CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Open Repository and Bibliography - Luxembourg

Automatically Repairing Web Application Firewalls Based on Successful SQL Injection Attacks

Dennis Appelt, Annibale Panichella, Lionel Briand
Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg

***Abstract*—Testing and fixing Web Application Firewalls (WAFs) are two relevant and complementary challenges for security analysts. Automated testing helps to cost-effectively detect vulnerabilities in a WAF by generating effective test cases, i.e., attacks. Once vulnerabilities have been identified, the WAF needs to be fixed by augmenting its rule set to filter attacks without blocking legitimate requests. However, existing research suggests that rule sets are very difficult to understand and too complex to be manually fixed. In this paper, we formalise the problem of fixing vulnerable WAFs as a combinatorial optimisation problem. To solve it, we propose an automated approach that combines machine learning with multi-objective genetic algorithms. Given a set of legitimate requests and bypassing SQL injection attacks, our approach automatically infers regular expressions that, when added to the WAF's rule set, prevent many attacks while letting legitimate requests go through. Our empirical evaluation based on both open-source and proprietary WAFs shows that the generated filter rules are effective at blocking previously identified and successful SQL injection attacks (*recall* between 54.6% and 98.3%), while triggering in most cases no or few false positives (*false positive rate* between 0% and 2%).**

***Keywords*-Web Application Firewalls, Regular Expression Inference, Web Security**

I. Introduction

Nowadays, IT companies are required to constantly improve their protection mechanisms to cope with the increasing number and complexity of cyber-attacks. A good practice consists of deploying a stratified protection infrastructure working at different levels, including application level, network level and communication protocol level [21], [52].

Web Application Firewalls (WAFs) are the most common application-level mechanisms for mitigating and preventing specific types of attacks, such as SQL Injection or Cross-Site scripting. WAFs examine incoming HTTP traffic and decide whether to accept or to discard each request before forwarding it to the protected application. The decision procedure is based on a rule set, i.e., a sequence of regular expressions designed to detect many forms of attacks.

Defining effective rule sets is extremely important to reduce the attack surface that can be exploited by attackers. However, as noticed by Wool [56], corporate firewalls often enforce poorly written regular expressions in practice. This is due to the fact that understanding and analysing how a firewall behaves is notoriously difficult and maintaining a proper configuration is error-prone [57].

In the context of WAF testing, most of research effort has been devoted to generating attacks able to bypass WAFs using various automated testing approaches [1], [2], [4], [22], [41]. While identifying security flaws helps highlight the inadequacy of existing configurations, little attention has been paid to supporting the refinement of inadequate rule sets. Indeed, once attacks are found, either with automated tools or by analysing the HTTP traffic, security analysts have to add additional rules to block the newly discovered attacks without preventing legitimate traffic. As highlighted by previous studies [10], [11], the average time required to fix the WAF configuration is higher than the average time required by a hacker to find and exploit a vulnerability. Therefore, there is a need for automated fixing strategies.

In this paper, we focus on the problem of inferring regular expressions to fix WAFs rule sets in the context of SQL injection (SQLi) vulnerabilities. According to Clarke [21], adding new rules to fix WAFs requires to consider two contrasting goals: (i) *maximise* the number of known SQLi attacks blocked by the firewall (*recall*); and (ii) *minimising* the number of legitimate requests improperly blocked (*false positive*). The first goal is very intuitive since we want to fix as many security flaws as possible; the latter goal is less intuitive but equally critical since, depending on the context, false positives may render an application unusable [21]. Finally, choosing the most appropriate trade-off between the two goals strictly depends on the application type and on the security mechanisms in place to complement WAFs [21].

For these reasons, we recast the derivation of new firewall rules as a multi-objective search problem, with recall and false positive rate as objectives to optimise. In particular, we use a well-known multi-objective genetic algorithm, namely NSGA-II [23], to derive Pareto efficient rule sets learning from both past legitimate traffic and SQLi attacks automatically discovered with a black-box testing technique [4]. We evaluated our multi-objective technique on one popular open-source WAF, namely ModSecurity for Apache HTTP Server, and one proprietary WAF from a financial service provider. The results show that the generated rules are effective at blocking many of the previously bypassing attacks (*recall* between 54.6% and 98.3%), while inducing

a very small number of false positives (*false positive rate* between 0%, most of the time, and 2%). This suggests that our solution is promising to help automatically repairing WAFs based on testing results, though such repair is partial.

As additional benefit, the multi-objective approach provides multiple Pareto efficient rule-sets representing optimal trade-offs between the two objectives. This allows security analysts to choose the rule set that best fit their needs, e.g., the rule set able to block all known SQLi attacks (however with more false positive), the rule set incurring the lowest false positive rate (however leaving few attacks to block with other protection mechanisms), or a rule set striking a compromise between the two objectives.

II. BACKGROUND

SQL Injection (SQLi) attacks have received much attention from academia and practitioners [3], [5], [17], [19], [25]–[28], [33], [39], [49], [50]. Yet SQLi incidents occur on a frequent basis since developers work under pressure and are not always fully aware of injection issues. The Open Web Application Security Project (OWASP) ranks injection attacks as the most dangerous web attacks, while stating that their impact is severe and their prevalence is common.[1]

SQLi is a widely-recognised attack technique in which attackers inject malicious SQL code fragments into input parameters. When web applications lack proper validation or sanitisation of input parameters, an attacker might be able to construct input values that change the behaviour of the resulting SQL statements. Such SQL queries may result in arbitrary actions on the application database, possibly violating its security properties, e.g., exposure of sensitive data, insertion or alteration of data without authorisation, loss of data, or even taking control of the database server.

Web applications with high security requirements are protected by a stratified security infrastructure which commonly includes WAFs. A WAF is placed in front of the web application to be protected and it examines every incoming request before forwarding it to the target application. The WAF hands over the request to the web application only if the request complies with the firewall's rule set.

A common approach to define the WAF's rule set is using a black-list. A black-list contains string patterns, typically defined as regular expressions. Requests recognised by these patterns are considered to be malicious attacks (e.g., SQLi) and, therefore, are blocked. For example, the following regular expression describes the syntax for SQL comments, e.g., /**/ or #, which are frequently used in SQLi attacks:

```
/\*!?|\*/|[';]--|--[\s\r\n\v\f]
    |(?:--[^-]*?-)
    |([^\-&])#.*?[\s\r\n\v\f]|;?\\x00
```

There are several reasons why a WAF may have loopholes and provide insufficient protection, including an incomplete rule set, implementation bugs, or misconfiguration. One way to ensure the resilience of a WAF against attacks is to rely on an automated testing procedure that thoroughly and efficiently detects its vulnerabilities.

Regarding the generation of bypassing attacks, our previous papers [4], [6] addressed this challenge for SQL injections, one of the main types of vulnerabilities observed in practice. In particular, we proposed a black-box approach that iteratively generates new candidate attacks learning from past test execution results [4]. The approach starts with an initial set of potential attacks that are generated using a BNF grammar suitably defined to cover various type of SQLi attacks. These tests are then executed against the WAF under test and they are labeled as "P" or "B" depending on whether they bypass or they are blocked by the firewall, respectively. Tests and corresponding labels are used as training dataset to learn Decision Tree (DT) models [29], [47] with the purpose of determining which string patterns are associated with bypassing attacks. The string patterns that are more likely to lead to actual SQLi attacks (according to a DT) are then used to create new test cases to execute against the WAF in the next iterations of the generation process. Periodically, the DT model is re-trained with bypassing and blocked attacks that have been generated since the latest training process.

While this testing technique is able to effectively find bypassing SQLi attacks, it does not help security analysts in fixing a vulnerable WAF. Indeed, newly discovered attacks only represent a starting point for the analysts who are in charge of manually improving the WAF's rule set. Bypassing attacks found with automated testing techniques, together with legitimate traffic, represent the input for the approach proposed in this paper, which aims to fix vulnerable WAFs to the maximum extent possible.

III. APPROACH

Consider the task of inferring some WAF filter rule, i.e. a regex, from a set of known bypassing attacks. The problem by itself can be viewed as a search problem where the goal is to find, among the set of all possible regular expressions, the regex that matches all bypassing attacks without matching any legitimate request. However, such a trivial formulation leads to an extremely large space of regexes to consider.

In our previous paper on SQLi attack generation [4], [6], we found that machine learning approaches, and DT in particular, can be used to determine string patterns that are more likely to characterise attacks bypassing the WAF. We argue that these string patterns are natural candidates to be translated into regular expressions since they will match most of the attacks in the DT learning set. In other words, these string patterns help focusing the search on sub-regions of the regex space that are more likely to contain "good" candidate solutions. Starting from this observation, in this paper we propose a two-step approach: we first use DT models to derive string patterns from SQLi attacks; then,

[1]For details refer to OWASP Top 10 2013

Table I
AN EXAMPLE OF ATTACK DECOMPOSITIONS AND THEIR ENCODING.

t.id	vector	label
1	$\langle s_1, s_2, s_3 \rangle$	B
2	$\langle s_1, s_2, s_4 \rangle$	B
3	$\langle s_4, s_5 \rangle$	P

t.id	s_1	s_2	s_3	s_4	s_5	clz
1	1	1	1	0	0	B
2	1	1	0	1	0	B
3	0	0	0	1	1	P

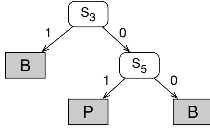

Figure 1. An example of a DT obtained from the data in Table I.

starting from the regexes matching these string patterns, we apply genetic algorithms to optimize them to block as many attacks as possible while blocking as few legitimate requests as possible. The two steps are described in the next subsections.

A. Extracting Attack Patterns with Decision Tree

In the first step, we build DT models starting from the set of SQLi attacks that either are blocked by or bypass the WAF under test. To this aim, we first decompose the attacks into slices using the procedure we introduced in our previous paper on SQLi attack generation [4], [6]. In particular, each attack can be viewed as a parse tree and its slices are subtrees that correspond to attack sub-strings [4]. Then, we form a dataset in which each instance encodes an attack, which is characterised by the presence and absence of slices and a label indicating whether the attack bypassed or was blocked.

As a concrete example, let us assume we have three attacks t_1, t_2, t_3; the first two are blocked while the last one bypasses a WAF. Their decompositions into slices and labels are shown on the left side of Table I, and their encoded presentation on the right side of the table. In total, there are five unique slices from all the attacks and they become attributes of the training data set fed to a machine learning algorithm. If a slice appears in an attack, its corresponding attribute value in the training data is "1", and otherwise "0".

Based on the obtained dataset, we train a DT model to derive which slices or combinations of slices are associated with attacks bypassing or being blocked by the WAF. In a DT, a node represents an attribute from a data set, each branch represents a possible value of an attribute, and a leaf node represents a classification for all instances that reach this node. In the context of SQLi attack generation, each node represents a slice and the branches from the node can be "0" or "1", indicating whether the slice is absent or present. A leaf node classifies instances into blocked or bypassing and is labelled accordingly with "B" or "P".

Figure 1 presents a decision tree learned from the example data set in Table I. It has three leaf nodes and two intermediate ones associated with slices s_3 and s_5. The paths from the root node of a decision tree to its leaf nodes embody the combinations of slices which are likely to be determining conditions for attacks to bypass or be blocked. We define the concept of *path condition* as [4]:

Definition 1 (Path Condition)**.** *A path condition represents a set of slices that the machine learning technique identifies as good predictors of the attack's classification into blocked or bypassing. The path condition is represented as a conjunction $\bigwedge_i^k (s_i = val)$, in which $val = 1 \mid 0$, and k is the number of relevant slices.*

Considering the decision tree in Figure 1 again, for the attack t_1, the attribute s_3 is present in its slice vector $\langle s_1, s_2, s_3 \rangle$, and thus t_1 follows the left branch and the path condition is simply $(s_3 = 1)$. Similarly, for the attack t_3 with slice vector $\langle s_4, s_5 \rangle$, since attribute s_3 is not present, the right branch is followed leading to attribute s_5, which is present in the slice vector. Therefore, the resulting path condition for t_3 is $(s_3 = 0 \wedge s_5 = 1)$.

B. Defining WAF Rules: A Search Problem

Given the results of the DT models, our search space is defined by path conditions as follows:

Definition 2 (Search Space)**.** *Let $P = \{p_1, \ldots, p_n\}$ be a set of path conditions and let $S_i = \{s_{i1}, \ldots, s_{im}\}$ be the set of slices that appear as terms in some path condition $p_i = s_{i1} \wedge \ldots \wedge s_{im}$, $p_i \in P$. The search space $\mathcal{S}$ for the WAF fixing problem is defined by the Cartesian product $\mathcal{S} = \mathcal{P}(S_1) \times \ldots \times \mathcal{P}(S_n)$, where $\mathcal{P}(S_i)$ denotes the power set of S_i.*

According to Definition 2, a candidate solution in the search space $\mathcal{S}$ is a n-tuple, which has one element per path condition $p \in P$ and each element in the tuple represents a combination of slices from the corresponding path condition.

For example, let us consider the three path conditions and their corresponding power sets shown in Table II. A candidate solution in this example is a 3-tuple, e.g. $c_1 = (\{s_{11}, s_{12}\}, \{s_{21}\}, \{s_{33}\})$. The first element of the tuple represents a combination of slices of p_1, the second element a combination of slices of p_2, and the third element a combination of slices p_3. The first element of c_1 can be translated into regex r_0, which identifies any attacks that contain the slices s_{11} and s_{12}. Similarly, the second and third element can be translated into regex r_1 and r_2 that identify attacks containing the slices s_{21} and s_{33}, respectively. In other words, each regex r_0, r_1, or r_2 matches a distinct set of attacks. The final regular expression able to block all attacks can be defined as $r = r_0 \mid r_1 \mid r_2$, i.e., r matches the attacks that are matched by either r_0, r_1, or r_2.

Given the final regex r, the problem consists of selecting the subset of slices composing each regular expression r_i in r such as to block as many as possible known SQLi attacks. However, some slices appearing in SQLi attacks (i.e., belonging to one or more path conditions) may also appear in

Table II
AN EXAMPLE OF THREE PATH CONDITIONS AND THEIR POWER SETS.

Path Cond.	Slices	$\mathcal{P}(S_i)$
p_1	$\{s_{11}, s_{12}\}$	$\emptyset, \{s_{11}\}, \{s_{12}\}, \{s_{11}, s_{12}\}$
p_2	$\{s_{21}\}$	$\emptyset, \{s_{21}\}$
p_3	$\{s_{31}, s_{32}, s_{33}\}$	$\emptyset, \{s_{31}\}, \{s_{32}\}, \{s_{33}\}, \{s_{31}, s_{32}\}, \{s_{31}, s_{33}\}, \{s_{32}, s_{33}\}, \{s_{31}, s_{32}, s_{33}\}$

legitimate requests. As a consequence, the selection of some slices to form the final regex r may result into a WAF rule which blocks some legitimate requests as well. As pointed out by Clarke [21], false positives must be avoided since they can critically compromise normal web application behaviour (e.g. the WAF blocks the authentication of legitimate users). Therefore, to assess the quality of a solution, both SQLi attacks and legitimate traffic should be taken into account.

Let R denote a set of requests that are processed by a WAF such that $A \subset R$ is a subset of malicious requests and $L \subset R$ is a subset of benign requests. Note that each request is either malicious or benign and, thus, $A \cap L = \emptyset$ and $A \cup L = R$. Let $M(c, R)$ denote a set of matched requests by applying a regex c to R, let $M_{tp}(c, R) = \{x \mid x \in A \wedge x \in M(c, R)\}$ denote the set of true positive matches and $M_{fp}(c, R) = \{x \mid x \in L \wedge x \in M(c, R)\}$ the set of false positive matches.

To assess the quality of a candidate solution we use two well-known measures, *false positive rate* and *recall*, that fit our security concerns: the former quantifies blocked legitimate traffic while the latter measures how well we detect attacks.

Definition 3 (Objective Functions)**.** *Given a solution $c \in \mathcal{S}$ and a set R of request, we assess the quality of c with:*

$$fpr(c, R) = \frac{|M_{fp}(c, R)|}{|L|} \quad rec(c, R) = \frac{|M_{tp}(c, R)|}{|A|} \tag{1}$$

A solution candidate is evaluated with respect to both objective functions: optimal solutions maximise *recall* and minimise the *false positive rate*. For the WAF Fixing Problem, *recall* and *false positive rate* are two conflicting objectives [21], because regular expressions that improve recall may also increase the number of legitimate requests being blocked. Therefore, solving this bi-objective search problem consists of finding optimal trade-offs between *recall* and *false positive rate*, which may lead to several solutions forming a Pareto front [23]. To compare the quality of two candidate solutions, we use the dominance $\prec$ relation:

Definition 4 (Dominated and non-dominated Solutions)**.** *Given a candidate solution $c \in \mathcal{S}$ and a set R of requests, c is said to be* ***non-dominated*** *if:*

$$\begin{aligned} \nexists c' \in \mathcal{S} : (fpr(c, R) > fpr(c', R) \wedge rec(c, R) \leq rec(c', R)) \\ \vee (fpr(c, R) \geq fpr(c', R) \wedge rec(c, R) < rec(c', R)) \end{aligned} \tag{2}$$

Accordingly, c is said to be ***dominated*** *if there is a $c' \in \mathcal{S}$ that satisfies Equation 2. In this case, we use the notation $c \prec c'$ to indicate that c' dominates c.*

In our problem context, the severity of both unblocked SQLi attacks (false negatives) and blocked legitimate requests (false positives) depends on the web application functionality and on the security infrastructure in place to protect it [21]. In general, false positives are more critical than true negatives because they affect the interactions between legitimate users and the application under protection [21] and therefore affect the business or functional viability of the application. For example, false positives may mean that registered users cannot access a web application or legitimate bank transfers are not allowed anymore after fixing the WAF. On the other hand, to some extent, it may be tolerable to let some SQLi attacks bypass the WAF if other mechanisms of the infrastructure adequately protect the application or are strengthened [21], e.g., by adding new checks in the vulnerable source code.

Before accepting a solution candidate, the security analyst has to consider how each false positive caused by the solution candidate impacts the functionality of the system under test and possibly choose the next best solution candidate. Therefore, we introduce a constraint on the number of false positives to limit the required manual effort:

Definition 5 (Feasible Candidate Solution)**.** *We say $c \in \mathcal{S}$ is a feasible candidate solution if it satisfies the constraint $|fpr(c, R)| < t$, where t is a user-defined threshold for an acceptable false positive rate. Additionally, $\mathcal{X} \subseteq \mathcal{S}$ denotes the set of all feasible candidate solutions. Based on the constraint we define a function g that measures the degree of constraint violation:*

$$g(c) = \begin{cases} 0 & \text{if } c \in \mathcal{X} \\ |M_{fp}(c, R)| - t & \text{if } c \notin \mathcal{X} \end{cases} \tag{3}$$

The problem addressed in this paper can now formally be defined as a constrained optimisation problem.

Definition 6 (WAF Fixing Problem)**.** *Given a set $L \subset R$ of benign requests, a set $A \subset R$ of malicious requests, a set of path conditions defining the search space $\mathcal{S}$, and a set $\mathcal{X} \subseteq \mathcal{S}$ of feasible candidate solutions, then the WAF Fixing Problem is to compute a set $\mathcal{C}_{opt}$:*

$$\mathcal{C}_{opt} = \{c \mid c \in \mathcal{X} \wedge \nexists c' \in \mathcal{X} : c \prec c'\} \tag{4}$$

According to Definition 6, the WAF Fixing Problem consists of finding a set C_{opt} of non-dominated solutions in the space of feasible solutions.

C. Tailoring NSGA-II to the WAF Fixing Problem

To solve the WAF Fixing Problem, in this section we introduce an approach based on a well-known multi-objective genetic algorithm, namely NSGA-II [23]. We selected NSGA-II because it has been widely used in the software engineering community to address problems with multiple objectives [55], e.g., regression testing [30]. In addition, it is

Table III
AN EXAMPLE OF TWO CANDIDATE SOLUTIONS ENCODED AS CHROMOSOMES BASED ON THE PATH CONDITIONS AND SLICES DEPICTED IN TABLE II.

id	Candidate Solution
c_1	$(\{s_{11}, s_{12}\}, \{s_{21}\}, \{s_{33}\})$
c_2	$(\{s_{12}\}, \emptyset, \{s_{31}, s_{33}\})$

id	p_1		p_2	p_3		
	s_{11}	s_{12}	s_{21}	s_{31}	s_{32}	s_{33}
c_1	1	1	1	0	0	1
c_2	0	1	0	1	0	1

applicable to multi-objective optimisation problems with a constrained search space like the WAF Fixing Problem [23].

NSGA-II is a search heuristic that is inspired by the evolutionary process in nature, where the fittest individuals of a population prevail and pass their genes to their offspring. Starting from a randomly generated initial population of candidate solutions, NSGA-II generates an offspring population by performing crossover and mutation operations on the individuals and by selecting the best candidate solutions according to the *dominance relation* (Definition 4) and *crowding distance*. The former mechanism reflects the degree with which a candidate solution achieves the desired objectives; the latter is in charge for maintaining diversity between solutions. In the following, we describe how the essential components of NSGA-II, i.e., chromosomes, selection, crossover, mutation, and constraint handling, are tailored to our context.

Chromosomes. A fundamental decision when using Genetic Algorithms (GAs) is how to encode a candidate solution and its properties. In the terminology of GAs, a solution candidate is called a *chromosome*. A chromosome is composed of several *genes*, each representing a possible value of a decision variable in the candidate solution. In our context, each path condition that is used to define the search space is a gene of the chromosome and each slice that is part of a path condition is encoded as a bit in the gene.

Table III shows an example of two candidate solutions and their corresponding encoding as chromosomes. The path conditions and slices used in this example are from the previous example in Table II. As before, there are in total three path conditions and, hence, the candidate solutions c_1 and c_2 are triplets. The first element of c_1 and c_2 represents a combination of slices of path condition p_1. For c_1, the first element is $\{s_{11}, s_{12}\}$ and, thus, the chromosome encoding of p_1 is $s_{11} = 1$ and $s_{12} = 1$. For c_2, the first element is $\{s_{12}\}$ and, thus, the chromosome encoding of p_1 is $s_{11} = 0$ and $s_{12} = 1$. The second and third element of c_1 and c_2 are encoded in the same fashion.

Selection, Crossover, and Mutation. To efficiently sample the search space, NSGA-II creates new solution candidates (children) by crossovering and mutating the chromosomes of candidates (parents) from the current population. First, for each individual in the current population, a fitness vector is calculated using our objective functions (Definition 3). Then, two parents are selected using a standard binary tournament selection [40] and their chromosomes are swapped starting from a randomly chosen gene in the chromosome, i.e. single-point crossover [42]. Subsequently, each child is mutated with a certain probability. We implement the mutation operator as a bit-flip mutation, where each bit of the chromosome is flipped with a certain probability.

Constraint Handling. To guide the search process towards feasible solutions, we introduce a constraint in the search space (Definition 5). We integrate the constraint handling mechanism into the standard binary tournament selection as suggested by the authors of NSGA-II [23]. Given two solutions i and j, which are selected to compete in a binary tournament, there are three possible situations: 1) i and j are both feasible; 2) either i or j is feasible; and 3) i and j are infeasible. For case 1), the usual binary tournament is performed. For case 2), the feasible solution is selected. For case 3), the solution with a lower constraint violation degree is selected.

IV. EMPIRICAL EVALUATION

This section details the research questions and the case studies used in this evaluation, and reports on results.

A. Subject Applications

We evaluate our multi-objective approach in two different case studies: the first case study involves a widely-used, open source WAF while the second one targets a proprietary application with a WAF from a financial service provider that processes several thousand transactions daily.

In the first case study, we assess ModSecurity[2], which is a popular open source WAF for the Apache HTTP Server. The rule set used to detect malicious requests is the OWASP Core Rule Set, which can detect a large number of common web attacks like SQL Injection (SQLi) and is actively developed by a community of security experts. Configuring ModSecurity with the OWASP Core Rule Set is a popular choice in practice [21]. In this study, ModSecurity is used to protect Cyclos[3], a popular online and mobile banking software.

In the second case study, we assess a proprietary WAF from a financial service provider. The WAF is configured to protect a complex SOA system, which processes financial transactions. To provide protection from malicious requests, the WAF validates incoming requests in two steps. In the first step, the values of each incoming request are validated with respect to data types (e.g. string or numeric) and boundary constraints, e.g. a credit card number is expected to be a sequence of 16 to 19 digits. In a second step, each value is checked to make sure that it does not contain known malicious string patterns (i.e., using a SQLi blacklist) commonly used in attacks. Only when the request passes both validation steps it is forwarded to the web services.

For both case studies, we collected SQLi attacks that are not correctly identified by the WAF, i.e. bypassing attacks,

[2]https://modsecurity.org
[3]http://www.cyclos.org

and a sample of benign requests. The benign requests represent the legitimate usage of the protected web application and, thus, the evaluated WAF is expected to let these requests pass. We collected the benign requests by executing the functional test suite of the respective web application and logging each HTTP request sent to the web application. In general, legitimate requests can be easily collected by monitoring and mining the daily HTTP traffic of the web application being protected by WAFs. Instead, collecting malicious requests is more difficult: they are smaller in number and difficult to detect in advance. Therefore, we executed the automated WAF testing strategy from our previous work [4] (and briefly summarized in Section II) to collect them. Notice that our attack generation approach is not mandatory and therefore other tools (e.g., SqlMap[4]) can be used to this aim. We used the approach in [4] because it finds more SQLi attacks able to pass through the target WAF (more effective) and in less amount of time (more efficient) than random testing.

Computation time required for such experiments is potentially extensive because of (i) a large number of web service operations, (ii) a large number of attacks, (iii) a large number of legitimate requests, (iv) a large number of repetitions to evaluate randomised search algorithms (100 runs for each operation). As a result, we randomly selected a subset of web service operations for the target applications (and their WAFs). For the open source case study, we selected three operations (`doPayment`, `expireTicket`, `simulatePayment`) since we found that the number of bypassing attacks does not vary significantly across the tested web service operations. For the industrial case study, on the contrary, we found that the number of bypassing attacks does vary significantly depending on the tested web service operation [6]. Therefore, we grouped the test results into four groups having a similar number of bypassing attacks. Then, for this experiment we randomly selected one web service operation from each group (referred to as `Operation1`, `Operation2`, `Operation3`, `Operation4`).

Table IV lists, for each selected operation, the total number of benign requests, the total number of bypassing attacks, and the number of path conditions learned from the bypassing attacks as detailed in Section III-A. For brevity, we refer to each of the seven combinations of operation, benign requests, attacks, and path conditions as a dataset.

B. Research Questions

The two case studies aim at answering the following research questions:

- **RQ1**: *How effective are the found regular expressions in identifying bypassing attacks?*

[4]https://sqlmap.org

Table IV
OVERVIEW OF THE CASE STUDIES.

Case Study	Benign Requests	Operation	#Bypassing Attacks	#Path Conditions
Open Source	1567	doPayment	1234	84
		expireTicket	1127	82
		simulatePayment	1265	88
Industrial	2600	Operation 1	943	49
		Operation 2	19957	103
		Operation 3	169	39
		Operation 4	11462	92

- **RQ2**: *To which extent do the found regular expressions misclassify legitimate traffic as attacks?*
- **RQ3**: *How does NSGA-II compare to random search?*

The goal of the proposed approach is repairing vulnerable WAFs, i.e. to find regular expressions that block the bypassing malicious requests without affecting legitimate requests. Therefore, **RQ1** investigates how many malicious requests the generated regular expressions identify, i.e. *recall*. **RQ2** investigates if the regular expressions misclassify legitimate requests as attacks, i.e. *false positive rate*.

The last research question (**RQ3**) aims at evaluating the benefits produced by our multi-objective approach (based on NSGA-II) compared to Random Search (RS). There are various reasons for considering RS as baseline. First, according to the guidelines in search-based software engineering [7], [46], RS is necessary to check whether new problems being addressed are sufficiently difficult to require sophisticated search strategies. Second, RS has turned out to be more effective and more efficient than evolutionary algorithms for automated software repair [46] in other contexts, as well as for other problems (e.g., [15]).

In our context, the implementation of RS is straightforward: it randomly generates individuals from the search space defined by the WAF Fixing Problem. As for NSGA-II, in RS individuals are binary vectors that are evaluated according to our two objectives, i.e., recall and false positive rate. Among all randomly generated solutions, the non-dominated ones (see Definition 4) satisfying the constraint (Definition 5) form the final Pareto front.

C. Experimental Procedure and Parameter Settings

We executed each search algorithm (i.e., NSGA-II and RS) on each of the three datasets from the open source case study and each of the four datasets from the industrial case study. At the end of each run, we collected the set of non-dominated solutions (i.e., the Pareto set) produced by a given algorithm as well as the corresponding recall and false positive scores (i.e., the Pareto frontier) for comparison. In each run, the algorithm is terminated after $10K$ fitness evaluations, which correspond to a maximum search timeout of 240 minutes on average. To account for the randomised nature of NSGA-II and RS, each run is repeated 100 times. Consequently, we performed a total of 2 (algorithms) $\times$ 7 (datasets) $\times$ 100 (repetitions) $=$ 1,400 executions.

Evaluation metric. To compare the Pareto fronts produced by NSGA-II and RS, we followed the guidelines by Wang et al. [55] to select a suitable quality indicator. Since our goal is to evaluate the convergence of the search as well as the diversity of the found solutions, we selected the well-known quality indicator hypervolume (HV) [34]. HV takes a value within the interval $[0; 1]$ and it measures the volume in the objective space that is dominated by a set of non-dominated solutions. The higher the HV value, the better the quality of the Pareto front. We also analyse the variability of the HV scores achieved by an algorithm across multiple runs using the *interquartile range* (IQR), which is a standard measure of statistical dispersion. It is defined as the difference between the first quartile and the third quartile of the HV distribution. A zero IQR value indicates that the algorithm produces the same HV score across different runs, while larger IQR values indicate that the results change over the runs. Finally, to check whether the difference between the HV values produced by NSGA-II and RS are statistically significant or not, we use the non-parametric Wilcoxon test with p-value $= 0.05$ as threshold for significance [7]. The Wilcoxon test does not make any assumption on the nature of the distributions.

Parameter settings. We adopt the default parameter settings defined in jMetal [44] for NSGA-II. In particular, we use the standard *single-point crossover* with a probability of $p_c = 0.9$, the *bit-flip mutation* with probability $p_m = 1/l$, where l is the length of the chromosome, and the population size is set to 100. For a fair comparison, both NSGA-II and RS were configured with $10K$ as maximum number of fitness evaluations. We use the number of fitness evaluations as a measure of computational cost (time) for the search algorithms since the overhead due the genetic operators is negligible compared to the cost required to evaluate each generated individual. While different results may be obtained with different settings, we opted for the default parameter values because previous empirical results [8] showed that such default settings give acceptable results.

Another parameter that can affect our results is the choice of the threshold t for the constraint (Definition 5). If t is too low, the search prematurely converges, since individuals violating the constraint are likely to be abandoned and their genetic information is lost. On the other hand, if the threshold is too high, the algorithm performs expensive fitness evaluations on infeasible solutions. In a series of pilot experiments, we found that NSGA-II performs well with $t = 9/10* \mid L \mid$, where L is the set of benign requests.

D. Implementation

We used the implementations of NSGA-II, RS, and the performance metric (HV) available in JMetal [44], a popular Java framework for search problems. Within the same framework, we implemented the instances of the WAF Fixing Problem for the seven datasets selected for our case study.

In our search problem, the fitness evaluation consumes a large majority of the computation time and its efficient implementation is of high importance. A naive way to compute false positive rate and recall consists in translating a solution candidate (binary chromosome) into a regular expression, adding it to the rule set of the subject WAF, and sending a HTTP request for each legitimate and attack request. Since updating the rule set frequently and sending a large quantity of HTTP requests induces a high computational overhead, our fitness evaluation procedure uses an embedded version of a regular expression engine of the subject firewall. Thereby, we avoid sending HTTP requests and skip other irrelevant operations related to the WAF, while still obtaining precise measures for recall and false positive rate. The regular expression engine used by the subject WAFs and our implementation is PCRE[5], version 8.33. Our implementation uses the command-line tool *pcregrep* to match a given regular expression against the legitimate and attack requests.

The total computation time of our experiments and the processing of results is equivalent to ≈ 97 days on a single CPU core of a typical notebook (i.e. 2.26 GHz).

E. Results

Figure 2 plots the average HV scores achieved for each dataset by NSGA-II and RS over the number of fitness evaluations. Recall that, in our context, computational time is mostly driven by fitness evaluations as explained in Section IV-C. The curves in Figure 2 are augmented with boxplots showing the variation of the HV scores across 100 independent runs at 300, 1000, 1700 and 2400 fitness evaluations. Note that the x-axis is truncated at 2400 evaluations although we terminate the algorithms after $10K$ evaluations. However, we note that the HV values do not vary after this point in all the datasets.

As we can observe, when reaching 2400 fitness evaluations, NSGA-II produces Pareto fronts with a median HV value above 0.99 for `doPayment` and `simulatePayment` (from the open source case study) as well as for `Operation2`, and `Operation4` (from the industrial case study). For the dataset `expireTicket` (open source case study) we observe a median HV value of 0.82. Finally, for the remaining two datasets from the industrial case study (i.e., `Operation1`, and `Operation3`), the median HV values at 2400 fitness evaluations are 0.92, and 0.98, respectively.

Looking at the variability of HV values produced by NSGA-II (see the boxplots in Figure 2), we observe that the IQR is large (0.06 on average) at 300 fitness evaluations but tends to decrease quickly and become very small (< 0.01 on average) at 2400 fitness evaluations. The only exception is the dataset `Operation1`, for which the IQR remains high and slightly decreases in the range up to $10K$ fitness

[5]http://pcre.org

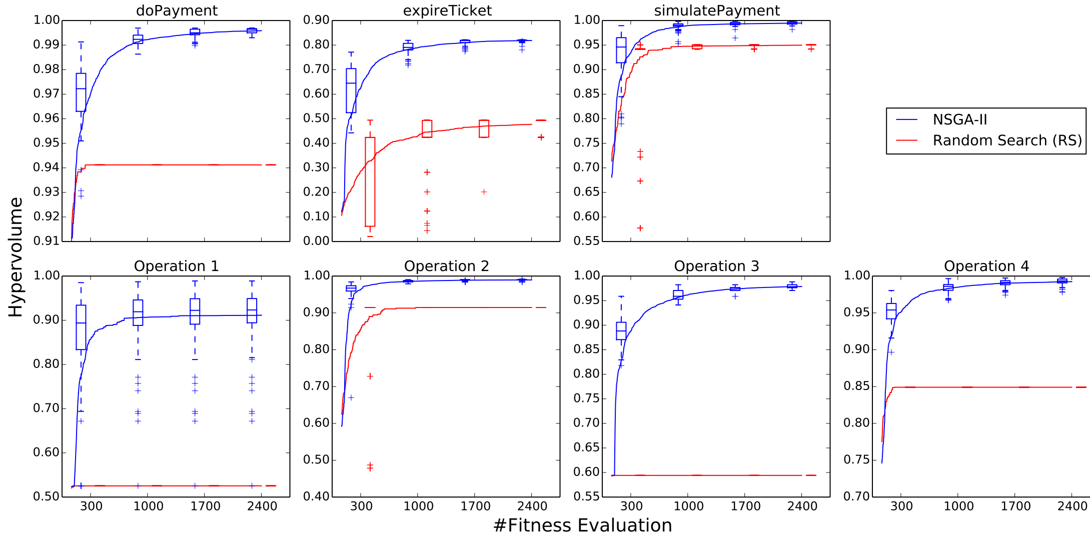

Figure 2. Comparison of the hypervolume over fitness evaluations for NSGA-II (blue) and RS (red).

evaluations. One possible explanation is that the dataset `Operation1` contains fewer bypassing attacks (< 1000) to be blocked and 2600 legitimate requests to handle. Thus, solutions slightly improving recall (e.g., one additional attack being blocked) are likely to result in a large increase of the false positive rate. Instead, datasets with better (lower) IQR scores are characterised by a better balance between attacks and legitimate requests. For example, `Operation2` is the dataset with the best IQR score (0.02) at 300 fitness evaluations. In this dataset, the number of SQLi bypassing attacks is 19000 while the number of benign requests is 2600. This analysis suggests that when the number of attacks to block increases, it becomes easier for NSGA-II to find slices (string patterns) that block attacks without preventing legitimate traffic, thus helping to lower variability of the HV scores over the runs.

Comparing NSGA-II and RS, we find that the former is consistently better than the latter in terms of HV. Indeed, for RS on most datasets, we observe that HR quickly converges to sub-optimal HV values and no considerable further improvement is made after 300 fitness evaluations. Only for `expireTicket` and `simulatePayment`, the HV scores of RS increase after 300 fitness evaluations. However, even in these cases, RS converges to sub-optimal fronts at 2400 and 1700 function evaluations, respectively. The difference in terms of HV scores between NSGA-II and RS ranges between 0.05 (for `simulatePayment`) and 0.40 (for `Operation1`) at 2400 fitness evaluations. The fact that RS cannot close the large HV gap with NSGA-II after $10K$ fitness evaluations indicates that the solutions found by NSGA-II are widely distributed in the Pareto front and unlikely to be found randomly. According to the Wilcoxon test, the differences between NSGA-II and RS in terms of HV scores are always statistical significant with

Table V
COMPARISON OF THE RECALL SCORES ACHIEVED BY NSGA-II AND RS WHEN SELECTING THE NON-DOMINATED SOLUTION WITH THE MINIMUM FALSE POSITIVE RATE.

Operation	NSGA-II		RS	
	$FPR(\%)$	$Recall(\%)$	$FPR(\%)$	$Recall(\%)$
doPayment	0.0	84.05	0.0	73.12
expireTicket	1.0	54.61	0.7	42.64
simulatePayment	0.0	75.43	0.0	73.06
Operation 1	0.0	52.54	0.0	52.54
Operation 2	0.0	98.28	0.0	91.46
Operation 3	0.0	97.06	0.0	59.15
Operation 4	0.0	98.16	0.0	84.91

p-values < 0.01.

The presented experiments were executed on a high-performance cluster [54]. The average running time on a single CPU core for a test run with $10K$ fitness evaluations is 63.7 minutes for NSGA-II and 58.58 minutes for RS. The fact that the difference in average running time between NSGA-II and RS is small confirms that the fitness evaluations consume a large part of the running time. Note that our results suggest that less than 3K fitness evaluations are sufficient to reach an optimal Pareto front and that, in practice, fitness evaluations can be performed in parallel on multiple CPU cores to reduce the effective running time.

Trade-off analysis. As discussed in Section III, false positives are often more critical than true negatives in practice [21]. Indeed, while bypassing SQLi attacks can still be handled by strengthening the other protection mechanisms of the infrastructure, e.g., input sanitization in the application, blocked legitimate requests (false positives) can have dramatic consequences on the viability of the application and cannot be addressed by other mechanisms. For example, we can easily imagine the financial consequences of arbitrarily denied credit card transactions or bank transfers. Therefore, we focus our analysis on the solutions (i.e., rule set) in the Pareto fronts that produces the lowest number of false

positives. For the sake of analysis, we pick Pareto fronts among 100 runs, which correspond to the median HV values depicted in Figure 2. The results are reported in Table V.

We notice that using NSGA-II, with the same false positive rate, recall increases between 2.37% (+30 blocked attacks) and 37.91% when compared to RS. We observe only one case, namely `Operation1`, for which both NSGA-II and RS achieve the same recall when the false positive rate is fixed to zero. However, RS produces only one non-dominated solution (the one reported in Table V) while it fails to produce alternative (near) optimal trade-offs for the security analysts (see the HV scores produced for this dataset in Figure 2). On the other hand, NSGA-II produces additional non-dominated solutions that the analysts can consider to make a more appropriate decision in context.

Validation. To further assess the regexes generated by our approach, we test the fixed WAFs against previously unseen requests. For this analysis, we consider the proprietary WAF from the industrial case study and the solutions (fixes) in the Pareto fronts that produce the lowest number of false positives. In particular, we first augmented (fixed) the WAF's rule set with the regular expressions automatically generated by NSGA-II and that correspond to the Pareto optimal solutions reported in Table V. Then, we tested the fixed WAF against a new set of HTTP requests. In total, we collected 575 new legitimate requests by executing additional functional test suites, and 222 additional bypassing attacks by executing the open-source penetration testing SqlMap[6]. Notice that these legitimate requests and bypassing SQLi attacks were not part of the learning set used to fix the WAF.

Our results show that the fixed WAF successfully blocks 203 bypassing attacks out of 222, with a recall ranging between 91.5% for `Operation 2` and 100% for `Operation 4`. Regarding legitimate requests, the false positive rate ranges between 0.86% for `Operation 4` and 1.91% for `Operation 1`. Therefore, our technique is able to generate regular expressions that are also effective for new requests that were not previously used for generating new firewall rules.

Summary. Based on the above results we can answer **RQ1** (Recall) and **RQ2** (False Positive Rate): For the datasets used in our experiments, there are solutions with a high recall and low or null false positive rate for the WAF Fixing Problem. NSGA-II finds these solutions automatically within 100 generations. Regarding **RQ3** (Comparison NSGA-II and RS), we find that NSGA-II statistically outperforms RS, thus motivating the usage of evolutionary algorithms for the WAF Fixing Problem. In most cases, differences are also practically significant.

V. Threats to Validity

To limit the threats to the *construct validity*, we use the HV indicator to compare different search algorithms. Such an indicator is a standard performance metric in multi-objective optimisation because it gives reasonable estimation of both convergence of the search as well as the diversity of the solutions in the generated Pareto fronts [34].

A potential factor that could influence our results (*internal validity*) is the randomised nature of the search algorithms being compared. To address such a threat, we run NSGA-II and RS 100 times and we reported the median results and distribution quantiles. Another potential threat regards the parameters setting used for NSGA-II. We configured NSGA-II with default parameter values since previous work [8] showed that such default configurations produce acceptable results in comparison to fine-tuned settings.

To address potential threats to *conclusion validity*, we followed the guidelines by Arcuri and Briand [7] by using the non-parametric Wilcoxon test to check the statistical significance of differences in HV distributions. We drew our conclusions exclusively from statistically significant results.

To improve the generalizability of our results (*external validity*), we conducted two case studies: a first study with a widely-used, open source WAF (i.e., ModSecurity) and a second study involving a proprietary, industrial WAF. These are real-world WAFs (one from a large financial service provider) protecting web applications with thousands of users. Moreover, we consider thousands of legitimate HTTP messages and attacks for each single web operation.

VI. Related Work

This section discusses the related literature on anomaly detection techniques for application firewalls, regex inference and automated software repair.

Anomaly detection techniques. There are several works in the literature that propose anomaly detection techniques for firewall policies [1], [2], [22], [41]. However, these works consider network firewall policies and do not repair WAFs regex rules. For example, Basile et al. [14] proposed a formal model to detect anomalies in application firewall filter policies by identifying *conflicting rules*, i.e. rules that are activated simultaneously but enforce different actions, and *unnecessary rules*, i.e. rules that can be removed without affecting the WAF decision. In contrast, our approach aims at improving the decision procedure of a WAF by fixing its rule set based on the set of observed bypassing SQLi attacks.

Kruegel et al. [35], [36], [48] proposed multiple anomaly detectors based on several characteristics of HTTP requests, e.g. parameter length, parameter value, and character distribution. Similarly, Kiani et al. [32] proposed a character distribution model that is specifically designed for the detection of SQLi attacks. Valeur et al. [53] proposed a reverse HTTP proxy that mitigates the impact of legitimate requests being misclassified as attacks by routing them to sibling web applications that have only limited access to sensitive data. Kruger et al. [37] introduced TokDoc, a self-healing WAF that analyses HTTP requests and replaces suspicious parts

[6]http://sqlmap.org/

of a request with benign parts observed in the past. Other approaches learn normal HTTP requests using deterministic finite automata [31] and Markov chains [51]. In our previous work [20], we propose a machine learning approach (SOFIA) to automatically detect SQLi vulnerabilities in a web application under test. We recast the oracle problem for SQLi vulnerabilities as a one-class classification problem, in which we learn to characterise legitimate SQL statements to accurately distinguish them from SQLi attack statements. There are important differences between our approach and SOFIA [20]. First, our approach aims at fixing WAF regexes detecting malicious strings in HTTP messages. Instead, SOFIA detects SQLi attacks in complete SQL statements once they are generated by the web application and they reach the database layer. Thus, SOFIA is not applicable at the WAF layer where the inputs are HTTP messages. Second, our approach combines machine learning with multi-objective algorithms to find regular expressions representing (near) optimal trade-offs in fixing the WAF. Instead, SOFIA uses clustering techniques to automatically classify SQL statements (and not HTTP messages) as attacks or legitimate queries.

Attacks that are identified by anomaly detection techniques or generated by automated testing techniques can be used in input for the approach presented in this paper: given a set of attacks (SQLi attacks) and legitimate requests, we want to fix the WAF's rule set such as to block the former while forwarding the latter to the application.

Regex Inference. Inferring regular expression from labelled samples is the subject of a large body of research. Li et al. [38] formulated the task of learning a regular expression from "positive" (to match) and "negative" (not to match) samples as a search problem. The authors apply a set of transformation operators to refine an existing regex with the goal of minimising the false positive matches and maintaining the true positive matches of the initial regular expression. Murthy et al. [43] proposed a human-assisted approach to refine existing regexes such as to maximise true positive matches. Instead, Babbar et al. [9] suggest to use clustering methods to refine the initial regex. Bex et al. [16] infer a Document Type Definition for a given set of XML-documents. The authors recast the problem of learning a regex from positive samples using automata inference. Similarly, Brauer et al. [18] learn regular expressions to match entity types (e.g., identifiers) by selecting suitable entity features and creating prefix and suffix automata. Bartoli et al. [12], [13] generate regular expressions from positive samples using genetic programming (GP).

The work discussed above differs from our paper in several ways. First, previous papers aim at inferring regular expressions from documents written in natural language (e.g., web pages) while we focus on WAF rule sets. In our context, the structure (i.e., chromosome in NSGA-II) of the regex to infer is fixed and known *a priori*: it is the disjunctions of the slices (string patterns) that appear in bypassing SQLi attacks. Instead, for the traditional inference problem the structure of the regular expression to infer is unknown, motivating the usage of GP to address this challenge [12], [13]. Finally, in our paper we address the WAF Fixing Problem with multi-objective genetic algorithms producing (near) optimal trade-offs between recall and false positive rate. Our multi-objective reformulation differs from the one proposed by Bartoli et al. [12], [13] for the traditional regex inference problem. In our paper, recall and false positive rate are addressed as two contrasting objectives to optimise whereas Bartoli et al. [12], [13] combine them into one single objective: the F-measure. The second objective used in [12], [13] is the size of the regular expression inferred by GP. As explained in Section III, for the WAF Fixing Problem the structure of the regular expression to infer is fixed and there is no need to minimise its size.

Automated program repair. Researchers have proposed several techniques to automatically fix source code by using existing test cases and their execution information. Such techniques include GP [24], random search [46], and symbolic execution [45]. Recently, Qi et al. [46] showed that, for the program repair problem, random search outperforms evolutionary algorithms in terms of both effectiveness and efficiency. In this paper, we address a different repair problem since our goal is to fix regular expressions (WAF rule sets) instead of web application source code. We assess the quality of the generated fixes using both bypassing attacks (test cases) and legitimate traffic. However, we strictly followed the guidelines provided by Qi et al. [46] in the context of program repair by comparing our many-objective approach with random search.

VII. Conclusion

This paper proposes an approach to automatically repair vulnerable WAFs by generating rule sets based on an analysis of test results, using machine learning and meta-heuristics. We formalised the WAF fixing problem as a combinatorial optimisation problem and addressed it using a multi-objective genetic algorithm (NSGA-II). Given a set of bypassing attacks and corresponding path conditions extracted from learnt classifiers [4], our approach infers a regular expression that, when added to the WAF's rule set, prevents as many attacks as possible from bypassing, while letting legitimate requests go through.

Experimental results show that with our approach, analysts can always pick solutions among a range of (near) optimal trade-offs such that, for example, the WAF does not prevent legitimate traffic. A WAF is only one component of a protection infrastructure but our results suggest that it can significantly contribute to blocking bypassing attacks with a *recall* ranging between 54.6% and 98.3%, for both open source and industrial WAFs, and no or few false positives

(*false positive rate* between 0% and 2%). In realistic conditions, the automated process of fixing the WAF would not take more than a few hours and can be run on a daily basis. As proof of concept, our approach is currently used by our industrial partner (whose proprietary WAF is repaired in our study) within their verification workflow.

Currently, our approach focuses only on augmenting WAF rule sets. The repair is therefore partial as it does not address incorrect rules. Because the root cause for attacks bypassing a WAF is the incompleteness of its rule set, our objective is to add regexes to enhance incomplete rules. Nevertheless, false positives can be due to existing rule sets and their regexes that should be deleted. We will investigate this aspect in our future work although our empirical results indicate that this scenario never applies for the applications in our case studies. Moreover, as part of our future agenda, we will assess our approach in combination with other automated testing techniques.

Acknowledgments

The authors were supported by the National Research Fund, Luxembourg (grant FNR/P10/03 and FNR4800382). The work was carried out in collaboration with Six Payment Services. The authors would also like to thank Cu D. Nguyen for his feedback on a preliminary draft of this paper.

References

[1] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE journal on Selected Areas in Communications*, 23(10):2069–2084, 2005.

[2] E. S. Al-Shaer and H. H. Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1):2–10, 2004.

[3] D. Appelt, N. Alshahwan, and L. Briand. Assessing the impact of firewalls and database proxies on sql injection testing. In *Proceedings of the 1st International Workshop on Future Internet Testing*, 2013.

[4] D. Appelt, C. Nguyen, and L. Briand. Behind an application firewall, are we safe from sql injection attacks? In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, pages 1–10, April 2015.

[5] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan. Automated testing for sql injection vulnerabilities: An input mutation approach. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 259–269, New York, NY, USA, 2014. ACM.

[6] D. Appelt, D. C. Nguyen, and L. Briand. Automated testing of web application firewalls. Technical report, 2016.

[7] A. Arcuri and L. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 1–10. IEEE, 2011.

[8] A. Arcuri and G. Fraser. On parameter tuning in search based software engineering. In *International Symposium on Search Based Software Engineering*, pages 33–47. Springer, 2011.

[9] R. Babbar and N. Singh. Clustering based approach to learning regular expressions over large alphabet for noisy unstructured text. In *Proceedings of the fourth workshop on Analytics for noisy unstructured text data*, pages 43–50. ACM, 2010.

[10] R. Barnett. Dynamic DAST/WAF integration: Realtime virtual patching.

[11] R. Barnett. Owasp virtual patching survey results.

[12] A. Bartoli, G. Davanzo, A. De Lorenzo, M. Mauri, E. Medvet, and E. Sorio. Automatic generation of regular expressions from examples with genetic programming. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 1477–1478. ACM, 2012.

[13] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1217–1230, 2016.

[14] C. Basile and A. Lioy. Analysis of application-layer filtering policies with application to http. *IEEE/ACM Transactions on Networking (TON)*, 23(1):28–41, 2015.

[15] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, Feb. 2012.

[16] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise dtds from xml data. In *Proceedings of the 32nd international conference on Very large data bases*, pages 115–126. VLDB Endowment, 2006.

[17] S. W. Boyd and A. D. Keromytis. Sqlrand: Preventing sql injection attacks. In *Applied Cryptography and Network Security*, pages 292–302. Springer, 2004.

[18] F. Brauer, R. Rieger, A. Mocan, and W. M. Barczynski. Enabling information extraction by inference of regular expressions from sample entities. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1285–1294. ACM, 2011.

[19] G. Buehrer, B. W. Weide, and P. A. Sivilotti. Using parse tree validation to prevent sql injection attacks. In *Proceedings of the 5th international workshop on Software engineering and middleware*, pages 106–113. ACM, 2005.

[20] M. Ceccato, C. D. Nguyen, D. Appelt, and 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*, pages 167–177. ACM, 2016.

[21] J. Clarke. *SQL injection attacks and defense*. Elsevier, 2009.

[22] F. Cuppens, N. Cuppens-Boulahia, J. Garcia-Alfaro, T. Moataz, and X. Rimasson. Handling stateful firewall anomalies. In *IFIP International Information Security Conference*, pages 174–186. Springer, 2012.

[23] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[24] C. L. Goues, T. Nguyen, S. Forrest, and W. Weimer. Genprog: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, Jan 2012.

[25] W. Halfond, J. Viegas, and A. Orso. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, volume 1, pages 13–15. IEEE, 2006.

[26] W. G. Halfond, S. Anand, and A. Orso. Precise interface identification to improve testing and analysis of web applications. In *Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA '09)*, pages 285–296, 2009.

[27] W. G. Halfond and A. Orso. Amnesia: analysis and monitoring for neutralizing sql-injection attacks. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 174–183. ACM, 2005.

[28] W. G. J. Halfond and A. Orso. Preventing SQL injection attacks using AMNESIA. In *Proceedings of the 28th International Conference on Software Engineering (ICSE' 06)*, pages 795–798, 2006.
[29] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.
[30] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, Dec. 2012.
[31] K. L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning dfa representations of http for protecting web applications. *Computer Networks*, 51(5):1239–1255, 2007.
[32] M. Kiani, A. Clark, and G. Mohay. Evaluation of anomaly based character distribution models in the detection of sql injection attacks. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 47–55. IEEE, 2008.
[33] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst. Automatic creation of SQL injection and cross-site scripting attacks. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*, pages 199–209, 2009.
[34] J. Knowles, L. Thiele, and E. Zitzler. A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Feb. 2006.
[35] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261. ACM, 2003.
[36] C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5):717–738, 2005.
[37] T. Krueger, C. Gehl, K. Rieck, and P. Laskov. Tokdoc: A self-healing web application firewall. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1846–1853. ACM, 2010.
[38] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Jagadish. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 21–30. Association for Computational Linguistics, 2008.
[39] A. Liu, Y. Yuan, D. Wijesekera, and A. Stavrou. Sqlprob: A proxy-based architecture towards preventing sql injection attacks. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 2054–2061, New York, NY, USA, 2009. ACM.
[40] S. Luke. *Essentials of metaheuristics*. Lulu Com, 2013.
[41] A. Mayer, A. Wool, and E. Ziskind. Offline firewall analysis. *International Journal of Information Security*, 5(3):125–144, 2006.
[42] M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.
[43] K. Murthy, P. Deepak, and P. M. Deshpande. Improving recall of regular expressions for information extraction. In *International Conference on Web Information Systems Engineering*, pages 455–467. Springer, 2012.
[44] A. J. Nebro, J. J. Durillo, and M. Vergne. Redesigning the jmetal multi-objective optimization framework. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 1093–1100, New York, NY, USA, 2015. ACM.
[45] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra. Semfix: Program repair via semantic analysis. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 772–781, Piscataway, NJ, USA, 2013. IEEE Press.
[46] Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang. The strength of random search on automated program repair. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 254–265, New York, NY, USA, 2014. ACM.
[47] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, Mar. 1986.
[48] W. Robertson, G. Vigna, C. Kruegel, R. A. Kemmerer, et al. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *NDSS*, 2006.
[49] L. K. Shar and H. B. K. Tan. Defeating sql injection. *Computer*, (3):69–77, 2013.
[50] L. K. Shar, H. B. K. Tan, and L. Briand. Mining sql injection and cross site scripting vulnerabilities using hybrid program analysis. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 642–651, 2013.
[51] Y. Song, A. D. Keromytis, and S. J. Stolfo. Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic. In *NDSS*, volume 9, pages 1–15. Citeseer, 2009.
[52] D. Stuttard and M. Pinto. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. John Wiley & Sons, 2011.
[53] F. Valeur, G. Vigna, C. Kruegel, and E. Kirda. An anomaly-driven reverse proxy for web applications. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 361–368. ACM, 2006.
[54] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. Management of an Academic HPC Cluster: The UL Experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*, pages 959–967, Bologna, Italy, July 2014. IEEE.
[55] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, pages 631–642. ACM, 2016.
[56] A. Wool. A quantitative study of firewall configuration errors. *Computer*, 37(6):62–67, 2004.
[57] A. Wool. Trends in firewall configuration errors: Measuring the holes in swiss cheese. *Internet Computing, IEEE*, 14(4):58–65, 2010.