, modifying only the set of malicious SQLi payloads, and leaving benign SQL queries unchanged. Given a dataset $\mathcal{Z}$ of benign queries and SQLi attacks labeled as 0 and 1 respectively, we first create a new set ($\mathcal{Z}'$) by randomly sampling a given amount of SQLi from the training dataset (line 1). Then, for each SQLi sample of this newly created set, we use WAF-A-MoLE (Sect. II-C) to generate the corresponding adversarial SQLi (line 3) and add it to the training data with its malicious (+1) label (line 4). The parameters of the model are finally optimized on the training set including the adversarial SQLi samples (line 5).

In our experiments, we will also retrain SecSVM with the proposed problem-space AT method, to consider less pessimistic, but practical, adversarial attacks. In principle, this should allow us to further improve the robustness-detection tradeoff of SecSVM against real-world SQLi attacks. We would like to finally remark that, even if we leverage an existing tool like WAF-A-MoLE to this end, to our knowledge, this is the first attempt to define a novel problem-space adversarial training approach to hardening ML-based WAFs against practical adversarial SQLi attacks.

IV. Experimental Analysis

We report here three different experiments to validate our methodology. First, we evaluate the detection capabilities of the CRS. This is achieved by using the vanilla ModSecurity WAF as the underlying engine (Sect. IV-B), showing that its naïve approach of combining the CRS rules based on manually-assigned weights is largely suboptimal and significantly vulnerable to adversarial SQLi attacks. Second, we empirically show that the ML-based tuning adopted by ModSec-Learn allows one to fill the gaps of the vanilla ModSecurity, by significantly enhancing its detection rate up to 30%, and we continue highlighting how ModSec-Learn enhances the

performances thanks to the adaptation of weights, while also reducing the number of rules needed (Sect. IV-C). Third, we present the results of our novel adversarial training approach showing that ModSec-AdvLearn is 85% more robust than ModSecurity (Sect. IV-D).

A. Experimental Setup

In this section, we describe the two datasets used in our analysis, along with the setup of the ModSecurity WAF, the WAF-A-MoLE fuzzer, and the ML models used.

Datasets. We conduct our experiments using two datasets. The first one, WAF-A-MoLE Dataset [10], which consists of 393,629 malicious and 345,199 benign SQL queries. Benign samples were generated from a restricted SQL grammar, while attacks were generated using well-known web security testing tools such as SQLmap and OWASP ZAP [10]. The second one, ModSec-Learn Dataset [11], instead consists of 25,000 malicious SQLi payloads and 25,000 benign HTTP requests, based on real-world traffic. Legitimate samples were collected from the *open-appsec* dataset,[7], which contains samples from various real-world scenarios. Malicious samples were collected from multiple sources, and generated through security testing tools such as SQLmap, by executing it with different tampering scripts designed for payload obfuscation.

We divide each dataset into four subsets: training (`train`), test (`test`), adversarial training (`train-adv`), and adversarial test (`test-adv`). Table II shows the distribution of samples across the four subsets for each dataset: `train` contains an equal number of benign and SQLi queries, and it is used to train our target WAFs; `test`, disjoint from `train`, is used to evaluate the baseline performances of target WAFs, including vanilla ModSecurity, ModSec-Learn, and ModSec-AdvLearn, across different PLs; `train-adv` contains the same samples of `train`, but 50% of the SQLi queries in the WAF-A-MoLE dataset and the 25% of the SQLi queries in the ModSec-Learn dataset are optimized using WAF-A-MoLE against the target WAF for problem-space AT; `test-adv` contains the same samples of `test`, but optimizes all the SQLi queries using WAF-A-MoLE to bypass the target WAF, i.e., to evaluate its robustness. 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 independently optimized against each target model at test time, resulting in the application of different, optimal manipulation strategies. Thus, the sets are independent, ensuring an unbiased evaluation.

ModSecurity and WAF-A-MoLE Setup. We evaluate ModSecurity v3.0.10 with the CRS v4.0.0. Since we focus on the detection of SQLi attacks, we only enable its SQLi rules.[8] We configure WAF-A-MoLE to use a maximum of $2,000$ queries, to ensure convergence of the attack optimization. Attacks are optimized by minimizing the confidence score assigned to the malicious class by the targeted WAF.

[7]https://github.com/openappsec/waf-comparison-project/tree/main/Data

[8]https://github.com/coreruleset/coreruleset/blob/v4.0.0/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf

TABLE II: Number of legitimate, SQLi, and adversarial SQLi samples in `train`, `test`, `train-adv`, and `test-adv`, for the WAF-a-MoLE and ModSec-Learn datasets.

	`train`		`train-adv`		
	Legitimate	SQLi	Legitimate	SQLi	Adversarial SQLi
WAF-a-MoLE	$10,000$	$10,000$	$10,000$	$5,000$	$5,000$
ModSec-Learn	$20,000$	$20,000$	$20,000$	$15,000$	$5,000$
	`test`		`test-adv`		
	Legitimate	SQLi	Legitimate	SQLi	Adversarial SQLi
WAF-a-MoLE	$2,000$	$2,000$	$2,000$	-	$2,000$
ModSec-Learn	$5,000$	$2,000$	$5,000$	-	$2,000$

Implementation Details. We use *pymodsecurity* v0.1.0,[9] as our Python interface to ModSecurity. To efficiently query and test ModSecurity, we have extended WAF-A-MoLE by developing a dedicated *pymodsecurity* interface, which avoids instantiating the whole web server. This interface is available in our open-source repository.

Machine Learning. We leverage scikit-learn v1.4.0 implementations of SVM (LinearSVC), LR, and RF to implement both ModSec-Learn and ModSec-AdvLearn. For the SVM and LR models, we experiment with both ℓ_1 and ℓ_2 regularizers. We implement SecSVM as described in Sect. III-B, using the linear programming solver provided by SciPy. We refer to it as *ModSec-Learn SecSVM* in the reported tables. The hyperparameters of each model are tuned via grid search, performing a 5-fold cross validation on the training set (`train`) to maximize the mean F1 score. In the case of ModSec-Learn, for the SVM and LR, we tune the regularization parameter $C \in \{10^{-3}, 10^{-2}, 10^{-1}, 0.5, 1.0\}$. The best value found is typically $C = 0.5$ for both models. For SecSVM, we tune the hyperparameter $t \in \{0.1, 0.2, \ldots, 1.0\}$. The best t value is typically found to be 0.5. The RF model is used with its default hyperparameters. In the case of ModSec-AdvLearn, adversarial training is applied only for PL4, given that the models trained on this PL demonstrated better performance on the test set (see Sect. IV-C). The hyperparameters are tuned using the same procedure described in this paragraph, finding approximately the same best values, except for SecSVM, for which $t = 1.0$ yields better results.

B. Evaluation of ModSecurity

The first goal of our experimental analysis is to understand the detection capability of the vanilla ModSecurity. Rather than focusing only on CRS default values, we experiment with it over its entire configuration space, considering all the possible values for the PLs and the classification threshold. Hence, for each PL, we compute the Receiver-Operating-Characteristic (ROC) curve, which reports the detection rate, a.k.a. 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. We report our findings with red lines

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

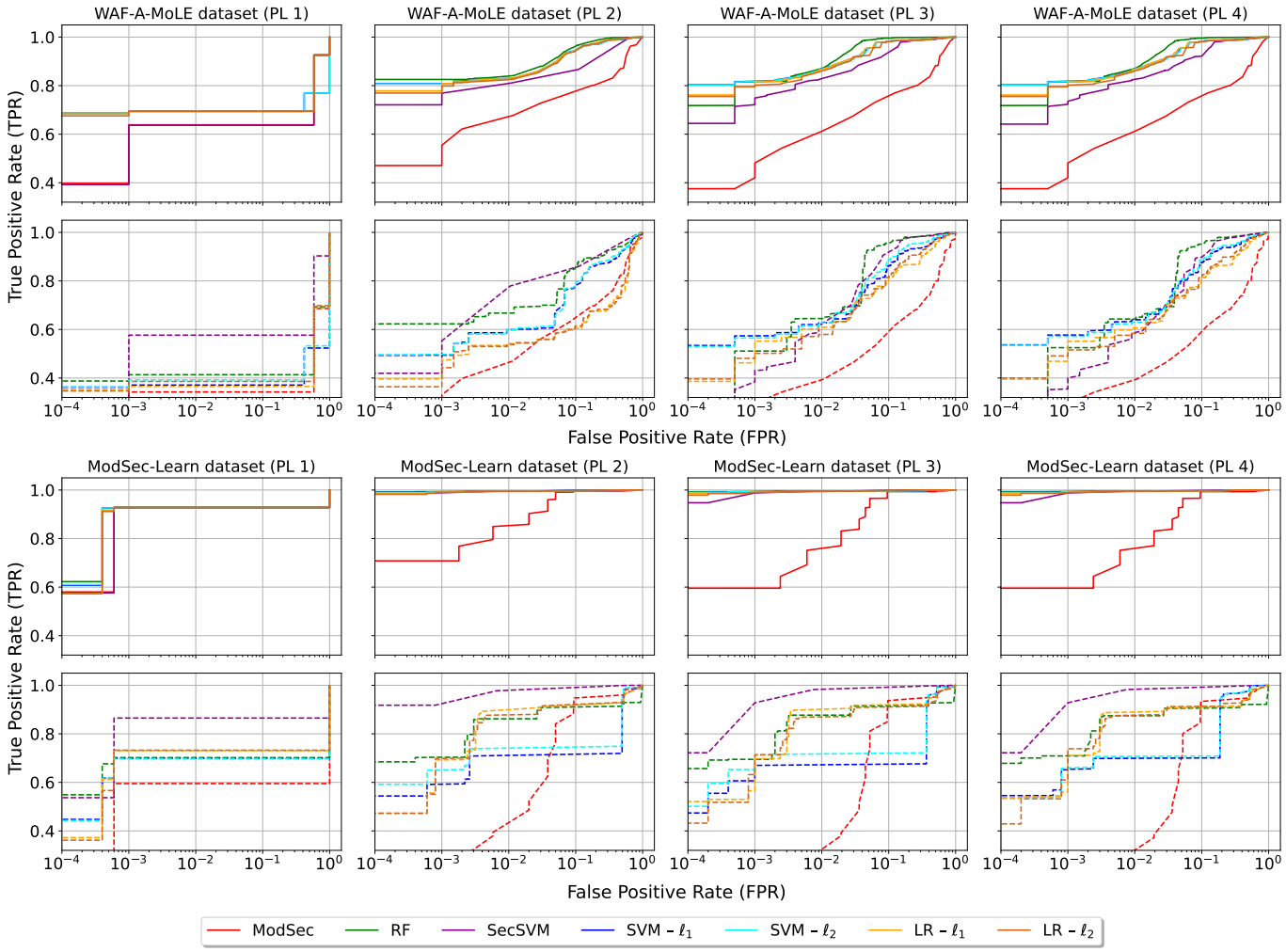

Fig. 2: ROC curves of vanilla ModSecurity (ModSec) and ModSec-Learn approaches (SVM, RF, LR, and SecSVM), evaluated on `test` (solid lines) and `test-adv` (dashed lines), for WAF-A-MoLE (*first two rows*) and ModSec-Learn (*last two rows*) datasets. Each curve reports the fraction of detected SQLi attacks against the fraction of misclassified legitimate requests.

in Fig. 2, while in Table III we extrapolate the TPR values at 1% FPR. We want to point out that, although the ROC curves in Fig. 2 already show the TPR for each possible operating point (i.e., the value of FPR), we report the results in Table III at 1% FPR because it is a reasonable value commonly adopted in the literature [20, 23]. We detail hereafter the key findings of our evaluations of ModSecurity against both the test set (`test`) and the adversarial test set (`test-adv`).

Evaluation on Clean Samples. We first test ModSecurity on the data we have gathered, and the results of this first evaluation are indicated with red solid lines in Fig. 2. The ROC curve of PL1 (default PL for ModSecurity) shows its inability to discriminate between benign and malicious SQL queries, with a TPR of 63.85% at a 1% False Positive Rate (FPR). The results for PL2 are the best among all PLs, with a TPR of 66.82% at 1% FPR. The ROC curves for both PL3 and PL4 are almost identical, as PL4 has only two active rules more than PL3, which do not even improve its detection rate.

Robustness against Adversarial SQLi. The results on the adversarial test set are indicated with red dashed lines in Fig. 2. The outcomes highlight a more alarming trend, thus clearly showing that ModSecurity is not able to withstand adversarial attacks. Both datasets exhibited a pattern similar to the evaluation on the test set (`test`). Specifically, with the WAF-A-MoLE dataset, the True Positive Rate (TPR) drops below 50% at a 1% FPR, which is worse than random guessing. For the ModSec-Learn dataset, the situation is slightly better, but still concerning. Particularly with PL3 and PL4, the TPR at 1% FPR is around 58%, only slightly better than random guessing. This emphasizes that increasing the number of rules does not improve detection capabilities; instead, it worsens them by increasing false positives.

C. Evaluation of ModSec-Learn

We analyze the performance of SVM and LR, using ℓ_1 and ℓ_2 regularizations, SecSVM and RF ModSec-Learn against both the baseline clean and the adversarial samples.

TABLE III: TPR at 1% FPR evaluated on the baseline/adversarial (`test`/`test-adv`) test sets for ModSecurity (ModSec); ModSec-Learn (SVM, LR, SecSVM, and RF); and ModSec-AdvLearn (SVM, LR, SecSVM, and RF) using PL4. We also report the number of active rules (AR) for each model. Results are shown for both datasets: WAF-A-MoLE and ModSec-Learn.

		PL1	PL2	PL3	PL4			
					Base	AR	ModSec-AdvLearn	AR
WAF-A-MoLE dataset								
ModSec vanilla	test	63.85%	**66.82%**	61.00%	61.00%	62/62	—	
	test-adv	34.25%	**45.86%**	39.12%	39.12%	62/62	—	
ModSec-Learn SVM (ℓ_1)	test	69.40%	83.00%	**86.84%**	86.80%	37/62	85.30%	42/62
	test-adv	37.10%	59.90%	62.72%	63.97%	37/62	**82.55%**	42/62
ModSec-Learn SVM (ℓ_2)	test	69.50%	83.10%	86.80%	**86.80%**	50/62	85.25%	51/62
	test-adv	39.20%	59.90%	62.71%	62.95%	50/62	**81.40%**	51/62
ModSec-Learn LR (ℓ_1)	test	69.40%	83.00%	**86.64%**	86.60%	29/62	84.10%	32/62
	test-adv	36.50%	54.20%	60.91%	60.62%	29/62	**79.85%**	32/62
ModSec-Learn LR (ℓ_2)	test	69.50%	82.54%	85.95%	**85.95%**	50/62	84.05%	52/62
	test-adv	38.60%	53.75%	57.95%	58.37%	50/62	**80.15%**	52/62
ModSec-Learn RF	test	69.50%	83.89%	87.00%	87.00%	46/62	**87.22%**	47/62
	test-adv	41.35%	66.75%	64.5%	64.30%	46/62	**84.90%**	47/62
ModSec-Learn SecSVM	test	63.70%	80.78%	82.75%	82.58%	61/62	**84.15%**	61/62
	test-adv	57.65%	76.71%	61.96%	58.68%	61/62	**81.84%**	61/62
ModSec-Learn dataset								
ModSec vanilla	test	**92.80%**	85.22%	75.71%	75.71%	62/62	—	
	test-adv	**59.50%**	42.12%	31.07%	30.85%	62/62	—	
ModSec-Learn SVM (ℓ_1)	test	92.80%	**99.46%**	99.31%	99.41%	35/62	99.26%	41/62
	test-adv	69.90%	70.87%	67.02%	69.86%	35/62	**93.46%**	41/62
ModSec-Learn SVM (ℓ_2)	test	92.80%	**99.46%**	99.31%	99.41%	49/62	99.26%	50/62
	test-adv	69.80%	73.87%	71.47%	70.66%	49/62	**92.86%**	50/62
ModSec-Learn LR (ℓ_1)	test	92.80%	99.46%	99.46%	99.46%	31/62	**99.61%**	30/62
	test-adv	73.05%	89.68%	89.95%	88.89%	31/62	**94.03%**	30/62
ModSec-Learn LR (ℓ_2)	test	92.80%	99.46%	99.46%	99.46%	49/62	**99.61%**	50/62
	test-adv	73.25%	87.75%	86.92%	87.61%	49/62	**94.29%**	50/62
ModSec-Learn RF	test	92.80%	99.56%	99.56%	99.56%	49/62	**99.65%**	50/62
	test-adv	70.02%	86.20%	87.70%	87.45%	49/62	**95.07%**	50/62
ModSec-Learn SecSVM	test	92.80%	99.50%	99.50%	99.50%	56/62	**99.65%**	59/62
	test-adv	86.50%	97.76%	98.26%	98.26%	56/62	**98.41%**	59/62

Evaluation on Clean Samples. We plot the ROC curves in Fig. 2 using and green (RF), violet (SecSVM), blue (SVM with ℓ_1), light blue (SVM with ℓ_2), orange (LR with ℓ_1), brown (LR with ℓ_2), solid lines. They clearly show the superiority of ModSec-Learn w.r.t. the respective ModSecurity counterpart regardless of the operating point, i.e., for any FPR value, the TPR of ModSec-Learn approaches is higher for all PLs greater than 1. 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, unlike the vanilla ModSecurity, all ModSec-Learn models achieve the best TPR for PL4 (even though the results for PL4 are slightly higher than those obtained for PL2). This result shows that, despite adding rules that may increase the FPR, machine learning can adjust their importance to improve the TPR/FPR trade-off.

Robustness against Adversarial SQLi. As shown in Fig. 2, the ModSec-Learn models suffer the presence of adversarial attacks, particularly with the WAF-A-MoLE dataset, but they still outperform the vanilla ModSecurity. Analyzing the results from the WAF-A-MoLE dataset, it is evident that, the best performance is achieved with PL4, except for the RF and SecSVM models. This may be since, in this case, the adversarial SQLi attacks are able to exploit the rules enabled by PL3 and PL4 to evade the model, by removing patterns that were considered important at training time. On the other hand, with the ModSec-Learn dataset, it is observed that starting from PL2, the detection capabilities exhibit a fairly consistent trend. While there is a slight deterioration, performance remains relatively stable across the different PLs. Among all evaluated models, SecSVM is the most robust, offering strong generalization and adversarial robustness. Even if RF was included to explore non linear alternatives, its performance is comparable to linear models like SecSVM, while worsening robustness. This confirms that linear models are sufficient to achieve excellent accuracy and are preferable given their robustness and transparency. In addition, linear models can be readily applied to the existing CRS system by updating the rule weights after model training.[10] This eases practical deployment while preserving interpretability of decisions — an important desideratum for web security.

Imposing Sparsity through Regularization. We now analyze the effects of regularization by examining whether it is possible to select a reduced set of CRS rules as features for

[10]https://owasp.org/www-project-waf-advanced-ruleset-management/

classification. We employ an ℓ_1 regularization term to impose sparsity on the trained models and assess its impact on the importance of each CRS rule in the classification process. Additionally, we compare these results with those obtained using ℓ_2 regularization, the default norm used by SVM and LR. Fig. 3 displays the distribution of rule weights for ModSec-Learn implemented with LR at PL4. We selected this PL to activate all CRS rules, providing a comprehensive overview of their impact. The blue and light red bars represent the weights calculated with ℓ_1 and ℓ_2 regularization, respectively, while the green bars represent the ModSecurity severity scores. Since the severity score ranges from 2 to 5, we normalized it using the minimum and maximum values of the LR weights. The results presented in Table III demonstrate that SVM and LR with ℓ_1 regularization can achieve the same performance as the counterpart with ℓ_2 regularization while utilizing fewer rules. Specifically, in the WAF-A-MoLE dataset, the SVM model employed 13 rules fewer than with the ℓ_2 norm, and the LR model 21 fewer than LR with ℓ_2. For the ModSec-Learn dataset, SVM used 14 fewer rules than SVM with ℓ_2 norm, while LR used 21 fewer. Additionally, it is important to note that, compared to the 62 total rules in CRS, linear models with ℓ_2 regularization, as well as RF, already reduce the number of rules used in the classification process compared to the vanilla ModSecurity. On average, these models use a maximum of 50 rules, effectively eliminating 12 rules deemed unnecessary for classification. The rules assigned a weight of 0 by the ML models are considered unnecessary for the classification task. Moreover, some rules may receive negative weights, suggesting that their presence might be more indicative of legitimate behavior rather than malicious activity. Applying this approach to the CRS introduces a more data-driven and less arbitrary method for selecting detection rules. Rather than rely on manual selection or a predefined set of rules that may not be optimal for the specific data being classified, ModSec-Learn enables automation of both rule selection and weight assignment, optimizing CRS's performance on the data.

D. Evaluation of ModSec-AdvLearn

Given the best results on PL*4* among all the PLs in terms of TPR/FPR, we select this configuration for re-training all ModSec-Learn models. We then evaluate the ModSec-AdvLearn against the `test` and `test-adv` sets of both datasets, and plot the results in Fig. 4. Also, we report the TPR of ModSec-AdvLearn at 1% FPR in the second-to-last column of Table III. Hereafter, we discuss the performance of ModSec-AdvLearn in comparison with ModSec-Learn. Overall, we observe that the robustness achieved by ModSec-AdvLearn clearly outperforms its non-hardened counterparts, i.e., ModSec-Learn. Finally, we analyze the weights and predictions of ModSec-AdvLearn, and we thoroughly explain its remarkable level of adversarial robustness.

Evaluation on Clean Samples. In the absence of attack, ModSec-AdvLearn has comparable performance to ModSec-Learn, except for SecSVM which shows an improvement in TPR on the ModSec-Learn dataset (cf. the violet solid lines in Fig. 4, *third row*). The reason is that SecSVM is retrained on less pessimistic attacks when considering problem-space AT, thereby yielding an improved robustness-accuracy tradeoff.

Robustness against Adversarial SQLi. Over the adversarial test set, ModSec-AdvLearn outperforms its non-hardened counterparts, especially when evaluated with the WAF-A-MoLE dataset, reaching thus higher robustness (see the dashed lines of Fig. 4). Looking at the results in detail, with the WAF-A-MoLE dataset, the re-trained models improve the average detection performance by 33% compared to their non-hardened versions. For the ModSec-Learn dataset, the improvement is less marked in some models, such as ModSec-AdvLearn LR, but it is still present. Moreover, considering the WAF-A-MoLE dataset for example, the best ModSec-AdvLearn (i.e., RF at PL4) is 85% more robust than the best vanilla ModSecurity (PL2). Of course ModSec-AdvLearn is still vulnerable to new adversarial examples optimized against it, but the decrement in performance is lower compared to the decrement caused by the non-hardened models.

Explaining Robustness of ModSec-AdvLearn. Here we explain why ModSec-AdvLearn achieves better robustness, focusing on the linear SVM model. First, we compute the *rule activation delta* as $\Delta \boldsymbol{a}_i = \boldsymbol{a}_i - \boldsymbol{a}'_i$. It captures the difference between the probability that rule *i* is activated by standard SQLi attacks ($\boldsymbol{a}_i$) and by their adversarial counterparts ($\boldsymbol{a}'_i$). If $\Delta \boldsymbol{a}_i$ is positive (negative), it means that WAF-A-MoLE is bypassing (activating) rule *i*, and if it is zero it means that WAF-A-MoLE does not affect rule *i*. Second, as we are considering a linear model, we also inspect its feature weights and observe how they change when the same model is retrained on adversarial SQLi queries. Within this scenario, we analyze how each rule is affected by the adversarial SQLi attacks generated through WAF-A-MoLE (Fig. 5), as well as how differently the baseline ModSec-Learn SVM and the ModSec-AdvLearn SVM compute weights for each rule (Fig. 5). In Fig. 5 we plot the probability that a rule is active, and we display how different the distributions induced by the malicious (cyan) and adversarial (yellow) SQLi queries are. The rules are sorted by their value of $\Delta \boldsymbol{a}_i$, and grouped into three classes (separated by vertical black lines): rules evaded by WAF-A-MoLE (left), rules that WAF-A-MoLE is unable to bypass (center), and rules that are triggered only by adversarial attacks as a side effect (right). The same order is also used for the weights of the ModSec-Learn SVM and the ModSec-AdvLearn SVM shown in Fig. 5. We can observe that more than one-third of the rules are exploited by WAF-A-MoLE to avoid detection, as the first group has a drop in the probability of being active. This is also confirmed by Fig. 5, where we can see that most of the positive weights (i.e., the ones that increase the scores towards the malicious class) assigned by ModSec-Learn (cyan) are all concentrated in the first group, which is exactly the one leveraged by adversarial attacks. Conversely, ModSec-AdvLearn (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 easier to detect. Of course, in this analysis,

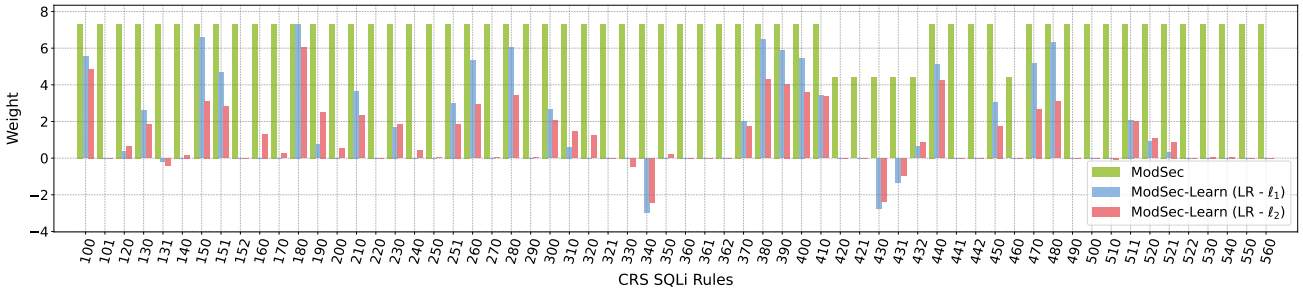

Fig. 3: Weight values learned at PL4 by ModSec-Learn LR-ℓ_1 (blue) and ModSec-Learn LR-ℓ_2 (light red), and the weights used by ModSecurity (green). Rules are expressed as the last three digits of their IDs (all starting with 942).

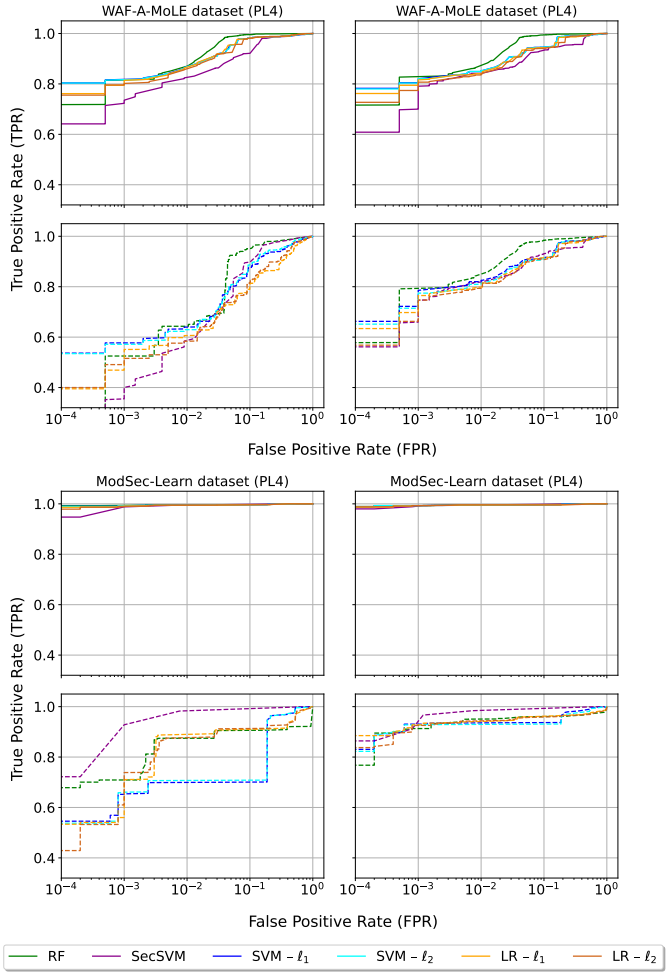

Fig. 4: ROC curves of ModSec-Learn/AdvLearn (*first/second column*) on `test`/`test-adv` (*solid/dashed lines*), for WAF-A-MoLE/ModSec-Learn (*top/bottom*) datasets.

we focus on a linear model, which, as shown in Sect. IV-D is still vulnerable to adversarial attacks. Indeed, by looking at Fig. 5, ModSec-AdvLearn attributes negative weights (i.e., those that decrease the score towards the benign class) to some features of the first block, hence leaving the ability to WAF-A-MoLE to find some successful adversarial SQLi queries. On the other hand, as shown in Fig. 4, for the most of the cases ModSec-AdvLearn RF is more robust than the linear models counterpart as its non-linearity exploits relationships among different rules, forcing the attacker to manipulate more rules in a consistent manner to bypass detection.

V. Related Work

In this section, we briefly review related work analyzing the performance and robustness of ModSecurity and the CRS, and conclude by discussing the main differences of our current work with our preliminary results in [11].

ModSecurity and CRS. Previous work has considered the impact of different types of web security threats on ModSecurity when using the CRS configurations [24, 25]. However, unlike ours, a very limited number of attack samples is normally used (e.g., [24] uses only 27 samples), without providing any detailed investigation of the trade-off between TPR and FPR. Other approaches [26, 27] have applied ML to detect web threats and only used ModSecurity as a baseline for comparison, without even evaluating adversarial robustness.

Adversarial SQLi. Other work has considered adversarial SQLi attacks against ModSecurity by proposing different approaches based on ML [5], Reinforcement Learning (RL) [13, 28, 29], fuzzing techniques [10, 30], and heuristic search algorithm like Monte-Carlo tree search [31]. However, the reported results are partial (e.g., [13, 31] just limit the analysis to the default PL1) and do not explain precisely why ModSecurity is failing and how it could be improved. To the best of our knowledge, no prior research has conducted a comprehensive analysis of the CRS for ModSecurity as we have done in this work. Additionally, no previous studies have explored the potential of adversarial training in this domain, making us the first to propose a robust ML methodology for effectively enhancing the robustness of WAFs.

ModSec-Learn. With respect to our work introducing ModSec-Learn in [11], we have extended here our approach as follows: (i) we have examined the impact of adversarial attacks on the CRS within the SQL domain; (ii) we have increased the robustness of our approach by developing a novel adversarial training procedure (ModSec-AdvLearn); (iii) we have investigated whether strongly-regularized models could withstand adversarial SQLi attacks, devising a novel version of SecSVM;

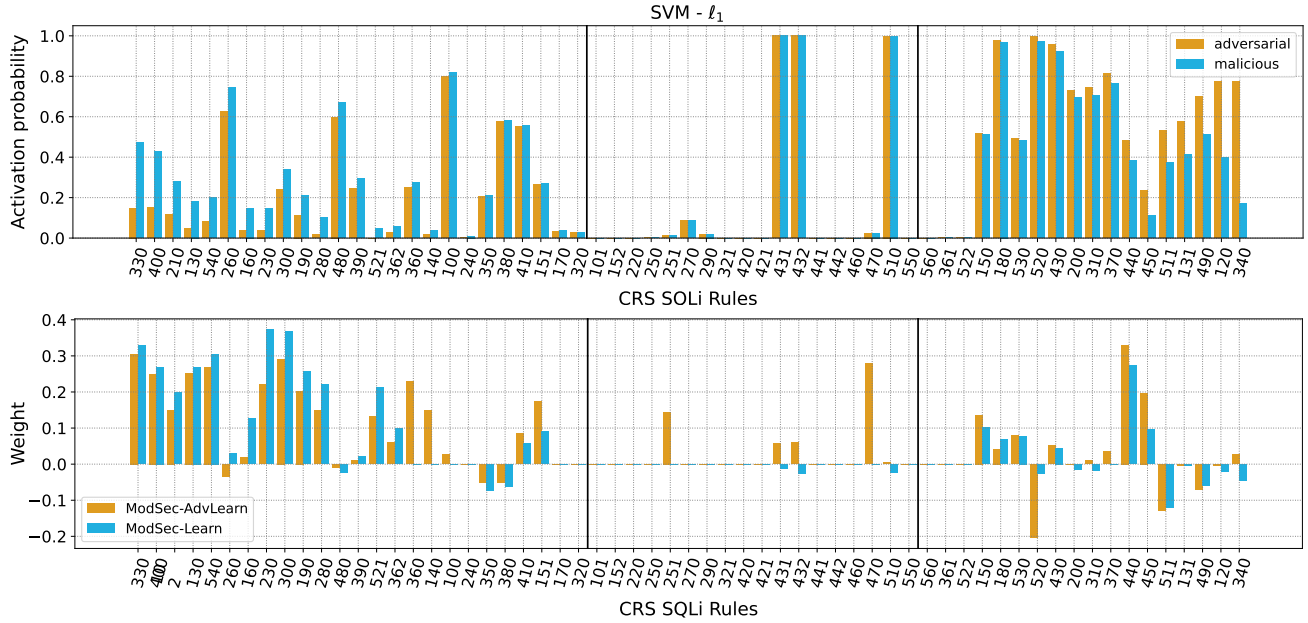

Fig. 5: *Top:* Activation probability of CRS rules (expressed using the last three digits of their IDs, which always starts with 942) on malicious/adversarial (cyan/orange) SQLi samples optimized against ModSec-Learn SVM-ℓ_1 on the WAF-A-MoLE dataset. *Bottom:* Rule weights learned by ModSec-Learn/ModSec-AdvLearn SVM-ℓ_1 (cyan/orange) on the WAF-A-MoLE dataset.

(iv) we have analyzed how the baseline and ModSec-AdvLearn compute weights for each rule, highlighting which rules are more robust to perturbations; and (v) we have included an additional dataset [10] in our experiments.

VI. Conclusions and Future Work

In this work, we proposed ModSec-AdvLearn, a novel methodology for training ML classifiers using the CRS rules as input features. This allows learning how to optimally tune the severity levels (i.e., the weights) of the CRS rules, yielding an improved trade-off between detection and false positive rates. Furthermore, our approach relies upon a novel problem-space adversarial training procedure that incorporates knowledge of state-of-the-art SQLi manipulations to counter the presence of adversarial SQLi attacks. Among the main findings, we show that ModSec-AdvLearn improves the detection rate of the vanilla ModSecurity by 30%, while removing 50% of the CRS rules through embedded feature selection with ℓ_1 regularization. It also improves adversarial robustness up to 85% via robust *linear* models, without hindering interpretability of decisions and providing ease of integration with the current CRS implementations. We can thus 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, highlighting novel, promising directions towards designing robust machine learning models for cybersecurity-related applications.

We foresee several other promising avenues for advancing our work. First, although in this work we only target SQLi attacks, our methodology is general enough to tackle other web threats like cross-site scripting (XSS). In this direction, future work could also explore the integration of automated pentesting tools that leverage large language models [32], to extend the capabilities of WAF-A-MoLE. Second, we also see future developments in evaluating other state-of-the-art ML-based WAFs. Indeed, we think that the same results can also be obtained on more advanced models such as Convolutional Neural Networks (CNN) [33], as well as on different feature representation approaches [27, 34]. This is also true for commercial WAFs. To this end, an interesting future extension of this work is to evaluate them in terms of transferability [35] of adversarial SQLi attacks optimized on ModSecurity.

Acknowledgments

This research has been partly supported by the TESTABLE project, funded by the EU H2020 research and innovation program (grant no. 101019206); the ELSA project, funded by the Horizon Europe research and innovation program (grant no. 101070617); projects FAIR (PE00000013) and SERICS (PE00000014) under the NRRP MUR program funded by the EU – NGEU. This work was carried out while C. Scano was enrolled in the Italian National Doctorate on AI run by the Sapienza University of Rome in collaboration with the University of Cagliari.

References

[1] O. B. Fredj, O. Cheikhrouhou, M. Krichen, H. Hamam, and A. Derhab, "An OWASP top ten driven survey on web application protection methods," in *15th Int'l Conf. Risks & Sec. of Internet Sys. (CRiSIS)*, p. 235–252, 2020.

[2] W. G. J. Halfond and A. Orso, "Preventing sql injection attacks using amnesia," in *Proc. of the 28th Int. Conf. on Software Engineering (ICSE)*, p. 795–798, 2006.

[3] A. Joshi and V. Geetha, "SQL injection detection using machine learning," in *2014 Int. Conf. Control, Instrumentation, Comm. Comput. Tech.*, pp. 1111–1115, 2014.

[4] D. Appelt, A. Panichella, and L. Briand, "Automatically repairing web application firewalls based on successful sql injection attacks," in *2017 IEEE 28th Int. Symp. on Software Reliability Eng. (ISSRE)*, pp. 339–350, 2017.

[5] D. Appelt, C. D. Nguyen, A. Panichella, and L. C. Briand, "A machine-learning-driven evolutionary approach for testing web application firewalls," *IEEE Trans. on Reliability*, vol. 67, no. 3, pp. 733–757, 2018.

[6] OWASP Foundation Inc., "OWASP top 10," 2021. Available online. Accessed on 22 February 2023.

[7] S. Applebaum, T. Gaber, and A. Ahmed, "Signature-based and machine-learning-based web application firewalls: A short survey," *Procedia Computer Science*, vol. 189, pp. 359–367, 2021.

[8] OWASP Foundation Inc., "OWASP core rule set." https://coreruleset.org, 2024. Accessed on 20th January 2024.

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

[10] L. Demetrio, A. Valenza, G. Costa, and G. Lagorio, "Waf-a-mole: Evading web application firewalls through adversarial machine learning," in *35th Annual ACM Symp. on Applied Computing (SAC)*, p. 1745–1752, 2020.

[11] C. Scano, G. Floris, B. Montaruli, L. Demetrio, A. Valenza, L. Compagna, D. Ariu, L. Piras, D. Balzarotti, and B. Biggio, "Modsec-learn: Boosting modsecurity with machine learning," in *Int'l Symp. Distributed Comput. and AI*, pp. 23–33, Springer, 2024.

[12] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler, "Mutation-based fuzzing," in *The Fuzzing Book*, CISPA Helmholtz Center for Inform. Sec., 2023.

[13] M. Hemmati and M. A. Hadavi, "Bypassing web application firewalls using deep reinforcement learning," in *18th Int. ISC Conf. Inform. Sec. Crypt.*, pp. 35–41, 2021.

[14] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *IEEE Symp. on Security and Privacy (SP)*, pp. 1332–1349, IEEE, 2020.

[15] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, 1995.

[16] C. M. Bishop and N. M. Nasrabadi, *Linear Models for Classification*, vol. 4. Springer, 2006.

[17] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[18] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.

[19] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[20] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Trans. on Dependable and Secure Computing*, vol. 16, pp. 711–724, 2019.

[21] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Functionality-preserving black-box optimization of adversarial windows malware," *IEEE Trans. on Inform. Forensics and Sec.*, vol. 16, pp. 3469–3478, 2021.

[22] H. Xu, C. Caramanis, and S. Mannor, "Robustness and regularization of support vector machines.," *Journal of machine learning research*, vol. 10, no. 7, 2009.

[23] I. Corona, B. Biggio, M. Contini, L. Piras, R. Corda, M. Mereu, G. Mureddu, D. Ariu, and F. Roli, "Deltaphish: Detecting phishing webpages in compromised websites," in *ESORICS 2017*, pp. 370–388, 2017.

[24] J. J. Singh, H. Samuel, and P. Zavarsky, "Impact of paranoia levels on the effectiveness of the modsecurity web application firewall," in *1st Int. Conf. on Data Intelligence and Security (ICDIS)*, pp. 141–144, 2018.

[25] T. D. Sobola, P. Zavarsky, and S. Butakov, "Experimental study of modsecurity web application firewalls," in *IEEE-BigDataSecurity, HPSC and IDS*, pp. 209–213, 2020.

[26] G. Betarte, Á. Pardo, and R. Martínez, "Web application attacks detection using machine learning techniques," in *17th IEEE Int'l Conference on Machine Learning and Applications (ICMLA)*, pp. 1065–1072, IEEE, 2018.

[27] N. Montes, G. Betarte, R. Martínez, and A. Pardo, "Web application attacks detection using deep learning," in *25th Progress in Patt. Rec., Image Analysis, Computer Vision, and Applications*, pp. 227–236, Springer, 2021.

[28] X. Wang and H. HU, "Evading web application firewalls with reinforcement learning," 2020.

[29] Y. Guan, J. He, T. Li, H. Zhao, and B. Ma, "Ssqli: A black-box adversarial attack method for sql injection based on reinforcement learning," *Future Internet*, vol. 15, no. 4, p. 133, 2023.

[30] K. Li, H. Yang, and W. Visser, "DaNuoYi: Evolutionary multi-task injection testing on web application firewalls," *IEEE Trans. on Software Engineering*, 2023.

[31] Z. Qu, X. Ling, and C. Wu, *AutoSpear: Towards Automatically Bypassing and Inspecting Web Application Firewalls*. Black Hat Asia, 2022.

[32] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *33rd USENIX Security Symp.*, pp. 847–864, 2024.

[33] A. Luo, W. Huang, and W. Fan, "A cnn-based approach to the detection of sql injection attacks," in *IEEE/ACIS 18th Int. Conf. on Computer and Information Science (ICIS)*, pp. 320–324, IEEE, 2019.

[34] D. Kar, S. Panigrahi, and S. Sundararajan, "Sqligot: Detecting sql injection attacks using graph of tokens and svm," *Computers & Security*, vol. 60, pp. 206–225, 2016.

[35] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *28th USENIX Security Symp.*, pp. 321–338, 2019.

Giuseppe Floris received his BSc degree in Electrical, Eletronical and Computer Engineering in 2021, and his MSc degree in Computer Engineering, Cybersecurity, and Artificial Intelligence with honors in September 2023 from the University of Cagliari. He is currently a Ph.D. student in electronics and computer engineering at the University of Cagliari. His research focuses on Adversarial Machine Learning and its applications in the cybersecurity domain.

Christian Scano received his BSc degree in Computer Science in 2021 and his MSc degree in Computer Engineering, Cybersecurity, and Artificial Intelligence with honors in September 2024 from the University of Cagliari. He is currently enrolled in the national PhD in Artificial Intelligence and Computer Security at University of Cagliari and Sapienza University of Rome. His research focuses on Web Application Security and Machine Learning.

Biagio Montaruli received his B.Sc. and M.Sc. degrees in computer engineering from the Polytechnic University of Bari (Bari), in 2018 and 2021, respectively. He is currently a Ph.D. candidate in artificial intelligence and computer security at EURECOM (France). His research focuses on adversarial machine learning, with strong focus on its application in the cyber-security domain.

Luca Demetrio (MSc 2017, PhD 2021) is an Assistant Professor at the University of Genoa, investigating the security of Windows malware detectors implemented with Machine Learning techniques. He is part of the development team of SecML, and the maintainer of SecML Malware, a Python library for creating adversarial Windows malware.

Andrea Valenza is an Application Security Engineer at Prima Assicurazioni. He received his PhD from the University of Genova, with a thesis focused on novel vulnerabilities, including counter-attacking automatic security scanners. His current research interest is automated security testing, with a focus on improving (and bypassing) regex-based validation, and detection of vulnerabilities via static analysis.

Luca Compagna works at Endor Labs, contributing to the software security analysis research area. He received his Ph.D. in Computer Science jointly from the U. of Genova and U. of Edinburgh, working on security protocols analysis. His areas of interests include security testing, security engineering, automated reasoning, and their application to the modeling and analysis of industrial relevant scenarios. After some work on DAST techniques for cross domain web-based scenarios and CSRF experiments, he recently focused his attention to static analysis and to the security testing of AI-based components.

Davide Ariu received a PhD in electronics and computer engineering from the University of Cagliari. He is co-founder and CEO of Pluribus One (http://www.pluribus-one.it) and co-chair of the OWASP (http://owasp.org) Italy Chapter.

Luca Piras is the Operation Manager and Co-founder of Pluribus One. He received his MSc Degree in Electronic Engineering in 2007 and his Doctor Europaeus and PhD Degree in Computer Engineering in 2011, both from the University of Cagliari. His expertise lies in Computer Vision, Pattern Recognition, and Machine Learning. His research has been published in several international, peer-reviewed journals and conferences. At Pluribus One, he is responsible for several EU-funded R&D projects and is a member of the OWASP Foundation.

Davide Balzarotti is a full Professor and head of the Digital Security Department at EURECOM. His research interests include most aspects of software and system security and in particular the areas of binary and malware analysis, reverse engineering, computer forensics, and web security. Davide authored more than 100 publications in leading conferences and journals. He has been the Program co-Chair of Usenix Security 2024, ACSAC 2017, RAID 2012, and Eurosec 2014. He received an ERC Consolidator and an ERC PoC Grant for his research in the analysis of compromised systems. Davide is also a member of the "Order of the Overflow" team, which organized the DEF CON CTF competition between 2018 and 2021.

Battista Biggio (MSc 2006, PhD 2010) is Full Professor of Computer Engineering at the University of Cagliari, Italy. He has provided pioneering contributions to machine learning security. His paper "Poisoning Attacks against Support Vector Machines" won the prestigious 2022 ICML Test of Time Award. He chaired IAPR TC1 (2016-2020), and served as Associate Editor for IEEE TNNLS and IEEE CIM. He is now Associate Editor-in-Chief for Pattern Recognition and serves as Area Chair for NeurIPS and IEEE Symp. SP. He is Fellow of IEEE and AAIA, ACM Senior Member, and Member of IAPR, AAAI, and ELLIS.