DETECTING AND REMOVING MALWARE USING WEB APPLICATION FIREWALL

Alimulkader.TAjithkumar.E
CSE DepartmentCSE Department
FX Engineering collegeFX Engineering college
alimaulk@gmail.comantoajith833@gmail.com

Dr.C.Gopala Krishnan
Assistant professer
FX Engineering college
skywarekrish@gmail.com

Abstract

Although a large research effort on web application security has been going on for more than a decade, the security of web applications continues to be a challenging problemThe use of a combination of methods to discover vulnerabilities in source code with fewer false positivesThis approach brings together two approaches that are apparently orthogonal: humans coding the knowledge about vulnerabilities It was implemented in the WAP tool, and an experimental evaluation was performed with a large set of applications

Introduction

The Web Real-Time Communication (WebRTC) framework provides support for direct interactive communication using audio, video, text, collaboration, games, etc. between two peers browsers. This memo describes the media aspects of the WebRTC framework. WEB application firewalls (WAFs) protect enterprise web systems from malware attacks. WAFs inspect incoming HTTP messages and decide whether blocking or forwarding them to the web application. The decision is performed based on a set of rules, which are used to detect attack patterns. Since cyber-attacks are increasingly sophisticated, WAF rules tend to become complex and difficult to manually maintain and test. Therefore, automated testing techniques for WAFs are crucial to prevent malicious requests from reaching web applications and services

We focus our testing efforts on a common category of attacks, namely, SQL injections (SQLi). SQLi has received a lot of attention from academia as well as practitioners [7], [10], [14], [24]–[27], [32], [34], [45]. Yet the Open Web Application Security Project (OWASP) finds that the prevalence of SQLi vulnerabilities is common and the impact of a successful exploitation is severe [50]. While we assess our approach based on the example of SQLi, we believe that many of the principles of our methodology can be adapted to other forms of attacks.

Various techniques have been proposed in the literature to detect SQLi attacks based on a variety of approaches, including white-box testing [21], static analysis [20], model-based testing [30], and black-box testing [18]. However, such techniques present some limitations that may adversely impact their practical applicability as well as their vulnerability detection capability. For example, white-box testing techniques and static analysis tools require access to source code [33], which might not be possible when dealing with third-party components or industrial appliances, and are linked to specific programming languages [19]. Model-based testing techniques require models expressing the security policies or the implementation of

WAFs and the web application under test [30], which are often not available or very difficult to manually construct. Black-box testing strategies do not require models or the source code but they are less effective in detecting SQLi vulnerabilities.

We defined two variants of ML-Driven, namely, MLDriven D and ML-Driven B, that differ in the number of tests being selected for generating new tests (offsprings). The former variant selects fewer tests to evolve but generates more offsprings per selected test, thus increasing *exploitation* (deep search). The latter variant selects more tests to mutate, which results in a lower number of offsprings per selected test, hence increasing *exploration* (broad search). Our preliminary study with ModSecurity,[1] a popular open source WAF, has shown that both ML-Driven D and ML-Driven B are effective in generating a large number of distinct SQLi attacks passing thought the WAF. However, ML-Driven D performs better in the earlier stages of the search, while ML-Driven B outperforms it in the last part of the search, for reasons we will explain.

In the race against cyber-attacks, time is vital. Being able to learn and anticipate attacks that successfully bypass WAFs in a timely manner is critical. With more successful, distinct attacks being detected, the WAF administrator is in a better position to identify missed attack patterns and to devise patches that block all further attacks sharing the same patterns. Therefore, our goal is to devise a technique that can efficiently generate as many successful distinct attacks as possible. To this aim, in this paper, we extended our prior work and propose an adaptive variant of ML-Driven, namely, ML-Driven E (enhanced), that combines the strengths of ML-Driven D (deep search), and ML-Driven B (broad search) in an adaptive manner. In ML-Driven E, the number of offsprings is computed dynamically depending on the bypassing probability assigned to each selected test by the constructed classifier. Conversely, ML-Driven B and D generate a fixed number of offsprings per attack/test. Therefore, ML-Driven E is a more flexible approach that better balances eploration and exploitation over run time.

Moreover, we conducted a much larger empirical study with two popular WAFs that protect three open-source and 44 proprietary web services. The results show that the enhanced version of our technique (ML-Driven E) significantly outperforms the following:

1) its predecessors ML-Driven B and ML-Driven D;
2) a random test strategy, which serves as a baseline;
3) two state-of-the-art vulnerability detection tools, namely,
   SqlMapand WAF Testing Framework.

We also performed a qualitative analysis of the attacks generated by our approach and we found out that they enable the identification of attack patterns that are strongly associated with bypassing the WAFs, thus providing better support for improving the WAFs' rule set.

To summarize, the key contributions of this paper include the following.

1) Enhancing ML-Driven with an adaptive test selection and generation heuristic, which more effectively explores attack patterns with higher likelihood of bypassing the WAF. This enhanced ML-Driven variant is intended to replace its predecessors, leaving the practitioners with a single, yet the best, option.
2) Assessing the influence of the selected machine learning algorithm on the test results by comparing two alternative classification models, namely, RandomTree and RandomForest, which are both adapted to large numbers of features and datasets, but with

complementary advantages and drawbacks.

3) Extending the previous evaluation and conducting a largescale experiment on a proprietary WAF protecting a financial institution.
4) Comparing ML-Driven with SqlMapand WAF
   Testing Framework, which are state-of-the-art vulnerability detection tools.
5) A qualitative analysis showing that the additional distinct attacks found by ML-Driven E help security analysts design better patches compared to the other ML-Driven variants, RAN, and state-of-the-art vulnerability detection tools.

The remainder of this paper is structured as follows. Section II-A provides background information on WAFs as well as SQLi attacks and discusses related work. Section III details our approach followed by Section IV where we describe the design and procedure of our empirical evaluation. Section V describes the evaluation results and their implications regarding our research questions, while Section VI explains and illustrates how to use the generated attacks for repairing vulnerable WAFs. Further reflections on the results are provided in Section VII while Section VIII concludes this paper.

Related work

The delay-based controller running at the receiver. Starting from this analysis Proposed a control algorithm to dynamically set the thresholdPrevious research on ensuring the resilience of IT systems against malicious requests has focused on the testing of firewalls as well as input validation mechanisms.

Offutt *et al.* introduced the concept of bypass testing in which an application's input validation is tested for robustness and security [38]. Tests are generated to intentionally violate the clientside input checks and are then sent to the server application to test whether the input constraints are adequately evaluated. Liu and Kuan Tan [35] proposed an automated approach to recover an input validation model from program source code and formulated two coverage criteria for testing input validation based on the model. Desmet*et al.* [17] verify a given combination of a WAF and a web application for broken access control vulnerabilities, e.g., forceful browsing, by explicitly specifying the interactions of application components on the source code level and by applying static and dynamic verification to enforce only legal state transitions. In contrast, we propose a black-box technique that does not require access to source code or client-side input checks to generate test cases. In our approach, we use machine learning to identify the patterns recognized by the firewall as SQLi attacks and generate bypassing test cases that avoid those patterns.

Tripp *et al.* [48] proposed *XSS Analyzer*, a learning approach to web security testing. The authors tackle the problem of efficiently selecting from a comprehensive input space of attacks by learning from previous attack executions. Based on the performed learning, the selection of new attacks is adjusted to select attacks with a high probability of revealing a vulnerability. More specifically, *XSS Analyzer* generates attacks from a grammar and learns constraints that express which literals an attack cannot contain in order to evade detection. The authors find that *XSS Analyzer* outperforms a comparable state-of-theart algorithm, thus, suggesting that learning input constraints (a concept similar to the path conditions in ML-Driven) is effective to guide the test case generation.

In contrast to our work, *XSS Analyzer* applies learning to individual literals only while our approach also learns if a combination of literals is likely to bypass or be blocked.

Therefore,*XSS Analyzer* cannot capture more complex input constraints involving multiple literals simultaneously, e.g., an attack should contain literal *a*, but not *b* and *c* in order to evade detection. Furthermore, to analyze which literals in an attack are blocked, *XSS Analyzer* first splits each attack into its composing tokens; then, it resends each token to the target web application. Since multiple attacks can share the same tokens, *XSS Analyzer* sends each token only once to avoid performing the same analysis multiple times.

This procedure consumes a large quantity of HTTP requests. In contrast, ML-Driven does not require to resubmit individual slices, but learns path conditions solely from the previously executed test case and, thus, spends its test budget more efficiently. In addition, there are differences between *XSS Analyser*and ML-Driven in terms of objectives: The former addresses cross-site scripting sanitization in web applications, while the latter addresses the detection of SQLi attacks in WAFs.

Several grammar-based approaches exist in the literature for testing security properties of an application under test [37], [47]. Godefroid*et al.* [23] proposed white-box fuzzing, which starts by executing an application under test with a given well-formed input and uses symbolic execution to create input constraints when conditional statements are encountered on the execution path. Then, new inputs are created that exercise different execution paths by negating the previously collected constraints. The implementation of the approach found a critical memory corruption vulnerability in a file-processing application. In a follow-up work, the authors propose grammar-based white-box fuzzing [21], which addresses the generation of highly structured program inputs.

In this paper, well-formed inputs are generated from a grammar and the constraints created during execution are expressed as constraints on grammar tokens. To generate new inputs, a constraint solver searches the grammar for inputs that satisfy the constraints. The authors implemented their work in a tool named *SAGE* and found several security-related bugs [22]. In contrast to the mentioned work of Godefroid*et al.*, our work does not require access to the source code of the application under test, which is in many practical scenarios not available, but proposes a black-box approach based on machine learning to efficiently sample the input space defined by an attack grammar.

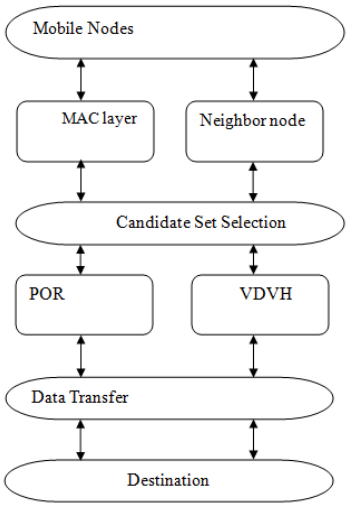
The topic of testing network firewalls has also been addressed by an abundant literature. Although network firewalls operate on a lower layer than application firewalls, which are our focus, they share some commonalities. Both use policies to decide which traffic is allowed to pass or should be rejected. Therefore, testing approaches to find flaws in network firewall policies might also be applicable to WAF policies. Brucker*et al.* [1] proposed a model-based testing approach which transforms a firewall policy into a normal form. Based on case studies, they found that this policy transformation increases the efficiency of test case generation by at least two orders of magnitude. Hwang *et al.* [29] defined structural coverage criteria of policies under test and developed a test generation technique based on constraint solving that tries to maximize structural coverage. Other research has focused on testing the firewalls implementation instead of policies. Al-Shaer*et al.* [3] developed a framework to automatically test if a policy is correctly enforced by a firewall. Therefore, the framework generates a set of policies as well as test traffic and checks whether the firewall handles the generated traffic correctly according to the generated policy. Some authors have proposed specification-based firewall testing. Jurjens and¨ Wimmel [30] proposed to formally model the tested firewall and to automatically derive test cases from the

formal specification. Senn*et al.* [43] proposed a formal language for specifying security policies and automatically generate test cases from formal policies to test the firewall. In contrast, in addition to targeting application firewalls, our approach does not rely in any models of security policies or the firewall under test, such formal models are rarely available in practice

Problem statement

We have also demonstrated the usefulness of mining more bypassing attacks in devising string patterns to fix WAFs. In our context, we experimented with RandomTree and RandomForest as machine classifiers for ML-Driven E. Since we show that the latter helps extract more path conditions that are useful in identifying patterns and fixing WAFs, we recommend the use of RandomForestSource code static analysis tools are a solution to find vulnerabilitiesThe insecurity of web applications

System architecture



Ideally, the congestion control algorithm should provide full link utilization while keeping zero queuing at steady state. A direct measurement of the queue length is not available at the end-points. Thus, the queue length must be estimated using the one-way delay or the RTT measurements which are however affected by several issues discussed

System modules

Multi-hop infrastructure-less transmission Network Creation Module

In the case of communication hole, a Virtual Destination-based Void Handling (VDVH) scheme is proposed; thistakes advantages of greedy forwarding (e.g., large progress per hop) and opportunistic routing to handle communication voids.

Position-based Opportunistic Routing (POR) Module

Position-based opportunistic routing mechanisms to select the forwarding candidates in the transmission range to effectively forward the data can be implemented. It can be deployed without complex modification to MAC protocol and achieve multiple receptions without losing the benefit of collision avoidance.

Memory Consumption and Duplicate Relaying

The forwarding candidate adopts the same forwarding scenario as the next hop node, which means it also calculates a candidate list, then in the worst case, the propagation area of a packet would cover the entire circle comprising the destination as the center and the radius could be as large as the distance between the source and the destination.

Selection and Prioritization of Forwarding Candidates

The nodes located in the forwarding area can be selected as the backup nodes. A node located in the forwarding area has to satisfy the following conditions: 1) it should make a

positive progress towards the destination and 2) its distance to the next hop node should not exceed half of the transmission range of a wireless node (i.e., R=2) so that all the forwarding candidates can be heared from one another. The nearer a node is to the destination, the higher priority it will get to forward the data. When a node sends , it selects the next hop forward as well as the forwarding candidates within its neighbors. Only the node specific in the candidate list can act as forwarding candidates. The lower the index of the node in the candidate list, the higher priority it will have.

Conclusion

The review we identified the different approaches for real time transmission or communication over the web. However methods are suffering from some limitations, which may be overcome by improving the existing methods. For the future we will suggest to present efficient method for real time communication over the web.

WAFs play an important role to protect online systems. The rising occurrence of new kinds of attacks and their increasing sophistication require that firewalls be updated and tested regularly, as otherwise attacks might remain undetected and reach the systems under protection.

We propose ML-Driven, a search-based approach that combines machine learning and EAs to automatically test the attack detection capabilities of WAFs. The approach automatically generates a diverse set of attacks, sends them to a WAF under test, and checks if they are correctly identified. By incrementally learning from the tests that are blocked or bypassing the firewall, our approach selects tests that exhibit string patterns with high bypassing probabilities (according the machine learning) and mutates them using an attack grammar designed to generate new and hopefully successful attacks. Identified bypassing attacks can be used to learn path conditions, which characterize successful attack patterns.

With such a set of bypassing attacks and path conditions that characterize them, a security expert can fix or fine-tune the WAF rules in order to block imminent SQLi attacks. In the attacker–defender war, time is vital. Being able to quickly learn and anticipate more attacks that can circumvent a firewall, in a timely manner, is very important to secure business data and services.

Though our approach was applied to SQLi attacks in this paper, it can be adapted to other forms of attacks by making use of other attack grammars targeting different types of vulnerabilities.

Our key contributions in this paper include the following:

1) enhancing our preliminary techniques by consolidating them and improving their performance;
2) comparing two different and adequate machine learning classifiers;
3) carrying out a large-scale evaluation on two popularWAFs; and
4) comparing our approach with state-of-the-art tools.

Evaluation results suggest that the performance of MLDriven(and its enhanced variant in particular) is effective at generating many undetected attacks and provides a good basis to identify attack patterns to protect against. Furthermost, it also fares significantly better than the best available tools.

In our future work, we will investigate automated approaches to generate effective patches for the WAF under test starting from the learned attack patterns. We reported on an initial attempt to automate the repairing process in a recent paper [11], where we generated patches

that block as many bypassing attacks as possible while limiting the blocking of legitimate inputs. Investigating further, more effective, repairing strategies that better exploit the attack patterns generated by ML-Driven E is a part of our future agenda. Finally, we plan to investigate various strategies (e.g., data resampling strategies, weighted training, and penalty-based training) to improve the effectiveness and the efficiency of ML-Driven by addressing the data imbalance problem

Referance

[1] A. D. Brucker, L. Brugger, P. Kearney, and B. Wolff, "Verified firewall¨ policy transformations for test case generation," in *Proc. Third Int. Conf. Softw. Testing, Verification Validation*, 2010, pp. 345–354.

[2] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos, "Management of an academic HPC cluster: The UL experience," in *Proc. Int. Conf. High Performance Comput. Simul.*, Bologna, Italy, Jul. 2014, pp. 959–967.

[3] E. Al-Shaer, A. El-Atawy, and T. Samak, "Automated pseudo-live testing of firewall configuration enforcement," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 3, pp. 302–314, Apr. 2009.

[4] S. Anand*et al.*, "An orchestrated survey of methodologies for automated software test case generation," *J. Syst. Softw.*, vol. 86, no. 8, pp. 1978– 2001, 2013.

[5] N. Antunes, N. Laranjeiro, M. Vieira, and H. Madeira, "Command injection vulnerability scanner for web services," 2009. [Online]. Available: http://eden.dei.uc.pt/ mvieira/

[6] N. Antunes, N. Laranjeiro, M. Vieira, and H. Madeira, "Effective detection of SQL/XPath injection vulnerabilities in web services," in *Proc. 6th IEEE Int. Conf. Services Comput.*, 2009, pp. 260–267.

[7] D. Appelt, N. Alshahwan, and L. Briand, "Assessing the impact of firewalls and database proxies on SQL injection testing," in *Proc. 1st Int. Workshop Future Internet Testing*, 2013, pp. 32–47.

[8] D. Appelt, A. Nguyen, Cu D. Panichella, and L. Briand, "Automated testing of web application firewalls," Univ. Luxembourg, Luxembourg City, Luxembourg, Tech. Rep. TR-SnT-2016-1, 2016.

[9] D. Appelt, C. D. Nguyen, and L. Briand, "Behind an application firewall, are we safe from sql injection attacks?" in *Proc. IEEE 8th Int. Conf. Softw. Testing, Verification Validation*, 2015, pp. 1–10