Towards a Web Application Attack Detection System based on Network Traffic and Log Classification

Rodrigo Branco[a], Vinicius Cogo[b] and Ibéria Medeiros[c]
LASIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, Portugal
fc54457@alunos.ciencias.ulisboa.pt, vvcogo@ciencias.ulisboa.pt, ivmedeiros@ciencias.ulisboa.pt

Keywords: Web Application Attacks, Netflows, Machine Learning, Natural Language Processing, Software Security

Abstract: Web applications are the preferred means of accessing online services. They have been built quickly and can be left with vulnerabilities due to human error and inexperience, making them vulnerable to attacks. As a result, security analysts must analyse and react to countless threats and alerts. Such alerts can not provide sufficient information about the attack performed on the web application, which is crucial for a correct risk assessment and remediation measures. Network Intrusion Detection Systems (NIDS) have been used as a primary defence mechanism against web attacks. However, HTTPS, a widely adopted protocol in web applications, encrypts traffic, hindering NIDS' efficiency in searching for network security threats and attacks. To enhance web application security, we present an approach that uses natural language processing (NLP) and machine learning (ML) algorithms to detect attacks through the analysis of network traffic (including HTTPS) and log-based payload contents. The approach employs anomaly detection by clustering netflows, and then NLP and supervised ML are used on the payload contents of anomalous netflows to identify attacks. Preliminary experiments have been made to detect SQL injection (SQLi), cross-site scripting (XSS), and directory traversal (DT) web attacks.

1 INTRODUCTION

The pressure to release new features to web applications often leads to prioritising quantity over quality, resulting in security breaches and creating vulnerabilities in web applications. As a result, such applications have been a target for attackers for the undue retrieval of sensitive and private data (e.g., bank accounts). According to OWASP[1], injection vulnerabilities are present in 94% of web applications, the three most common being SQLi, XSS, and DT[2].

The HTTPS protocol has been widely used in an attempt to mitigate web attacks, protecting data communication in web applications by encrypting network packets to guarantee data confidentiality. Although organisations have adopted it, the vulnerable code of the end-point software behind HTTPS remains unprotected and a target for malicious users.

Network Intrusion Detection Systems (NIDS) are the standard mechanisms to detect network attacks. To detect these types of web attacks, a Deep Packet Inspection (DPI) [Cheng et al., 2020] NIDS is required since such attacks are only detected by inspecting the packet content [Shema, 2010]. For instance, for SQLi it looks for malicious queries and related special characters [Pramod et al., 2015].

DPI-based NIDS can create network bottlenecks as it has to inspect every packet, which will be increased when traffic has to be decrypted first. To overcome this, NIDS can employ two different approaches: Netflows and intermediary proxies. Network flows (or Netflows) can solve the bottleneck problem by enabling the analysis of large amounts of traffic without individual inspection. A Netflow aggregates into a single flow a sequence of packets that share common characteristics and are observed in a given point by a period of time [Claise, 2004]. Although Netflows can work on HTTPS protocol, since there is no packet decryption, the detection of injection web attacks is unable over them since these do not comprise application-level data required to detect such attacks. Intermediary proxies focus on intercepting and decrypting the network traffic so that the DPI can analyse it [O'Neill et al., 2016]. Although they work on HTTPS protocol and allow the detec-

[a] https://orcid.org/0009-0008-6727-0642
[b] https://orcid.org/0000-0002-1299-8950
[c] https://orcid.org/0000-0003-4478-8680
[1] https://owasp.org/www-project-top-ten/
[2] https://owasp.org/Top10/A03_2021-Injection/

tion of these attacks, they raise security concerns because they increase the attack surface by adding more components with access to the communication parties' cryptographic keys.

NIDS systems can resort to ML algorithms for anomaly detection [Buczak and Guven, 2016], allowing the identification of patterns from attacks it knows, but their efficiency is limited when processing encrypted data or an anomaly is flagged, but not much relevant information is obtained of it. The Security Operation Center (SOC) team suffers from this lack of information while receiving numerous alerts. Hence, knowing the nature of the attack is highly beneficial to alert prioritisation, assessment and remediation measures. DPI solves the problem of lack of information, while anomaly detection works efficiently but does not give any helpful information. Therefore, a DPI-based NIDS system capable of functioning in both unencrypted and encrypted traffic, without additional data decryption, which provides useful information when detecting the attacks, is crucial for the security of modern web applications.

This work proposes an approach to protect web applications by identifying and classifying web attacks in HTTP(S) network traffic based on NLP and ML algorithms and log-based payload contents. Relying on unsupervised ML, the approach detects anomalous Netflows in HTTP(S) traffic, followed by a DPI using NLP and supervised ML that inspects the traffic information recorded in the web server logs and associated with the detected anomalous Netflows to search for and classify web attacks. The approach leverages the web server logs to overcome the encrypted data challenge without decrypting the packets. The approach uses a clustering algorithm to identify clusters with suspicious Netflows. Next, the data recorded in the web server log and associated with those Netflows is standardised into a vector by an NLP algorithm to be processed by different supervised ML algorithms. These last classify the vector into a web attack, and then a heuristic classifies the vector to decide which class of injection attack it belongs to.

Our initial implementation focuses on detecting SQLi, XSS, and DT attacks and handles HTTP(S) traffic. The supervised ML algorithms are trained with NLP vectors of attacks performed by two web scanners—Wapiti[3] and Burp[4]. Preliminary experiments aided in selecting and assessing the ML models used for classifying the web attacks. The results show that the web attacks' class can be discovered.

The main contributions of the paper are: (1) a DPI-based NIDS approach to detect anomalous HTTP(S) traffic and perform a log-based classification of the web attacks, resorting to NLP and ML algorithms (§4); (2) an implementation of the approach for detection of SQLi, XSS, and DT attacks (§5); (3) a preliminary evaluation of this approach (§6).

[3]https://wapiti-scanner.github.io/

[4]https://portswigger.net/burp

2 WEB INJECTION ATTACKS

Web injection attacks allow the injection of malicious code into an application for it to be executed.

SQL injection (SQLi) injects SQL code (e.g., `' OR 1=1;--`), provided by a user input field, in a SQL query. If the user input is not sanitised or verified, and the target query is not protected, the web application may contain an SQLi vulnerability.

A Cross-Site Scripting (XSS) exploitation occurs when an application uses untrusted data to generate a web page without preventing the execution of scripts provided from user inputs, like `<script>alert("XSS attack")</script>`.

A Directory Traversal (DT) attack allows the attacker to access arbitrary unprotected directories and files outside the web application's root directory. The user input `../../Windows/system.ini` tries to access the `system.ini` file and display it to the attacker if the system allows it to.

3 RELATED WORK

Several works in the literature study the detection of attacks in network traffic. We discuss those from the two methods most related to our proposed solution: the use of machine learning (ML) and natural language processing (NLP).

Deep Packet Inspection. Deep Packet Inspection (DPI) examines the contents of data packets and uses pattern-matching algorithms to identify potential threats and respond appropriately to them [El-Maghraby et al., 2017]. DPI is useful for detecting anomalous traffic, but its popularity has decreased with the widespread use of HTTPS, which encrypts packets for secure transactions. Proxy servers can be deployed, with its associated risks, in encrypted communications to cope with that [Jarmoc, 2012]. The network traffic is intercepted by a proxy, meaning that the client's request is captured and terminated to inspect the conversation in plain text. The intercepting process sends a second request to the server on behalf of the client. As a result, the client sends the data to the proxy, and the proxy resends the data to the server

and vice-versa. The proxy has access to the plain-text communication because it uses different encryption keys for each communication channel. However, this implementation has some risks, such as communication exposure, a single point of failure in the proxy, cypher strength decreasing since two communications possess two different encryption keys and transitive trust issues. We also intend to evaluate HTTPS traffic without intercepting and decrypting the traffic itself. We will resort to web server logs, which already record packet content data after decryption.

Netflows. Different Netflows datasets for supervised ML were created, with netflows labelled as benign or malicious [Sarhan et al., 2021]. Despite the good results obtained, their datasets do not reflect real scenarios, incurring in overfitting. Alternatively, unsupervised ML can detect anomalous traffic with Netflow analysis [Durão, 2022]. Their evaluation highlighted that the source IP address significantly differs between benign and malicious traffic—a distinction also observed in the real-world. Like the latter, our proposal addresses the lack of labelled data in real-world scenarios.

ML and NLP for Anomaly Detection. Many studies have delved into the realm of ML for cybersecurity [Xin et al., 2018]. Most approaches employ supervised ML or a blend of supervised and unsupervised ML. Others used NLP and deep learning (DL) to create an anomaly detection system for HTTP traffic [Seyyar et al., 2022]. They used the BERT algorithm to create a vector of the user input and DL to detect an attack. To the best of our knowledge, no other work has attempted to classify the category of an injection attack using NLP and ML, as we propose.

4 PROPOSED SOLUTION

4.1 Approach Overview

Our main goal is to create a NIDS to detect and classify web attacks on HTTP(S) traffic. We intend to detect anomalous network traffic by analysing Netflows and classifying their packet payloads in an injection attack class without payload decryption (for HTTPS). To do so, our approach uses unsupervised ML for Netflows, a log-based DPI with NLP, and supervised ML for web attack classification.

To achieve this goal, the approach must handle three aspects: (i) HTTPS traffic without requiring decryption to access the payload, (ii) minimise bottleneck caused by DPI, and (iii) correct identification of web attacks. To cope with (i), we propose using Netflows since they do not deal with the application-layer data (i.e., payloads). Instead, they contain data from the transport and network layers of the TCP/IP model. Therefore, monitoring HTTPS traffic over Netflows is feasible without decrypting the communication. Thus, we propose using Netflows for system monitoring and Netflow clustering through unsupervised ML to detect anomalous HTTP(S) Netflows. However, as DPI requires accessing packet payloads in clear text, the approach obtains such data from the web server logs, the end-point of communication where payloads are stored in clear text, even for HTTPS. For handling with (ii), the approach will only inspect the payloads of the detected anomalous Netflows, which will be retrieved from the web server's logs. Lastly, for aspect (iii), as anomaly detection methods may not provide the needed insights for classifying the vulnerability type [Seyyar et al., 2022, Gao et al., 2017] and the wide range of (sub)strings involved in the attacks are not suitable to a correct attack classification, we propose using NLP algorithms in a supervised classification model accomplished with a heuristic to classify the web attack.

Figure 1 shows the architecture of the proposed NIDS, comprising four modules: *Netflow Detector*, *Log Extractor*, *Data Preprocessor*, and *Web Attack Classifier*. While the first three modules are responsible for detecting suspicious Netflows, retrieving their payloads from web server logs, and preparing them to be delivered to the fourth module, the latter attempts to classify the received preprocessed payloads into a known web attack class, determined by NLP and ML algorithms and a heuristic. If this module determines that an attack occurred, an alarm is generated and sent to the SOC, which can verify its veracity and take a

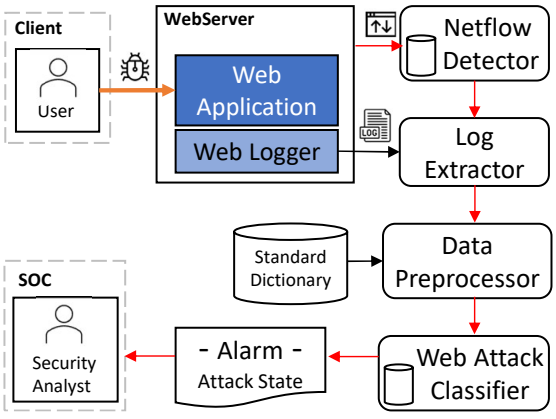

Figure 1: Architecture of the proposed NIDS to detect and classify web attacks over network traffic. The arrows in red represent the main workflow.

remediation measure. In the system, a Client is an entity that interacts with the web application (WebApp), which can be a normal user, an attacker or a web scanner. The following section details each module, with insights into their design.

4.2 Detailed Design Insights

4.2.1 Netflow Detector

The *Netflow Detector* aims to detect anomalous HTTP(S) Netflows associated with malicious activity (e.g., web scanners). This module captures the traffic the entities Client-WebApp generate. The traffic is obtained as PCAP and converted into a Netflow. This Netflow has information on the bi-directional traffic of Client-WebApp in a single flow that occurs in a period. The Netflow has a set of features in the network and transport layers of TCP/IP protocol, such as source and destination IP and ports, and others that are calculated, for instance, the number of packets and bytes exchanged in that period. These features are converted to numbers for the ML algorithms to use. Afterwards, Netflows are then clustered by an unsupervised ML algorithm, such as k-Means, DBSCAN, and Meanshift, thus generating distinct clusters with similar characteristics. Clusters containing scarce Netflows are deemed anomalous and marked to retrieve the associated logs from the Web Logger on the WebServer entity. At this point, the approach has significantly reduced the amount of data for further analysis, leaving only a few Netflows to be inspected.

4.2.2 Log Extractor

The *Log Extractor* retrieves the logs corresponding to these anomalous Netflows. It filters the Web Logger's logs and retrieves the data sent by the Client to the WebApp, the URL and the parameters involved in the attack, i.e., the page input fields.

It is presupposed that the Netflows possess only anomalous traffic, so the system assumes that all the payloads in these logs may be suspicious. Analysing the logs is not straightforward since POST requests are not logged by the web server by default, so we must force this logging process since these types of requests contain user input that is interesting to access (see Section 5 on how this feature was implemented).

The module, to get the correct log subset from the Web Logger, uses for matching keys the *datetime*, *source IP*, and *destination IP*, as these features are also present in the Netflows. It also considers the *datetime* deviation between the time when traffic is captured and the data is logged. So, it manages this deviation through a threshold to ensure it gets all the logs related to the identified anomalous traffic. It also discards the logs that do not have a payload included. The log data comes in chunks, so the module rebuilds these logs, identifying the GET and POST requests, joining all the parts, and rebuilding the payloads. The final output of this module is a steady stream of logs with all the requests and their payloads.

4.2.3 Data Preprocessor

This module takes suspicious logs and prepares payloads for classification. The preparation process involves capturing the essence of an injection attack, enabling the system to preprocess the payload and quickly identify and classify an attack.

To accomplish this, this module parses the previous payloads to get the user values, i.e., not the remaining features included in the URL (e.g., parameters or input fields). This way, we can generalise the system to any web application since, by removing the input fields, the system does not need any knowledge of the web application it monitors. Next, it removes the dummy values among the user values retrieved. Dummy values are strings without any metacharacters contained in an attack. Note that, web scanners and attackers do not inject malicious inputs on all input fields included in a URL The system can detect these dummy values because when the Web Logger saves the payloads, it encodes all the metacharacters into their HTML URL representation. For example, the metacharacter *slash* ("/") is encoded into "`%2F`".

With only the suspicious user values, the system standardises them using a *Standard Dictionary*. This dictionary contains all the known malicious XSS events and tags, SQL commands and some recurrent substrings used in DT attacks. It catches these kinds of data and replaces them with tokens for further analysis. This dictionary can be extended with more attacks of the same or a new type.

Listing 1 exemplifies the data treatment pipeline when processing a payload containing an SQLi attack. The system isolates the user values and removes the dummy values, keeping only the suspicious instances. Then, the instances are standardised.

```
login=hello&password=my_password&num_secure=158%22%29+and
      +82%3d82+and+%28%2216%22%3d%2216&submit=submit
                              ↓
["hello", "my_password", "158%22%29+and+82%3d82+and
      +%28%2216%22%3d%2216", "submit" ]
                              ↓
        158%22%29+and+82%3d82+and+%28%2216%22%3d%2216
                              ↓
NUM %22 %29 AND NUM %3d NUM AND %28 %22 NUM %22 %3d %22 NUM
```

Listing 1: An example of a complete payload preprocessing

The final output of this module is the list of standardised suspicious instances detected on the logs.

4.2.4 Web Attack Classifier

The *Web Attack Classifier* module is designed to classify suspicious user inputs into one of three main types of injection attacks, namely SQLi, XSS, and DT. The module uses NLP to create a vector of the suspicious string, which is processed by up to four supervised ML models: a multi-classification model (that recognises three classes) and three binary classification models (each one for a type of injection attack). Once they make their predictions, we iterate over these probabilities and, based on a modular heuristic, classify the suspicious user input and generate an alarm if needed.

Firstly, the NLP algorithm vectorises each attack string (i.e., the suspicious user values extracted by the previous module) into a vector. Following the example of the standardised instance at the bottom of Listing 1, a resulting vector from the vectorisation process is depicted in Listing 2.

```
[0.0399 0.1332 -0.0544 ... 0.0021 0.9311 -0.1202 -0.2332]
```

Listing 2: Resulting vector of the instance after the vectorisation of the standardised instance of Listing 1

Subsequently, this vector is used to feed four main classification ML models—the three binary and the multi-class—, all trained to classify SQLi, XSS, and DT attacks. Therefore, each model takes the vector and classifies it into an attack class. A significant advantage of our system is that it is highly modular, which means we can use any ML model; it all depends on the nature of the data. We can also add more binary ML models for other types of injection attacks and update the multi-classification ML model with newer classes.

Once all models have classified the vector, the module gets their resulting probabilities and calculates the final prediction by application of a heuristic. The heuristic we propose can give one of four main states to a vector:

- *ATT*: the heuristic considers the instance an `ATTack` and correctly predicts the class (SQLi, XSS, or DT);
- *UNC*: the heuristic considers the instance belonging to one of the three classes it knows but does not have certainty of its class (`UNCertain-class`);
- *AON*: when the heuristic cannot guarantee the instance belongs to an attack class it knows (`Attack-Or-Not`);
- *NAT*: the heuristic is certain that the instance is not of any attack class (`Non-ATtack`) it knows;

It is important to note that the system cannot guarantee that an instance is benign just because it is classified as *NAT*. *NAT* is a state used for instances the system has identified as not belonging to any known classes. On the other hand, *AON* is used for instances the system is uncertain about, and it cannot determine if they belong to the known classes. Given a multi-probability tuple composed of the probability of an instance belonging to each class c_n, being $1 <= n < \infty$ performed by a multi-classification model: $v_m = (c_1, .., c_n)$, and a threshold t_1 being $\frac{1}{n} < t_1 <= 1$. As seen in Equation 1, if $max(c_1, .., c_n) > t_1$, then the vector is stated with a temporary state $ts(v_m)$ of `DEFined` (*DEF*), otherwise it is considered `UNDefined` (*UND*).

$$ts(v_m) = \begin{cases} DEF, & \text{if } max(v_m) > t_1 \\ UND, & otherwise \end{cases} \quad (1)$$

After evaluating the multi-classification probability, we evaluate the binary classification and perform the final classification. Given the binary probability tuple composed with the positive label p of each binary classification model corresponding to each class c_n: $v_b = (p_1, .., p_n)$, a threshold t_2 being $0.5 < t_2 <= 1$, the function $class(x)$ that returns the respective class c_n of $max(v_m)$. Equation 2 details $s(v_b, ts)$ that calculates the final state given v_b and ts.

$$s(v_b, ts) = \begin{cases} ATT, & \text{if } ts = DEF \wedge max(v_b) > t_2 \wedge \\ & class(max(v_b)) = class(max(v_m)) \\ UNC, & \text{if } ts = DEF \wedge max(v_b) > t_2 \wedge \\ & class(max(v_b)) \neq class(max(v_m)) \\ AON, & \text{if } (ts = DEF \wedge max(v_b) <= t_2) \vee \\ & (max(v_b) > t_2 \wedge ts = UND) \\ NAT, & \text{if } ts = UND \wedge max(v_b) < t_2 \end{cases} \quad (2)$$

The state, together with all the relevant information, like the class of the web attack and the payload used, is then given to the SOC team, which will try to behave accordingly. If the threat is classified as *ATT*, the SOC team can identify its nature and respond accordingly. For instance, if the attack is an SQLi, the team can restrict database access; if it is an XSS, they can look for hidden scripts in the database.

5 IMPLEMENTATION

This Section outlines decisions for the system's current implementation, including details on the con-

trolled network used and how we generate and extract logs. We also discuss standardising instances and the ML models chosen.

Development Scenario. We implemented the system in a controlled network to test different scenarios, collect data for building models, and make implementation decisions. We used two Virtual Machines (VM) implemented with Oracle VM VirtualBox version 6.1; one hosted the WebServer that runs the Apache Server, and the other was the client. The network allowed for testing vulnerable applications, using web scanners, and having total traffic control. We used BWAPP[5] for testing, a well-known vulnerable PHP web application with diverse vulnerabilities.

Netflow Detector and Log Extractor. The Netflow Detector module captures the network traffic using *Wireshark*[6]. The Netflows are saved in PCAP format via *tcpdump*[7], which are then processed using the *SiLK*[8] framework, which allows the customisation of the data fields for ML applications. In this implementation, we used the *rwp2yaf2silk* tool to generate the Netflow file with a *.rw* extension that was then converted to a CSV file using the *rwcut* tool.

The Apache server's logs, which the Log Extractor module uses to retrieve the suspicious logs, are enhanced with the *mod_dumpio* add-on that allows us to record the POST requests beyond the GET requests. The error log format is customised with *trace7* log level. The Log Extractor module uses Python v3.11 scripts to filter and parse logs with payloads and GET/POST requests.

Data Preprocessor. This module is a script built in Python v3.11. The script extracts the user strings by removing the input fields and filtering out dummy values. Some metacharacters are converted into their decimal representation since they are not considered malicious (e.g., "!"), and the user does not use them with any harmful intent. Then, standardisation is performed using a standard dictionary. The dictionary includes SQL commands, XSS events and tags, recurrent substrings used in a DT attack, and their corresponding tokens. The standardisation result is a sentence for each instance (see Section 4.2.3).

Web Attack Classifier. Our current implementation uses two supervised ML algorithms. The Support Vector Machine (SVM) algorithm for binary classifiers and the Logistic Regression algorithm for multi-classification, both from the scikit-learn library[9]. We opted for these models as they outperformed others we experienced, such as K-Nearest Neighbor (KNN), Random Forest, and Naive-Bayes classifiers. They have been tested for binary and multi-classification, proving to be worse than the two selected. Section 6.1 details the evaluation performed.

The implementation also utilises Doc2Vec from the *gensim* library[10] to convert input data into a numerical vector form, allowing the models to learn from the data and ultimately classify it depending on its class of vulnerability. Combining these models provides a robust and accurate means of detecting and preventing potential security threats in the input data.

The heuristic is a script developed in Python v3.11 that receives the probabilities in a list format, iterates over them, and outputs the final state of the vector in analysis (following the definition in Section 4.2.4).

6 PRELIMINARY RESULTS

This Section presents a preliminary evaluation of the system, aiming to answer the following two questions: $Q1$. Can the system correctly classify an attack instance using NLP and ML? $Q2$. Can the proposed heuristic correctly classify the samples based on the probabilities given by the ML models?

After presenting the experimental environment and methodology, Sections 6.1 and 6.2 discuss the results we obtained from the experiments.

Experimental Environment. We used the Apache web server to run the BWAPP web application, and the scanners Burp v2023.9.10 and Wapiti v3.1.3 to attack BWAPP from a client host with only one type of injection attack (e.g., SQLi) per flow. This method enabled us to conveniently gather the payloads to be processed in the Data Preprocessor.

We obtained 64 SQLi, 135 DT, and 38 XSS instances (i.e., attacks) with the Wapiti scanner and 694 SQLi instances with the Burp scanner, thus totalling 758 SQLi, 135 DT, and 38 XSS instances.

Since we are in the presence of an unbalanced number of instances per web attack class, which will generate biased results, we performed an augmentation of the DT and XSS classes and a random undersampling of the SQLi class. To do so, we used the SMOTE and RandomUnderSampler, respectively,

[5]http://www.itsecgames.com/

[6]https://www.wireshark.org/

[7]https://www.tcpdump.org/manpages/tcpdump.1.html

[8]https://tools.netsa.cert.org/silk/index.html

[9]https://scikit-learn.org/stable/

[10]https://radimrehurek.com/gensim/index.html

from imbalanced-learn[11]. We obtained a final dataset with 379 SQLi, 334 DT, and 238 XSS instances.

6.1 Classification Models Tuning and Selection

This Section focused on answering Question *Q1* by searching for the best ML algorithm for each model, one multi-class classifier and three binary classifiers. Using a generic Doc2Vec NLP model with 30 epochs, we established a vocabulary using 100% of our dataset and inferred the vectors.

As our dataset only contains attacks of three types (SQLi, DT, and XSS), the classes of the multi-class model are defined for these three attacks and the classes of the binary models are defined as follows: the positive class is the attack's type we want the model learns to classify, and the negative class will contain the instances of the other two attack's types. Based on this, we have four datasets: the initial dataset that will be used with the multi-class model and three binary datasets, one for each type of attack.

We evaluated five ML algorithms for both binary and multi-class models, including SVM, Logistic Regression, KNN, Random Forest, and Naive Bayes. We split each dataset into 70% for training and 30% for testing. With the training datasets, we performed model tuning and selection using grid search 10-fold cross-validation to find the best hyperparameters for each model.

Finally, we evaluated the models with the test set, where their metrics are presented in Tables 1 and 2. The SVM algorithm for binary classification had the best results in all classes, and the Logistic Regression algorithm was the best for the multi-class model. So, these two are the algorithms we employ.

6.2 Heuristic Classifier

This Section focused on answering Question *Q2* by using the probability prediction performed by the models selected in the previous Section. We divided this Section into two evaluations: (1) Assessing whether the heuristic can correctly classify web attacks that belong to one of the three classes that the system knows; and (2) Assessing whether the heuristic can correctly classify web attacks that do not belong to any of the three classes that the system knows.

System-known Classes Injection Attack Classification. Using the trained models and the same test set, we ran the models and provided the heuristic with the probabilities resulting from each test instance. We evaluated the heuristic with different thresholds to reason about the results and conclude the best threshold ranges. As stated in Section 4.2.4, heuristic has two thresholds: t_1 that controls multi-class classification probability and t_2 that controls the binary ones.

Table 1: Metric results for Logistic Regression multi-class classification ML model.

Accuracy	Precision	Recall	F1-Score
0.9790	0.9798	0.9790	0.9789

Table 2: Metric results for SVM binary-classification ML model for each class of injection attack.

Class	Accuracy	Precision	Recall	F1-Score
SQLi	0.9755	0.9765	0.9755	0.9754
XSS	0.9790	0.9807	0.9790	0.9793
DT	0.9895	0.9898	0.9895	0.9895

Table 3: Heuristic results for the different threshold configurations to classify the test set.

t_1	t_2	*ATT*	*UNC*	*AON*	*NAT*	*FP*
0.45	0.5	**272**	5	3	0	6
0.5	0.75	266	2	11	2	5
0.75	0.5	262	0	17	3	**3**
0.65	0.7	265	0	10	7	4

We tested 4 configurations for thresholds t_1 and t_2. Table 3 shows the results, where they indicate that with the best configuration $t_1 = 0.45$ and $t_2 = 0.5$, the heuristic can correctly classify 272 instances out of the 286 instances in the test set. However, this configuration generates the most false positives (*FP*), but, 6 FPs in 286 instances is an acceptable number.

The worst configuration was with $t_1 = 0.75$ and $t_2 = 0.5$, which only correctly classified 262 samples out of the 286. It was also the configuration that classified the most samples as `Attack-Or-Not` (*AON*).

From these preliminary and promising results, we can already draw some conclusions on the thresholds:

- High t_1 leads to less `UNCertain-attack` (*UNC*) and more `Attack-Or-Not` (*AON*).
- High t_2 leads to less `Attack-Or-Not` (*AON*) and less `Non-ATtack` (*NAT*).

CRLF-Injection Payload Classification. For the second evaluation, we used an open-source collection of 61 CRLF-Injection (CRLFi) web attack payloads[12] to provide the system attacks it does not know. We expect all the CRLFi samples to be classified as *NAT*, meaning the system is sure that the sample is not an SQLi, XSS, or DT. We trained the models with the complete dataset presented in Section 6 and fed the

[11] https://imbalanced-learn.org/stable/index.html

[12] https://github.com/cujanovic/CRLF-Injection-Payloads/blob/master/CRLF-payloads.txt

Table 4: Heuristic results for the different threshold configurations to classify the CRLFi payloads.

t_1	t_2	*ATT*	*UNC*	*AON*	*NAT*	*FP*
0.45	0.5	0	18	23	16	4
0.5	0.75	0	10	23	25	3
0.75	0.5	0	0	33	25	3
0.65	0.7	0	0	4	**55**	**2**

heuristic with the resulting probabilities of these 61 CRLFi attacks.

Using the same threshold configurations of Section 6.2, Table 4 presents the outcomes. The results indicate that the heuristic correctly did not classify any of CRLFi as an `ATTack` (*ATT*) and can accurately classify most samples as `Non-ATtack` (*NAT*) when both thresholds are higher than 0.6. When one of the thresholds is 0.5, and the other is any value, the heuristic starts with doubts of *NAT*, splitting the classification by `Attack-Or-Not` (*AON*), `UNCertain-attack` (*UNC*), and `Not-ATtack` (*NAT*), but never by *ATT*. Such division is explained by some malicious CRLFi code that could be similar to some examples of DT, and the resulting probability in this attack class can rise to 0.6.

7 CONCLUSION

This paper introduced a system for web application security. The system detects web application attacks using unsupervised ML to identify suspicious HTTP(S) Netflows, and, for these, it analyses their payloads with NLP and supervised ML to find web attacks and classify them based on their injection attack classification. Currently, the system uses Logistic Regression and SVM algorithms to categorise attacks as SQLi, XSS, and DT. We plan to expand the system to other attack classes, with other ML algorithms, namely NLP and deep learning, and the dataset using more web scanners without an augmentation process. We aim to improve the heuristic and develop a recommendation system based on similarity search. We are improving the standardisation and exploring the complexity of web attacks, which may require using regular expressions to address the similarity problem.

ACKNOWLEDGMENTS. This work was supported by FCT through the LASIGE Research Unit, ref. UIDB/00408/2020 (https://doi.org/10.54499/UIDB/00408/2020) and ref. UIDP/00408/2020 (https://doi.org/10.54499/UIDP/00408/2020)

REFERENCES

Buczak, A. L. and Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176.

Cheng, Z., Beshley, M., Beshley, H., Kochan, O., and Urikova, O. (2020). Development of Deep Packet Inspection System for Network Traffic Analysis and Intrusion Detection. In *Proc. of the IEEE International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering*, pages 877–881.

Claise, B. (2004). Cisco systems netflow services export version 9. RFC 3954, RFC Editor. http://www.rfc-editor.org/rfc/rfc3954.txt [Accessed: 18/03/2024].

Durão, N. P. G. C. (2022). Discovery of Web Attacks by Inspecting HTTPS Network Traffic with Machine Learning and Similarity Search. Master's thesis, Faculdade de Ciências da Universidade de Lisboa.

El-Maghraby, R. T., Abd Elazim, N. M., and Bahaa-Eldin, A. M. (2017). A survey on deep packet inspection. In *Proc. of the 12th Int. Conf. on Computer Engineering and Systems (ICCES)*, pages 188–197.

Gao, Y., Ma, Y., and Li, D. (2017). Anomaly detection of malicious users' behaviors for web applications based on web logs. In *Proc. of the 17th IEEE Int. Conf. on Communication Technology (ICCT)*, pages 1352–1355.

Jarmoc, J. (2012). SSL/TLS interception proxies and transitive trust. *Black Hat Europe*.

O'Neill, M., Ruoti, S., Seamons, K., and Zappala, D. (2016). TLS Proxies: Friend or Foe? In *Proc. of the 16th ACM Internet Measurement Conference (IMC)*, page 551–557.

Pramod, A., Ghosh, A., Mohan, A., Shrivastava, M., and Shettar, R. (2015). SQLi detection system for a safer web application. In *Proc. of the IEEE Int. Advance Computing Conf. (IACC)*, pages 237–240.

Sarhan, M., Layeghy, S., Moustafa, N., and Portmann, M. (2021). Netflow datasets for machine learning-based network intrusion detection systems. In *Big Data Technologies and Applications*, pages 117–135. Springer.

Seyyar, Y. E., Yavuz, A. G., and Ünver, H. M. (2022). An Attack Detection Framework Based on BERT and Deep Learning. *IEEE Access*, 10:68633–68644.

Shema, M. (2010). Chapter 4 - server misconfiguration and predictable pages. In Shema, M., editor, *Seven Deadliest Web Application Attacks*, pages 71–90. Syngress.

Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., and Wang, C. (2018). Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381.