WAF-A-MoLE: An adversarial tool for assessing ML-based WAFs

Andrea Valenza, Luca Demetrio, Gabriele Costa, Giovanni Lagorio

University of Genova

IMT Lucca

Abstract

Web Application Firewalls (WAFs) are plug-and-play security gateways. They promise to enhance the security of a (potentially vulnerable) system with minimal cost and configuration. In recent years, machine learning-based WAFs are catching up with traditional, signature-based ones. They are competitive because they do not require predefined rules. Instead, they infer their rules through a learning process.

In this paper we present WAF-A-MoLE, a WAF breaching tool. It uses guided mutational-based fuzzing to generate adversarial examples. The main applications include WAF (*i*) penetration testing, (*ii*) benchmarking and (*iii*) hardening.

Keywords: web application firewall, SQL injection, penetration testing, adversarial machine learning

1. Motivation and significance

Many modern systems expose some web services over the Internet. When they are vulnerable, the security of the entire system is compromised. A widespread mitigation technique is to deploy a *Web Application Firewall* (WAF). A WAF attempts to detect malicious incoming payloads and drop them before they reach their target. Clearly, the ability to craft payloads that pass undetected gives a tremendous advantage to attackers.

WAFs are traditionally signature-based, with a predefined set of rules for attack identification. However, this approach lacks generality and requires a significant effort to maintain the rule set. For this reason, researchers have recently considered the adoption of *machine learning* (ML). ML-based WAFs overcome some of the limitations of traditional WAFs. Their detection rules are inferred from a set of payloads through a training process.

On the flip side, an aware attacker can take advantage of biases in the training set. For instance, the training set might miss some relevant payloads, so causing blind spots in the classification space. *Adversarial machine learning* [1] studies this phenomenon, i.e., how to craft misleading payloads, a.k.a. *adversarial examples*.

In this paper we present WAF-A-MoLE, a tool to generate adversarial examples for ML-based WAFs. In particular, the current version of the tool focuses on SQL injection (SQLi) attacks. WAF-A-MoLE starts from a payload and mutates it to bypass a target WAF. The tool relies on a set of semantics preserving mutation operators. The mutation process is guided by the classification confidence of the target WAF.

This paper is structured as follows. Section 2 describes WAF-A-MoLE and its main functionalities. Section 3 shows an example of how WAF-A-MoLE bypasses a target toy WAF. Section 4 highlights the impact of WAF-A-MoLE on the security community. Finally, Section 6 concludes the paper.

2. Software description

WAF-A-MoLE uses a *mutation-based fuzz testing* [2] methodology to create attacks that bypass a target WAF. More precisely, WAF-A-MoLE uses the classification score of the WAF to guide the fuzzing process by prioritizing the most promising payloads. We describe the overall architecture and main functionalities of WAF-A-MoLE in the next section.

2.1. Software Architecture

WAF-A-MoLE is both a library and a command line tool (obtained by means of *Click*[1] decorators on the main exported functions) implemented in Python 3. Figure 1 shows the main workflow of WAF-A-MoLE. Briefly, the orchestrator (not shown in the figures) takes an initial payload p_0, that the target WAF detects as malicious with a confidence score $\sigma_0 \in [0, 1]$, and inserts it in the initially empty payload `Pool`. The `Pool`, in turn, manages a priority queue, storing payloads in decreasing ordered of their scores.

During each iteration, the head of the queue p_n is picked from the `Pool`, and passed to the `Fuzzer`, which randomly mutates p_n into p_{n+1} by applying some mutation operators (see Section 2.2). Then, p_{n+1} is submitted to the target WAF for classification. Since we do not expect WAFs to adhere to any specific interface, WAF-A-MoLE uses specific *adapters* that ensure

[1] `https://click.palletsprojects.com/en/7.x/`

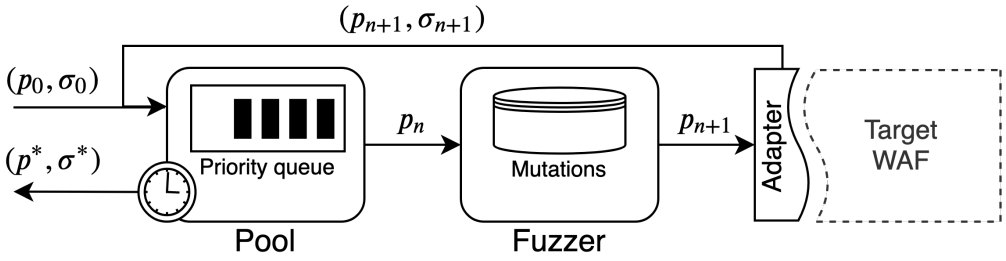

Figure 1: Main workflow of WAF-A-MoLE.

compatibility. The `Adapter` then returns the classification score σ_{n+1} of p_{n+1}, which is fed back into the `Pool`.

This cycle finishes successfully whenever the best confidence score σ^* is less than a given threshold, or is interrupted, returning the best pair (p^*, σ^*) found so far, because the number of iterations, queue sizes or computation time reach their maximum values.

In order to apply WAF-A-MoLE to different machine learning models, without incurring into a tight coupling, we designed an interface, modeled in Python as an abstract class called `Model`, which generalizes the behaviour of those models. This class provides two abstract methods, `classify` and `extract_features`, that need to be instantiated for each kind of model. That is, since, no real model matches exactly our interface, for each of them we need an adapter class that wraps the target model and exports our `Model` interface (see Figure 1).

We provide many wrappers out of the box, which are the ones that we used for running our experiments. They also serve as examples of how to implement new wrappers. In particular, we offer wrappers for two well-known frameworks: `SklearnModelWrapper` for *scikit-learn*[2], and `KerasModelWrapper` for *keras*[3].

2.2. Software Functionalities

As discussed in Section 2.1, the main components of WAF-A-MoLE are `Pool` and `Fuzzer`. The former handles the priority queue and termination conditions. Although they raise some technical issues (e.g., due to the memory usage of large data structures), these aspects belong to the generic context of program optimization. Instead, `Fuzzer` requires more attention.

[2] `https://scikit-learn.org/stable/index.html`
[3] `https://keras.io/`

Following the mutational fuzzing approach, `Fuzzer` applies a number of mutation operators. Mutation operators act on the string representation, and they modify the syntax of a payload without altering its semantics. Since we focus on SQLi, the currently implemented mutation operators work on SQL. We describe them below.

CS. The *Case Swapping* operator randomly changes the capitalization of keywords in a query (e.g., `Select` to `sELecT`).

WS. Whitespace Substitution leverages the equivalence between several alternative separators (whitespaces) between query tokens. For instance, alternative whitespaces include `\n` (line feed), `\r` (carriage return) and `\t` (horizontal tab). Each whitespace can be replaced by an arbitrary, non-empty sequence of whitespaces (e.g., `1 = 1` may become `1\n\t=\r 1`).

CI. The *Comment Injection* operator randomly adds an inline comment (`/*...*/`) between two query tokens. As whitespaces, inline comments act as token separators (e.g., modifying `1 = 1` to `1/**/= 1`).

CR. The *Comment Rewriting* operator randomly modifies the content of a comment. This both applies to inline and trailing (`#` and `-- `) comments (e.g., `/*abc*/` may become `/*xy*/`).

IE. The *Integer Encoding* operator modifies the representation of numerical constants. This includes alternative base representations, e.g., from decimal to hexadecimal (e.g., `0x2` for `2`), as well as statement nesting (e.g., `(SELECT 42)` for `42`).

OS. Some operators can be replaced by others that behave in the same way. For instance, `1 = 1` (equality check) is simulated by `1 LIKE 1` (pattern matching).[4] We call this mutation *Operator Swapping*.

LI. A *Logical Invariant* operator modifies a boolean expression by adding terms that preserve its semantics (e.g., `1 = 1` is equivalent to `1 = 1 AND True`).

In defining our mutation operators, we took inspiration from some malicious payload samples such as those listed by Awesome WAF[5] and Payloads All The Things[6]. All in all, our operators generalize the techniques for producing payloads similar to those mentioned above.

3. Illustrative Example

In this section, we provide a demonstration of WAF-A-MoLE applied to a toy WAF. The toy WAF assigns a score to a (non-empty) payload p through

[4]Notice that, in general, `LIKE` is not equivalent to `=`. However, the equivalence holds when restricting to specific domains, e.g., comparison between integer constants.

[5]`https://github.com/0xInfection/Awesome-WAF`

[6]`https://github.com/swisskyrepo/PayloadsAllTheThings`

Op.	Mutant	σ	Op.	Mutant	σ
WS	`admin'\t OR\n 1=1#`	0.75	CI	`admin' OR 1=1/**/#`	0.67
CR	`admin' OR 1=1#abcde`	0.63	IE	`admin' OR 0x1=1#`	0.75
OS	`admin' OR 1 LIKE 1#`	0.79	LI	`admin' OR 1=1 AND 0<1#`	0.68

Table 1: Mutants and classification scores for the toy WAF.

the following function:

$$\sigma(p) = \min\left\{1, \frac{3 \cdot S(p)}{T(p)}\right\}$$

where $S(p)$ is the number of special characters `'`, `=` and ␣ (a single white space) and $T(p)$ is the total number of characters in p. For instance

$\sigma(\texttt{admin' OR 1=1\#}) = \min\left\{1, \frac{3 \cdot 4}{14}\right\} \approx 0.86$ $\sigma(\texttt{admin}) = \min\left\{1, \frac{3 \cdot 0}{5}\right\} = 0$

Assuming the acceptance threshold of the toy WAF to be 1/2 (that is p is rejected when $\sigma(p) > 1/2$) the first payload above is rejected. Table 1 reports the σ values of some mutants obtained through the application of the operators of Section 2.2.

Let assume that WAF-A-MoLE generated the payloads of Table 1. They are ordered by their σ values and inserted in the payload pool. Then, the next mutation round starts from the payload with the lowest σ, i.e., the one generated by CR.

4. Impact

The impact of our tool on penetration testing activities is straightforward. Penetration testers can use WAF-A-MoLE as an off-the-shelf utility to craft attacks. For instance, we used WAF-A-MoLE against 9 instances of WAFs taken from literature to assess its effectiveness. The results are promising, and we provide an excerpt in Figure 2. Briefly, WAF-A-MoLE rapidly decreases the confidence score of the considered WAFs. The plot on the left shows how confidence decreases with the number of applied mutations. The plot on the right shows how it decreases over time, with a logarithmic scale. Since our approach is inherently stochastic, we ran our tool multiple times and chose the best run (i.e., the one that reached the threshold in the fewest mutation rounds) for each classifier.

For our analysis, we considered different classifiers:

1. WAF-Brain[7], a deep neural network trained on raw characters contain-

[7] `https://github.com/BBVA/waf-brain`

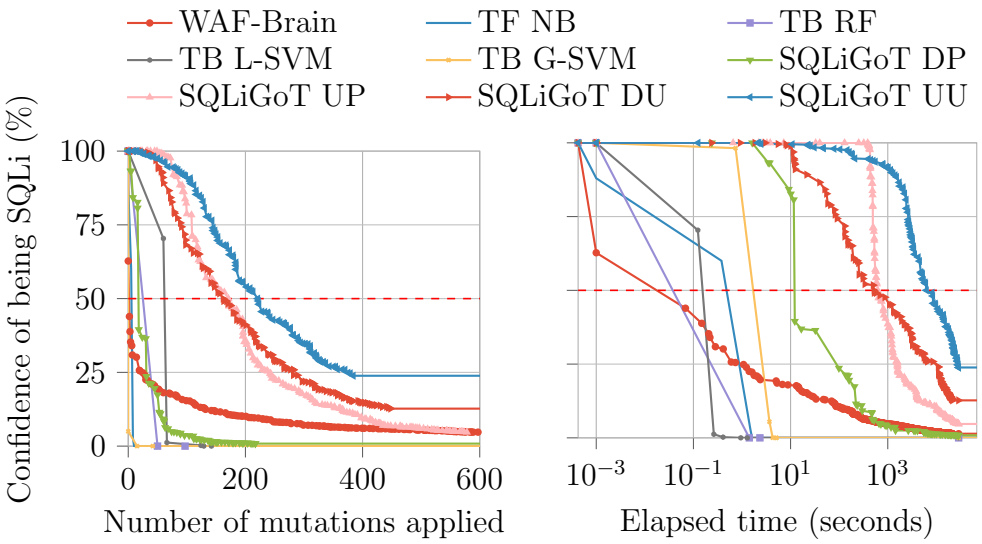

Figure 2: WAF-A-MoLE applied to the `admin' OR 1=1#` payload

ing legitimate and malicious payloads,

2. Token-based models [3, 4], implemented using different algorithms, built on a histogram of tokens extracted from the queries, and
3. SQLiGoT [5], a Support Vector Machine (SVM) [6] classifier that reasons on top of a graph structure extracted from input queries.

WAF-A-MoLE bypasses WAF-Brain in 7 mutation rounds. Some Token-based approaches performed worse than WAF-Brain, Token-based Random Forest and Gaussian SVM variants reached the threshold in respectively 2 and 3 mutation rounds. The Linear SVM and Naive Bayes variants performed better, with 24 and 46 rounds. SQLiGoT proved to be the most resilient: one of its variants, namely the Undirected Proportional one, reached the threshold after 134 rounds, and the Undirected Unproportional version needed 290 steps.

The full details about our experiments can be found at `https://github.com/AvalZ/waf-a-mole`.

The by-product of WAF-A-MoLE are *adversarial examples* [7] for the target WAF. Adversarial examples are a cornerstone in *adversarial training* [7, 8, 9]. ML-based WAFs are trained on datasets that very rarely characterize the entire classification domain. Developers can use the adversarial examples to re-train their WAF, covering areas originally not included in the training dataset [7, 8, 9]. Intuitively, training the classifier with regular data and adversarial examples leads to a more robust model. On the other hand, an adversarially trained model loses accuracy w.r.t. its standard counterpart,

as the problem to be learned is more complex. Although the methodology of [7, 8, 9] is not applied to our working domanin, i.e., SQL payloads, we believe that a similar technique can be ported in our context. In this way, WAF-A-MoLE can support the WAF hardening process.

Using WAF-A-MoLE, we showed that ML-based WAFs are vulnerable to adversarial attacks. We believe that the main reason is the gap between the syntax level (of the WAFs classification) and the semantic level (of the vulnerable application). This observation pushes forward an open research question: to what extent (syntax-based) WAFs prevent injection attacks? WAF-A-MoLE candidates to be a valuable assessment tool to support this research line.

5. Related work

Machine learning based WAFs. Ceccato et al. [10] propose a clustering method for detecting SQL injection attacks against a victim service. The algorithm learns from the queries that are processed inside the web application under analysis using an unsupervised one-class learning approach, namely K-medoids [11]. New samples are compared to the closest medoid and flagged as malicious if their edit distance w.r.t. the chosen medoid is higher than the diameter of the cluster. Kar et al. [5] develop *SQLiGoT*, an SVM that express queries as graphs of tokens, whose edges represent the adjacency of SQL-tokens. Pinzon et al. [12] explore two different directions: visualization and detection, achieved by a multi-agent system called *idMAS-SQL*. To tackle the task of detecting SQL injection attacks, the authors set up two different classifiers, namely a Neural Network and an SVM. Makiou et al. [4] develop a hybrid approach that uses both machine learning techniques and pattern matching against a known dataset of attacks. The learning algorithm used for detecting injections is a Naive Bayes [13]. They look for different 45 tokens inside the input query, chosen by domain experts. Similarly, Joshi et al. [3] use a Naive Bayes classifier that, given a SQL query as input, extracts syntactic tokens using spaces as separators. The algorithm produces a feature vector that counts how many instances of a particular word occurs in the input query. The vocabulary of all the possible observable tokens is set a priori. Komiya et al. [14] propose a survey of different machine learning algorithms for SQL injection attack detection.

Adversarial ML attacks. Among all the techniques proposed in the state-of-the-art that leverage on white-box gradient techniques [7, 15, 16], we focus on black-box attacks, as they are similar to our method. Ilyas et al. [17] use the Natural Evolution Strategy (NES) [18] to guide the creation of adversarial

examples against well-known image classifiers. Xu et al.[19] propose a genetic algorithm that automatically learns which mutations should be applied to PDF malware to bypass a target classifier. Anderson et al. [20] train an agent to learn the best sequences of mutations applied to Windows malware to fool a target classifier. Chen et al. [21] estimate the target function's boundary locally around a particular input. Then, they guide the generation of adversarial examples by computing an approximated gradient using the values obtained from the target classifier in that local region.

6. Conclusions

In this paper we presented WAF-A-MoLE, a guided mutational fuzzing tool to generate adversarial examples for ML-based WAFs. The tool has several possible applications, the main one being the security assessment of the WAFs.

There are numerous future directions for this research line. In particular, there are three that we consider of primary importance. In the first place, we plan to apply WAF-A-MoLE to commercial WAFs. The main difficulty is that vendors usually do not share details about the internals of their products. Hence, this direction requires establishing an agreement with vendors. Secondly, we aim at extending our approach to deal with hybrid WAFs that also consider payload signatures. For both commercial and hybrid WAFs, we would also like to explore the possibility of integrating our tool in a retraining process for ML classifiers. Finally, we are interested in finding new mutation operators as well as investigating their effectiveness when applied alone or combined with others.

Acknowledgements

This work was partially funded by the Horizon 2020 project "Strategic Programs for Advanced Research and Technology in Europe" (SPARTA).

[1] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, J. D. Tygar, Adversarial machine learning, in: Proceedings of the 4th ACM workshop on Security and artificial intelligence, ACM, 2011, pp. 43–58.

[2] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, C. Holler, Mutation-based fuzzing, in: Generating Software Tests, Saarland University, 2019, retrieved 2019-05-21 19:57:59+02:00.
URL `https://www.fuzzingbook.org/html/MutationFuzzer.html`

[3] A. Joshi, V. Geetha, Sql injection detection using machine learning, in: 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), IEEE, 2014, pp. 1111–1115.

[4] A. Makiou, Y. Begriche, A. Serhrouchni, Improving web application firewalls to detect advanced sql injection attacks, in: 2014 10th International Conference on Information Assurance and Security, IEEE, 2014, pp. 35–40.

[5] D. Kar, S. Panigrahi, S. Sundararajan, Sqligot: Detecting sql injection attacks using graph of tokens and svm, Computers & Security 60 (2016) 206–225.

[6] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (3) (1995) 273–297.

[7] I. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: International Conference on Learning Representations, 2015.
URL http://arxiv.org/abs/1412.6572

[8] K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. McDaniel, Adversarial examples for malware detection, in: European Symposium on Research in Computer Security, Springer, 2017, pp. 62–79.

[9] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, Sixth International Conference on Learning Representations (ICLR).

[10] M. Ceccato, C. D. Nguyen, D. Appelt, L. C. Briand, Sofia: an automated security oracle for black-box testing of sql-injection vulnerabilities, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, 2016, pp. 167–177.

[11] L. K. P. J. RDUSSEEUN, Clustering by means of medoids.

[12] C. I. Pinzon, J. F. De Paz, A. Herrero, E. Corchado, J. Bajo, J. M. Corchado, idmas-sql: intrusion detection based on mas to detect and block sql injection through data mining, Information Sciences 231 (2013) 15–31.

[13] M. E. Maron, Automatic indexing: an experimental inquiry, Journal of the ACM (JACM) 8 (3) (1961) 404–417.

[14] R. Komiya, I. Paik, M. Hisada, Classification of malicious web code by machine learning, in: 2011 3rd International Conference on Awareness Science and Technology (iCAST), IEEE, 2011, pp. 406–411.

[15] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, A. Swami, The limitations of deep learning in adversarial settings, in: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2016, pp. 372–387.

[16] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: 2017 IEEE Symposium on Security and Privacy (SP), IEEE, 2017, pp. 39–57.

[17] A. Ilyas, L. Engstrom, A. Athalye, J. Lin, Black-box adversarial attacks with limited queries and information, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, 2018.
URL `https://arxiv.org/abs/1804.08598`

[18] D. Wierstra, T. Schaul, J. Peters, J. Schmidhuber, Natural evolution strategies, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 3381–3387.

[19] W. Xu, Y. Qi, D. Evans, Automatically evading classifiers, in: Proceedings of the 2016 Network and Distributed Systems Symposium, 2016, pp. 21–24.

[20] H. S. Anderson, A. Kharkar, B. Filar, P. Roth, Evading machine learning malware detection, Black Hat.

[21] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, C.-J. Hsieh, Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models, in: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, ACM, 2017, pp. 15–26.

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.0.0
C2	Permanent link to code/repository used for this code version	`https://github.com/AvalZ/waf-a-mole`
C3	Legal Code License	MIT
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python 3
C6	Compilation requirements, operating environments & dependencies	Click
C7	If available Link to developer documentation/manual	`https://waf-a-mole.readthedocs.io/en/latest/`
C8	Support email for questions	andrea.valenza@dibris.unige.it, luca.demetrio@dibris.unige.it

Table 2: Code metadata