Achieving Interpretable DL-based Web Attack Detection through Malicious Payload Localization

Peiyang Li*†, Fukun Mei*, Ye Wang*, Zhuotao Liu*, Ke Xu‡§, Chao Shen¶, Qian Wang‖, Qi Li*§✉
*INSC and the State Key Laboratory of Internet Architecture, Tsinghua University, †Ant Group
‡DCST and the State Key Laboratory of Internet Architecture, Tsinghua University
§Zhongguancun Laboratory, ¶Xi'an Jiaotong University, ‖Wuhan University
peiyangli.20@gmail.com, {mfk25, wangye22}@mails.tsinghua.edu.cn, {zhuotaoliu, xuke}@tsinghua.edu.cn,
chaoshen@mail.xjtu.edu.cn, qianwang@whu.edu.cn, qli01@tsinghua.edu.cn

***Abstract*—Web attacks pose a significant threat to Web applications. While deep learning-based systems have emerged as promising solutions for detecting Web attacks, the lack of interpretability hinders their deployment in production. Existing interpretability methods are unable to explain Web attacks because they overlook the structure information of HTTP requests. They merely identify some important features, which are not understandable by security operators and fail to guide them toward effective responses.**

In this paper, we propose WebSpotter that achieves interpretable Web attack detection, which enhances existing deep learning-based detection methods by locating malicious payloads of the HTTP requests. It is inspired by the observation that malicious payloads often have a significant impact on the predictions of detection models. WebSpotter identifies the importance of each field of HTTP requests, and then utilizes a machine learning model to learn the correlation between the importance and malicious payloads. In addition, we demonstrate how WebSpotter can assist security operators in mitigating attacks by automatically generating WAF rules. Extensive evaluations on two public datasets and our newly constructed dataset demonstrate that WebSpotter significantly outperforms existing methods, achieving at least a 22% improvement in localization accuracy compared to baselines. We also conduct evaluations on real-world attacks collected from CVEs and real-world Web applications to illustrate the effectiveness of WebSpotter in practical scenarios.

I. Introduction

Web applications are vulnerable to various Web attacks. Attackers can insert malicious payloads into specific fields of HTTP requests to launch the attacks, resulting in severe consequences such as malicious code execution. Recently, a number of methods [1], [2], [3] propose to apply deep learning (DL) to detect Web attacks. These methods achieve various advantages over conventional rule-based Web Application Firewalls (WAFs), such as eliminating manual rule configurations and detecting unknown attacks [4].

However, existing DL-based methods only output a label (e.g., normal or abnormal) and cannot produce interpretable results. This causes practical concerns for security operators when deploying these systems in production. In particular, security operators typically need to analyze the detected attacks to take appropriate actions, such as configuring new firewall rules. Uninterpretable detection results require them to spend a significant amount of time and manual effort in analyzing the attacks. Moreover, the lack of interpretability could cause attack desensitization [5], i.e., security operators lose their trust in the effectiveness of the detection.

The interpretability of DL models has been extensively studied [6], [7], [8], [9], [10], [11], [12]. Nevertheless, these methods are not applicable to Web attack detection because they are unable to consider the structural information embedded in HTTP requests. Specifically, HTTP requests have well-defined structures and contain various fields, including the request method, multiple path segments, query parameters, etc. Existing interpretability methods overlook these structures and only analyze the numerical features of DL models, which have very limited contributions to explain the maliciousness of requests.

In this paper, we propose WebSpotter, a novel framework that enhances existing DL-based Web attack detection models by enabling detection interpretability. WebSpotter achieves interpretable detection by identifying the locations of malicious payloads within attack requests, thereby allowing security operators to gain insights into attack behaviors and apply targeted defense strategies. We observe that malicious payloads often exert a considerable influence on the predictions of detection models. Building upon this insight, WebSpotter utilizes gradient-based analysis to quantify the importance of individual fields within HTTP requests, as perceived by the detection models. Subsequently, we leverage a machine learning model to learn the correlation between these importance scores and the presence of malicious payloads. Note that, WebSpotter is orthogonal to existing DL-based Web detection methods and can be applied to enhance them.

At a high level, WebSpotter consists of three modules. First, the embedding attribution module is responsible for assessing the importance of embedding vectors in existing Web attack detection models. It can reflect the contribution of each embedding value to the model predictions through gradient analysis and mitigate the interference of gradient noises through interpolation. Second, the HTTP-structure alignment module evaluates the importance of each field in a request

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.231029
www.ndss-symposium.org

based on the embedding importance. It consolidates embedding importance across embedding dimensions to obtain token importance and derives importance of HTTP fields by identifying the association between these fields and tokens. Third, the malicious payload localization module establishes a robust connection between the location of the malicious payload and the importance of each HTTP field. This module constructs specialized location features by integrating the textual semantics with the importance of HTTP fields, and trains a lightweight localization model using a small number of location-labeled attack requests, which effectively mitigates the problem of spurious correlations.

We evaluate WebSpotter using two public datasets and one newly constructed dataset. To measure the localization accuracy, we annotate these datasets with the location labels of malicious payloads. We conduct a comparative analysis of WebSpotter against four state-of-the-art methods specifically designed for localization tasks. Experimental results show that WebSpotter significantly outperforms existing methods, achieving at least a 22% improvement in localization accuracy compared to all baselines. We perform an ablation study to assess the impact of each module in WebSpotter on overall performance. We also test WebSpotter on real-world scenarios. Specifically, we collect attacks from i) CVEs reported between 2021 and 2024 and ii) two real-world Web applications, and perform evaluations on these attacks. Our method achieves an F1 score of 0.945 at most on these real-world attacks, demonstrating its effectiveness in practical scenarios.

To further demonstrate how WebSpotter can assist security operators in mitigating attacks, we introduce a method for automated generation of WAF rules. We consider the scenario where security operators need to generate new WAF rules based on the detection results of DL-based Web attack detection methods to enhance the capabilities of existing WAFs. Specifically, we first categorize the attacks with similar semantics based on the localization results produced by WebSpotter, and derive effective rules for each attack group by extracting the common payload pattern. We evaluate the effectiveness of the generated rules on a widely used WAF. Experimental results show that our generated rules improve the attack detection rate of this WAF by 26%.

Our contributions are summarized below:

- We design WebSpotter, a framework to achieve interpretable deep learning-based Web attack detection by localizing malicious payloads within HTTP requests. WebSpotter obtains the importance of each field within HTTP requests from the detection model according to gradient analysis, and employs a lightweight localization model to identify malicious payloads.
- We build a new Web attack detection dataset. Compared with public datasets, our dataset (i) incorporates contemporary attack techniques, (ii) provides precise location labels of malicious payloads and (iii) includes an additional out-of-distribution testing set, which enables us to assess the performance of WebSpotter more comprehensively.
- We evaluate WebSpotter on two public datasets and our newly constructed dataset. The results demonstrate that WebSpotter significantly outperforms existing baselines. We also conduct evaluations on real-world attacks collected from CVEs and real-world Web applications, showing the effectiveness of WebSpotter in practical scenarios.
- We demonstrate how WebSpotter can assist security operators in responding to attacks. We propose a method to automatically generate WAF rules based on the localization results produced by WebSpotter. We also conduct an empirical evaluation leveraging a popular WAF, and the results show that our generated rules can improve the attack detection rate of the WAF by 26%.

```
POST /tienda1/miembros/editar.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.
8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,te
xt/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Host: localhost:8080
Cookie: JSESSIONID=D8B2A8778565BCF42799D84360295207
Content-Type: application/x-www-form-urlencoded
Content-Length: 335
…

modo=registro&login=sawyers&password=encUmb2ad3%27%3B+DRO
P+TABLE+usuarios&nombre=Max&apellidos=Sj%F6strand+Ampurias&
email=tamburi%40hispadis.ar&dni=30293636Z&direccion=C%2F+Virge
n+De+Las+Angustias+122+4%3FA&ciudad=Fuentespina&cp=48220&pr
ovincia=Asturias&ntc=8898020239162472&B1=Registrar
```

Fig. 1: Example of an attack request with malicious payload locations indicated in red.

II. Background

A. Web Attacks

Web attacks target Web applications deployed on servers. Attackers exploit vulnerabilities in Web applications by crafting malicious payloads in specific fields of HTTP requests. These attacks can take various forms, such as SQL injection and cross-site scripting (XSS). Figure 1 shows an example of SQL injection attacks sourced from the CSIC dataset. In this request, the malicious payload is located at the `password` parameter of the request body. The attacker inserted malicious SQL code into this field, which decodes to "`encUmb2ad3'; DROP TABLE usuarios`". Note that the malicious payload does not necessarily reside in only one location within the HTTP request. Many attacks require multiple fields to work together. Take the Jorani RCE attack (CVE-2023-26469 [13]) as an example: the attacker needs to perform directory traversal using the `language` parameter to access unauthorized directories, while simultaneously injecting malicious PHP code through the `login` field.

B. Deep learning for Web Attack Detection

Existing deep learning-based Web attack detection methods aim to construct a function that maps input HTTP requests to

discrete labels, such as benign or anomalous. These methods typically consist of three key modules: preprocessing, embedding, and detection. The preprocessing module is mainly responsible for segmenting the raw HTTP requests into a sequence of tokens to facilitate semantic analysis. The embedding module is responsible for converting text into numerical feature vectors. Each token is mapped to feature vectors based on an embedding matrix whose parameters are learned during model training. Finally, the detection module trains a neural network model, such as TextCNN [14], FastText [15], and Bi-LSTM [16], to learn the patterns in the training data. The trained model is then deployed for detecting Web attacks.

Current methods only classify an entire request as either an anomaly or a specific type of attacks. In contrast, we focus on pinpointing the exact location of the malicious payload within the request. For example, with the attack request illustrated in Figure 1, our approach would alert the security operator that the attack payload is specifically located within the `password` parameter of the request body. A formal definition of our objective will be presented in Section III.

III. Problem Definition

Our goal is to achieve interpretable DL-based Web attack detection by locating malicious payloads in HTTP requests. To establish a formal definition for "locating malicious payloads", we begin by introducing the concept of *minimal semantic units* (MSUs) of an HTTP request.

Definition 1 (MSU). *An HTTP request can be decomposed into multiple units based on the HTTP protocol specification [17], where each unit represents an indivisible piece of semantic information. These units, referred to as minimal semantic units (MSUs), are defined as follows: (i) The request method itself is considered a single unit. (ii) For the URL in the request line, each path segment and query [18] are treated as an individual unit. (iii) Each HTTP header is treated as an individual unit. (iv) For the body field, if its content type follows the key-value structure or can be converted into this structure, each key-value pair constitutes a separate unit.*

The above definition considers the request body that follows the key-value structure (e.g., when the content type is "application/x-www-form-urlencoded"), or can be transformed into such a structure, such as by flattening the JSON object [19] to obtain key-value pairs. This is a common practice in existing detection systems [4], [14] and can cover the majority of Web attacks [20]. For other formats, the body field is not analyzed. The reason is that the HTTP protocol does not strictly define body formats. For example, an application could transmit an encrypted binary stream in the body field, making it impractical to extract meaningful semantics.

After defining the MSUs of an HTTP request, locating the malicious payloads is regarded as a retrieval task, i.e., determining which MSUs contain the malicious payload. Formally, let the input space be $\mathcal{U}$, where each $u \in \mathcal{U}$ is an MSU list of an HTTP request. Specifically, $u = [u_1, u_2, \ldots, u_L]$, where u_i represents the i-th MSU in the HTTP request, and L is the length of the MSU list. Our goal is to build a function g that maps each $u \in \mathcal{U}$ to the location output vector $y = [y_1, y_2, \ldots, y_L]$, where $y_i \in \{0, 1\}$ indicates whether the corresponding MSU contains the malicious payload. Finally, we assume that a small set of location-labeled attack requests R can be obtained in advance through manual investigation for constructing the function g. Similar to the definition of y, the location labels for an attack request can also be formulated as a 0-1 vector with the length of L.

IV. Overview

We propose a novel framework, WebSpotter, which achieves interpretable Web attack detection by locating malicious payloads. We observe that malicious payloads often have a significant impact on the predictions of the detection models. Inspired by this observation, we identify the importance of each MSU within HTTP requests detected by the trained detection model via gradient analysis, and utilize a machine learning model (i.e., the localization model) to learn the relationship between the importance and the location of malicious payloads. Note that WebSpotter is orthogonal to existing DL-based Web attack detection methods [2], [21], [20] that capture attack requests, and can be applied to these methods to locate malicious payloads. Based on our localization results, WebSpotter helps security operators understand attack behaviors and automatically generate WAF rules to defend against the attacks. As shown in Figure 2, WebSpotter contains three modules below.

- **Embedding attribution** assesses the importance scores of embedding vectors in existing Web attack detection models. It utilizes a gradient-based method to evaluate the contribution of each embedding value to the model prediction results and performs linear interpolation to mitigate the interference of gradient noise.
- **HTTP-structure alignment** evaluates the importance of each MSU within HTTP requests based on embedding importance. We leverage a two-stage aggregation method to align embedding importance with MSUs. Specifically, it first consolidates importance scores across different embedding dimensions into a single score for each token and then derives importance scores of MSUs by identifying the association between these units and tokens.
- **Malicious payload localization** constructs specialized location features by integrating the textual semantics and importance scores of each MSU within HTTP requests, and trains a lightweight localization model using a very limited number of location-labeled attack requests. The location features are finally fed into the localization model to determine the locations of malicious payloads.

V. Design Details

A. Embedding Attribution

The embedding attribution module aims to analyze the behavior of a trained detection model by assessing the influence of each embedding value of the model on its predictions. The module needs to i) provide insights into how changes in

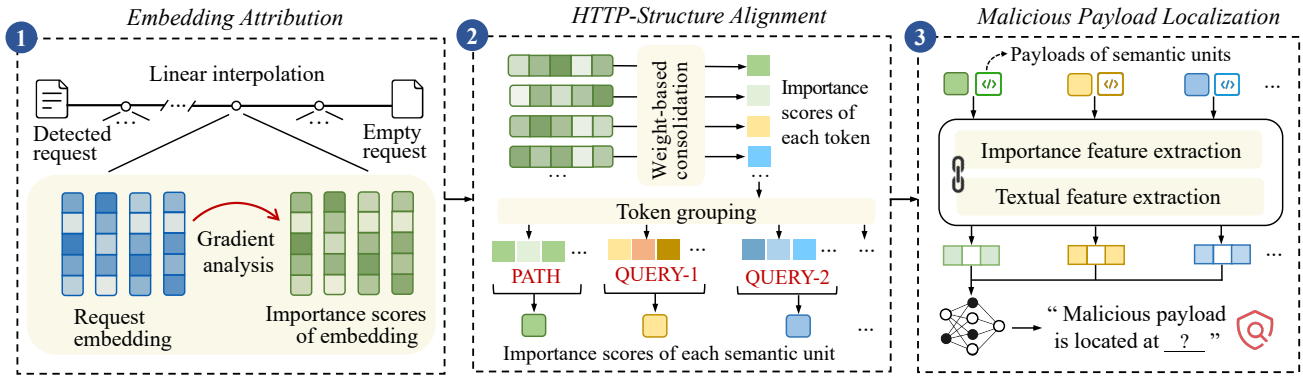


Fig. 2: The overview of WebSpotter.

embedding values affect the output, and ii) be computationally efficient, considering that the number of embedding values is typically quite large.

We perform gradient analysis to achieve this goal. Generally, a larger absolute value of the gradient indicates a greater influence of the corresponding feature on the predictions. Therefore, for a request that has been identified as an attack by detection models, we compute the gradients of the output labels with respect to each embedding value. Such a gradient-based approach has two benefits. First, it can reflect how much each feature contributes to the output more faithfully compared to methods that approximate feature importance through perturbations [22], [23] or surrogate models [6], [24], since the internal parameters and mathematical properties of the model are directly utilized. Second, the importance of each embedding value can be calculated efficiently and concurrently in a single backpropagation process, unlike existing methods that introduce substantial time overhead for additional model training [7] or inference [22].

Nevertheless, the embedding importance calculated via gradients is highly susceptible to noise because of the local nature of gradients. For example, when the model is overfitted to some embedding values, these values may exhibit a gradient of 0, even though they are salient with regard to the prediction [25]. To deal with the issue, we choose to perform linear interpolation between the embedding of the original HTTP request and an empty request (i.e., zero embeddings), and aggregate the gradients of these interpolation points, following the idea of integrated gradients [26]. Such a method avoids relying on the gradients at a single point and provides a global view of embedding importance to mitigate the problem of noise.

The detailed procedure for obtaining the importance of the embedding values is described below. We first utilize the existing Web attack detection system (see Section II-B for details) to process the raw HTTP requests, thereby yielding the token sequences, embeddings, and predictions, respectively. We represent token sequences as $t = [t_1, t_2, \ldots]$, where t_i denotes the i_{th} token of the sequence. The symbol e represents the embedding matrix of the detection model, with e_{ij} referring to the value in the j_{th} dimension of the embedding for the token t_i. The symbol $\hat{y}$ is used to denote the predicted label. Then, we perform backpropagation to obtain the gradient-based embedding importance. Specifically, the importance of e_{ij}, denoted as G_{ij}, is computed as follows:

$$G_{ij} = \frac{1}{m} \cdot \sum_{k=1}^{m} \frac{\partial f_\theta(\hat{y} \mid \alpha_k \cdot e)}{\partial e_{ij}}, \tag{1}$$

where m represents the number of interpolation points and serves as a hyperparameter that will be discussed and evaluated later, α_k equals to $\frac{k}{m}$ and represents the step intervals of interpolations, f_θ is the detection model.

B. HTTP-structure Alignment

HTTP-structure alignment module aims to align the embedding importance with MSUs of HTTP requests, and obtain the importance of these units. This module is necessary because, as mentioned earlier, we need to judge which HTTP semantic units contain the malicious payload instead of simply attributing attacks to embedding values. Specifically, given the importance scores of embeddings of an HTTP request, we process them through a two-stage aggregation to perform the alignment. First, we perform token-level aggregation to consolidate embedding values of each token into the token importance score. Second, we identify the tokens corresponding to each MSU and aggregate the importance of these tokens, namely protocol-level aggregation. During these two stages, distinct aggregation techniques are employed to accommodate the variability in data types.

Token-level Aggregation. A simple way to obtain an importance score for each token is to compute the average or sum of the importance scores of the corresponding embedding values. However, such a method is imprecise because embedding values at different dimensions have different semantics and therefore their importance is at different scales.

A mathematically sound approach would be to compute the gradient of each token according to the chain rule to evaluate its importance. Unfortunately, the look-up operation of the embedding layer is not differentiable because the inputs to the embedding layer (i.e., the index of the corresponding token) are discrete variables. To address this, we treat the

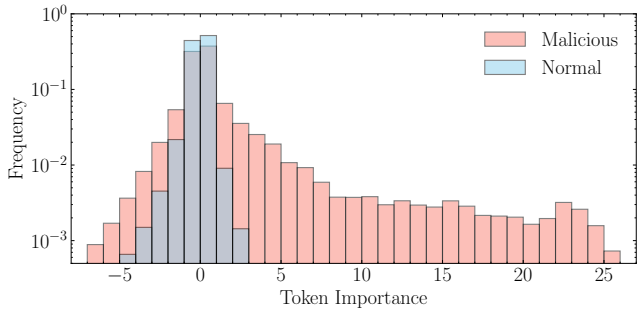


Fig. 3: Token importance distribution of MSUs with malicious and normal payloads (using the FPAD dataset).

Algorithm 1 Constructing the location features

Input: MSU list $u = [u_1, u_2, \ldots, u_m]$ of a HTTP request, m is the number of MSUs; Importance scores $U = [U_1, U_2, \ldots, U_m]$ of u; index i of the target MSU, sentence embedding model $g(\cdot)$, size d_I of importance features, size d_S of sentence embeddings.
Output: Location features x_i of the i_{th} MSU u_i.

```
Initialize a d_I-dimensional zero vector z_I;
z_I[0] ← U_i;
U ← Remove U_i from U;
U ← Sort U in descending order;
if m ≥ d_1 then
    z_I[1 : d_I] ← U[0 : d_I − 1];
else
    z_I[1 : m − 2] ← U[0 : m − 1];
end if
z_S ← g(u_i, d_S);
x_i ← Concatenate (z_I, z_S);
return x_i
```

word embedding look-up operation as a dot product between the weight of embedding layer and a one-hot encoding vector [27]. Then, we can approximate the importance of each token by calculating the gradient of the 1-entry in the one-hot vector, considering the other 0-entries do not include valid information. Specifically, according to the chain rule, the importance S_i of the token t_i is computed as follows:

$$S_i = \text{vec}(G_i \cdot \frac{\partial e_i}{\partial v_i})^i = G_i \cdot e_i \ , \tag{2}$$

where $\text{vec}(\cdot)^i$ represents the operation of indexing the i_{th} element of a vector, v_i is the one-hot encoding vector of the token t_i, $e_i = [e_{i1}, e_{i1}, \ldots, e_{id}]$ is the embedding vector of the token t_i and d is the embedding size, and $G_i = [G_{i1}, G_{i1}, \ldots, G_{id}]$ is the importance of e_i.

Protocol-level Aggregation. To align token importance scores with MSUs, we first identify which tokens are contained within a specific MSU. To achieve this, a leave-one-out approach is utilized. Specifically, for each MSU, we remove it from the entire HTTP request individually and then execute the preprocessing algorithm for the MSU-removed requests. The tokens associated with it are determined by the difference between the original token sequence and the MSU-removed token sequence. Note that for some existing preprocessing techniques, applying them to each MSU individually is also a logical approach to establish the correspondence between tokens and MSUs. However, certain preprocessing approaches may require considering the HTTP request as a whole, e.g., [20]. Thus, our leave-one-out approach, which regards preprocessing as a black box, is more broadly applicable.

Next, we determine the importance score for each MSU by summing the importance scores of its constituent tokens. Formally, the importance score U_i of the i_{th} MSU is computed as follows:

$$U_i = \sum_{j \in M_i} S_j, \tag{3}$$

where M_i denotes the index set of tokens contained within the MSU u_i. The rationale behind the summation operation is based on the observation that, for MSUs not containing malicious payloads, the importance of their tokens approximately follows a normal distribution with a mean of zero. Consequently, the expected value of their sum remains zero. In contrast, malicious MSUs typically contain multiple tokens with high importance. Therefore, the summation operation can maximize the difference in importance scores between malicious and normal payloads, which is beneficial for achieving more precise localization in subsequent steps. Figure 3 provides an example of the distribution of token importance to further illustrate this issue.

C. Malicious Payload Localization

The malicious payload localization module utilizes the MSU importance scores derived from previous steps to identify which MSUs contain malicious payloads. A straightforward strategy is to set a threshold, where MSUs with importance scores above this threshold are considered malicious. However, this strategy is susceptible to the spurious correlation problem [28]. While malicious payloads often exert a significant influence on model predictions, not all highly impactful payloads are necessarily malicious. Due to limitations in the training data or model capabilities, deep learning models may inadvertently learn spurious correlations, leading some non-malicious payloads to be also deemed highly impactful by the model. For example, in the training data, attack requests may only appear on a specific webpage, which can cause the model to infer a spurious correlation between the URL path of this webpage and the malicious intent.

To mitigate the risk of the localization being misled by detection models, we incorporate the textual semantics of MSUs and train a lightweight machine learning-based localization model to establish a robust connection between the location of malicious payloads and the MSU importance. Intuitively, the textual semantics of malicious payloads typically differ from those of normal payloads. Thus, leveraging textual semantics offers an additional perspective to complement MSU importance and enhance localization robustness. Specifically, we first construct a hybrid location feature that integrates both the importance and textual semantics for each MSU. Then, we train a localization model based on the location features and a small number of location-labeled attack requests, and apply

Algorithm 2 Training the localization model

Input: Attack requests $R = \{(u^1, c^1), (u^2, c^2), \cdots, (u^n, c^n)\}$, n is the number of requests, $u^i = [u^i_1, u^i_2, .., u^i_{m^i}]$ denotes the MSU list of the i_{th} request, m^i is the number of MSUs of the i_{th} request, $c^i = [c^i_1, c^i_2, .., c^i_{m^i}]$ is a 0-1 vector that indicates the location of malicious payloads; training algorthim $\mathcal{T}(\cdot)$.
Output: Localization model g_θ.

```
1: D ← φ;
2: for i = 1, 2, ..., n do
3:     for j = 1, 2, ..., m^i do
4:         x ← Building location features for u^i_j based on Alg. 1;
5:         D ← D ∪ {(x, c^i_j)};
6:     end for
7: end for
8: D_r ← BalanceSample(D)
9: g_θ ← T(D_r)
10: return g_θ
```

the trained localization model to each MSU within an HTTP request to determine whether it contains malicious payloads.

Constructing the Location Features. We construct the location features from two aspects, i.e., importance features and textual features. First, the importance features of an MSU are determined by its own importance score as well as the importance scores of its sibling MSUs, where sibling MSUs refer to all other MSUs in the same HTTP request. Note that considering the importance scores of sibling MSUs is crucial, as this enables the localization model to assess the relative importance of each MSU within its original HTTP request. Second, the textual features are extracted using a sentence embedding model. Here, we adopt the Nomic-embed technique [29], which trains a BERT-like encoder model through contrastive learning and utilizes Matryoshka representation learning [30] to obtain resizable embeddings.

Algorithm 1 demonstrates the detailed process of constructing location features. Let d_I and d_S denote the numbers of dimensions of importance features and textual features, respectively. For the importance feature, the first dimension is set to the importance score of the target MSU (line 2). The remaining d_I-1 dimensions are the importance scores of its siblings MSUs in descending order (line 4). If the number of sibling MSUs exceeds d_I-1, we only select the top d_I-1 scores. Otherwise, we pad zeros to the end of the importance features (lines 5-8). For the textual feature, it is computed using the Nomic-embed model (line 10). We then concatenate two features together (line 11).

Training the Localization Model. The localization model is essentially a lightweight binary classification machine learning model. It takes the location features of an MSU as input and outputs either 0 or 1, where 0 indicates that the MSU does not contain malicious payloads, and 1 means otherwise. We train the localization model using a small number of location-labeled attack requests. Location features of MSUs are extracted from these attack requests and paired with their corresponding labels to form the training dataset. We adjust the number of normal and malicious samples in the training dataset to avoid class imbalance. This is necessary because malicious payloads generally make up only a small fraction of the HTTP requests, resulting in a disproportionate amount of normal data. By addressing the class imbalance, we prevent the model from favoring the majority class and incorrectly identifying inputs as normal. With the dataset, we train a random forest model, which is an efficient non-deep learning model and often used in security tasks [31].

The detailed steps are shown in Algorithm 2. For each attack request, we construct location features for all MSUs (lines 2-7). Balancing is achieved by randomly removing 0-class samples until the number of 0-class samples matches the number of 1-class samples (line 8). Finally, we conduct the training algorithm to obtain localization models (line 9).

VI. Generating WAF Rules

In this section, we demonstrate how security operators can leverage the localization results produced by WebSpotter to automatically generate WAF rules. Specifically, we consider a common scenario where security operators deploy a traditional rule-based WAF for real-time attack blocking, and meanwhile use a machine learning model to offline detect potential attacks that the WAF may fail to cover. To efficiently and promptly mitigate these WAF-uncovered threats, new WAF rules should be generated. We clarify that our objective in automatically generating rules is not to deploy them directly in WAFs, but rather to facilitate security operations. Specifically, these generated rules serve as initial references for the security operators to derive the final rules.

A. Rule Format

We select SecLang [32] as the configuration language for WAF rules to ensure that the generated rules are deployable and broadly applicable across different WAF engines. SecLang was originally implemented by ModSecurity [33] which is a widely adopted open-source WAF, and has been supported by the majority of other open-source and commercial WAF solutions (e.g., Coraza [34], HAProxy WAF [35], IBM SVA WAF [36], and VMware Avi WAF [37]).

Each SecLang rule consists of three fields: `Variable`, `Operator`, and `Action`. The `Variable` is an enumerated type value that specifies the scope of rule matching. For example, the value ARGS_GET means the corresponding rule will inspect the payloads of all query parameters. The `Operator` functions as a conditional expression, defining the conditions under which the rule will be triggered. Typically, a regular expression is defined in this field to detect patterns associated with specific malicious payloads. The `Action` determines the behavior executed upon successful rule matching. For instance, a common strategy involves rejecting the request and returning an HTTP 403 status code. Readers can refer to the official document [32] for detailed syntax of SecLang rules. In this work, we primarily focus on how to generate the appropriate `Variable` and `Operator` fields. The `Action` field is not considered because it is not related to attack detection and is usually configured according to the specific needs of Web applications in practice.

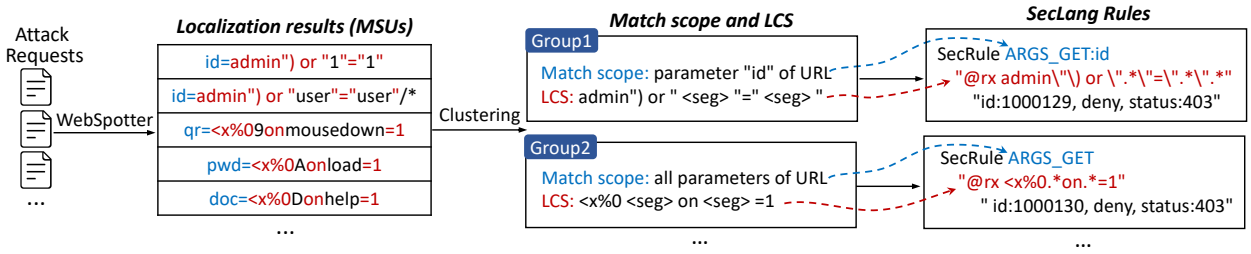


Fig. 4: An illustrative example showing how to generate SecLang rules based on the localization results produced by WebSpotter.

B. Rule Generation

Inspired by existing work [38], [39], [6] on generating detection rules for network intrusion detection systems, we propose an automated method to derive effective WAF rules from the localization results generated by WebSpotter. Our design consists of two steps. First, we employ payload clustering to categorize MSUs that exhibit similar semantics based on textual similarity. Then, for each cluster, we generate a SecLang rule by determining the match scope and extracting the regular expressions that reflect their common pattern. This clustering-then-generation scheme facilitates the extraction of common malicious patterns, ensuring that the generated rules can match a certain type of attacks sharing similar characteristics rather than being constrained to specific individual requests. An illustrative example of this process is shown in Figure 4.

We leverage the idea of hierarchical clustering algorithm [40] to categorize MSUs with similar semantics. The clustering algorithm operates iteratively. Initially, each MSU payload is treated as a separate group, with the payload itself serving as the group representative. In each iteration, we obtain the distances between all pairs of groups based on the distance between their representatives. The two groups with the shortest distances are merged, and the representative of the new group is the payload with the smallest average distance to other elements in the group. Here, we choose the edit distance as the distance metric, given its ability to quantify the similarity between two strings. To ensure high intra-group similarity, we impose conditions that prevent merging highly dissimilar groups. Specifically, two groups are deemed non-mergeable if their edit distance surpasses a proportion threshold (set to 0.25 in our evaluations) of their combined length. The clustering process terminates when no groups can be merged.

After obtaining groups with similar MSUs, we propose to automatically generate one WAF rule tailored to each group. As mentioned earlier, we need to determine the `Variable` and `Operator` fields of the SecLang rules. For the `Variable` field, we determine its value based on the location of MSUs in the group. We use the REQUEST_FILENAME, ARGS_GET, REQUEST_HEADERS, and ARGS_POST to inspect request paths, query parameters, request headers, and body parameters, respectively. For the `Operator` field, we generate regular expressions that define the common pattern within the group. Initially, we extract the longest common subsequence (LCS) among all payloads in the group, representing shared characters. Subsequently, the LCS is divided into segments when non-matching characters exist within the LCS across different payloads in the same group. These segments are concatenated using the Kleene star operator * to form a regular expression. The operator allows for zero or more repetitions of characters between segments, ensuring that payloads with the segments in relative order are matched, despite variation in the presence of obfuscation.

VII. Evaluation

A. Settings

Datasets. We use two widely used public datasets [21], [14], [3] and a newly constructed dataset for evaluations. Each item of these datasets contains the original HTTP request, attack category label, and the location label for malicious payloads. The statistics of these datasets are shown in Appendix A.

- The CSIC dataset [41] is collected from an e-commerce Web application and includes two categories: benign and attack HTTP requests. Note that the original CSIC dataset does not contain the location labels of attack requests. To address this, we ask two PhD students in the field to perform manual annotation.
- The PKDD dataset [42] is the only publicly available dataset known to include location labels of malicious payloads. It is generated by recording real traffic and comprises one benign category and seven attack categories (e.g., XSS). In this dataset, some non-malicious fields of requests are replaced by random strings. This reflects the characteristics of certain real-world Web applications that generate requests including a number of fields without semantic meaning, e.g., some parameters consist of randomly generated IDs.
- We also construct the new FPAD (**F**usion **P**ayload Web **A**ttack **D**etection) dataset. The FPAD dataset offers the following key advancements over existing datasets: (i) it incorporates advanced and contemporary attack techniques, such as the 2021 Log4j2 vulnerability exploit [43]. (ii) It provides precise location labels of malicious payloads. (iii) Beyond the standard training and testing sets, it includes an additional OOD (out-of-distribution) attack testing set, enabling evaluation of the model performance in handling unknown attacks. For example, the OOD testing set includes SQL injection attacks targeting specific databases (e.g., Oracle), whereas the training set only contains common SQL blind injection attacks. FPAD includes one benign and four

attack categories. We construct the dataset using the method similar to the one used for creating the CSIC dataset [41]. For benign requests, we directly use the benign data from CSIC. To generate attack requests, we collect attack techniques from multiple sources (e.g., PayloadsAllTheThings [44] and xray [45]) and then leverage Burp Bounty [46], a Web security tool, to construct valid attack requests. Further construction details of FPAD are provided in Appendix B.

Baselines. We select four state-of-the-art methods specifically designed for localization tasks as baselines.

- *LTD* [47] employs the Xception model [48] to generate the feature maps of HTTP requests. Then, it applies a sliding window mechanism to generate a set of candidate regions on the feature maps, and utilizes a specifically designed neural network to predict the position of malicious payloads.
- *SSD* [49] applies multiple down-sampling blocks to extract multi-scale feature maps from HTTP requests, and then locates malicious payloads in a manner similar to LTD.
- *REG* [50] constructs a regression task to predict the positions of malicious payloads. It employs a dual-head neural network to simultaneously handle both the regression task and the original attack detection task.
- *CLS* [51] treats each MSU as an independent entity and assesses its potential maliciousness. It analyzes the textual semantics of each MSU and trains a machine learning model to determine whether the MSU contains malicious payloads.

LTD, SSD, and REG output a set of string indices representing the start and end positions of malicious payloads within the HTTP request. We map the predicted positions to MSUs by calculating the intersection over union (IoU), which is a common strategy used in localization tasks. Additional implementation details of baselines are presented in Appendix D. It is also noteworthy that we exclude existing interpretable machine learning methods for network security (e.g., Lemna [7]) because, as mentioned previously, they fail to produce effective interpretability for Web attack detection. Nevertheless, these methods or their variants could potentially be used within our framework as replacements for the embedding attribution module to derive importance scores. In Section VII-C, we conduct ablation experiments to evaluate these alternatives.

Metrics. We evaluate the localization performance using the widely used multi-label classification metrics: precision (Pre), recall (Rec), F1-score (F1), and Jaccard Index (Jac) [52]. We first calculate these metrics for each individual attack request. As described in Section III, for each attack request, we output a binary vector $y = [y_1, y_2, \ldots, y_L]$ to indicate the location of malicious payloads. Suppose that the corresponding ground truth location is represented as a binary vector $z = [z_1, z_2, \ldots, z_L]$, these metrics are computed by comparing y and z. Then, we compute the average performance across the dataset to obtain an overall evaluation.

Configuration. We select TextCNN [14] as the basic DL-based Web attack detection model because it demonstrates the best overall detection performance in comparative testing with various models. We also show the generality and effectiveness of our approach on other models in Appendix C. Regarding the hyperparameters of WebSpotter, we set m to 50, d_I to 10, d_S to 256. We evaluate the hyperparameter sensitivity in Appendix E. Moreover, our method and baselines require some location-labeled attack requests for training. Considering that manual labeling incurs significant costs, we limit the labeling overhead to 1%, i.e. randomly using only 1% of attack requests in the training set. We believe that this level of overhead (e.g., labeling only 117 requests for the CSIC dataset) is acceptable in practice. The impact of labeling overheads is evaluated in Section VII-B.

```
POST /tienda1/miembros/editar.jsp HTTP/1.1
...

modo=entrar&login=hodgson&password=d68as4o&nombre=Evangelino
&apellidos=De Len Pizano&email=leuwerik@white-pearl-resort.tt&dni=3
0293636Z&direccion=Calle Perez Galdos, 167&ciudad=Tapia de Casarie
go&cp=49660&provincia=A Corua&ntc=4997441125317554&B1=..../%5
c..../%5cetc%5cissue
```

Fig. 5: An example showing the limitation of LTD, where the malicious payloads located by LTD are highlighted in red and the `ntc` parameter is incorrectly identified.

Implementation. Our evaluations are conducted on a GPU server equipped with NVIDIA GeForce RTX 4090 GPUs and 384GB of memory. We implement the deep learning models with PyTorch 2.1.2 and CUDA 12.1 toolkit and use the Sentence Transformers library to obtain the Nomic embedding. We run each experiment three times with varying random seeds and report average results.

B. Localization Performance

First, we evaluate whether our method outperforms baselines with limited location-labeled training data. We set the labeling overhead to 1% and the results are shown in Table I. Compared to all baselines across different datasets, WebSpotter achieves at least a 16.3% and 22.2% relative improvement in F1-score and Jaccard index, respectively. For example, on the FPAD dataset, WebSpotter improves the F1-score from 0.828 to 0.988, which corresponds to 721 additional malicious payloads being correctly identified and 4333 fewer false positives. In most cases, the F1-score of WebSpotter approaches 1.0, demonstrating the effectiveness of our method in accurately identifying the location of the malicious payloads within the attack requests. Moreover, despite a slight performance degradation, our method still achieves the F1-score of 0.970 on the FPAD-OOD dataset, which illustrates the generalization of WebSpotter in handling unseen attack patterns. Among the baselines, CLS achieves the best performance in most cases, suggesting that employing a binary classification strategy for each MSU offers significant advantages over methods that directly generate the string indices of malicious payloads, especially in scenarios with limited training data. One exception is that CLS performs poorly on the PKDD dataset. The main reason is that the PKDD dataset replaces some non-malicious fields with random strings, and thus the model is unable to extract their textual semantics effectively.

TABLE I: Overall performance comparisons of malicious payload localization when the labeling overhead is set to 1%.

Method†	CSIC				PKDD				FPAD				FPAD-OOD			
	Pre	Rec	F1	Jac	Pre	Rec	F1	Jac	Pre	Rec	F1	Jac	Pre	Rec	F1	Jac
LTD	0.269	0.638	0.361	0.269	0.388	0.883	0.507	0.391	0.467	0.870	0.581	0.462	0.497	0.878	0.606	0.491
SSD	0.179	0.999	0.292	0.187	0.125	0.807	0.211	0.136	0.265	0.929	0.381	0.264	0.282	0.932	0.398	0.284
CLS	0.392	0.831	0.499	0.392	0.098	0.966	0.171	0.096	0.781	0.929	0.828	0.780	0.788	0.932	0.834	0.785
Ours	0.968	0.980	0.972	0.968	0.990	0.986	0.986	0.983	0.987	0.993	0.988	0.983	0.966	0.982	0.970	0.959

† REG is not shown due to its excessively poor performance in this setting.

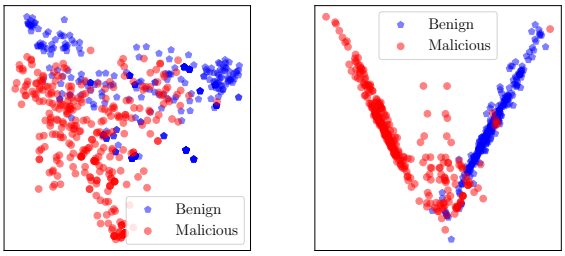
TABLE II: Overall performance comparisons of malicious payload localization when the labeling overhead is set to 100%.

Method	CSIC				PKDD				FPAD				FPAD-OOD			
	Pre	Rec	F1	Jac	Pre	Rec	F1	Jac	Pre	Rec	F1	Jac	Pre	Rec	F1	Jac
LTD	0.420	0.776	0.520	0.420	0.771	0.947	0.826	0.761	0.787	0.944	0.837	0.781	0.803	0.940	0.845	0.795
SSD	0.266	0.998	0.401	0.271	0.369	0.964	0.501	0.377	0.309	0.981	0.448	0.309	0.325	0.974	0.459	0.327
REG	0.102	0.235	0.129	0.102	0.148	0.334	0.185	0.143	0.376	0.587	0.430	0.369	0.333	0.602	0.392	0.321
CLS	0.867	0.969	0.900	0.867	0.462	0.972	0.589	0.460	0.989	0.995	0.991	0.989	0.984	0.991	0.986	0.954
Ours	0.980	0.987	0.982	0.980	0.991	0.992	0.991	0.990	0.997	0.997	0.997	0.997	0.993	0.987	0.988	0.984

Second, we evaluate WebSpotter when the labeling overhead is 100%, i.e., all attack requests in the training set are used to train the localization model. As shown in Table II, our method also surpasses all baselines, which indicates that our method is still advantageous even without considering the constraints of labeling overhead. The performance of LTD improves significantly when the labeling overhead is increased from 1% to 100%. However, its overall performance remains inferior to that of CLS, with a particularly large gap in the precision metric. The main reason we identified is that LTD only considers fixed character intervals when generating candidate regions and fails to incorporate the HTTP structural information. Consequently, it often recognizes some normal payloads surrounding malicious payloads as malicious. Figure 5 shows a representative example where LTD incorrectly identifies the normal `ntc` parameter adjacent to the malicious `B1` parameter as malicious.

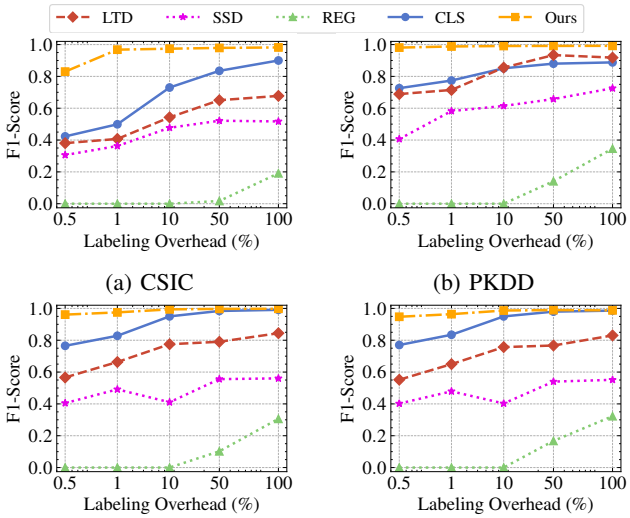
Moreover, CLS does not perform as well as our method because it only considers textual semantic features of MSUs, while we additionally consider the importance features that provide a more effective criterion to identify MSUs with malicious payloads. We validate this claim by visualizing the feature distribution. We randomly select 1% of the attack requests from the FPAD dataset and project their textual features and importance features into 2-dimension using the principal component analysis (PCA). Figure 6 shows the results. It can be observed that textual features significantly overlap between benign and malicious MSUs, while the importance features extracted by WebSpotter have a more distinct category boundary, illustrating the effectiveness of our importance features.

Figure 7 further shows the trend of F1-score for each method as the labeling overhead changes. It can be seen that WebSpotter is stable and consistently maintains a significant performance advantage across all datasets. With only 1% labeling overhead, WebSpotter can achieve results comparable to, or even better than, existing methods that impose 100% labeling overhead, demonstrating that WebSpotter can significantly reduce the labeling costs. In contrast, baselines perform poorly under low labeling overhead, although their F1-scores improve as the labeling overhead increases.



(a) Textual features (b) Importance features

Fig. 6: PCA Visualization of different types of features.



(a) CSIC (b) PKDD

(c) FPAD (d) FPAD-OOD

Fig. 7: Localization performance w.r.t. the labeling overhead.

C. Ablation Study

We construct several variants of WebSpotter to validate the effectiveness of each individual module. First, we replace our embedding attribution module with existing interpretable

TABLE III: Performance comparison of various variants of WebSpotter. S.S., T.F., and L.M. are abbreviations for single-stage summation, textual features, and localization models, respectively.

Method	CSIC			PKDD			FPAD			FPAD-OOD		
	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
WebSpotter w/ CADE	0.884	0.962	0.907	0.983	0.985	0.982	0.959	0.991	0.969	0.947	0.984	0.958
WebSpotter w/ SHAP	0.439	0.817	0.537	0.958	0.996	0.964	0.927	0.976	0.942	0.907	0.961	0.923
WebSpotter w/ Lemna	0.817	0.939	0.855	0.900	0.977	0.921	0.950	0.990	0.963	0.938	0.978	0.949
WebSpotter w/ VG	0.876	0.960	0.903	0.912	0.981	0.930	0.961	0.994	0.971	0.958	0.985	0.963
WebSpotter w/ S.S.	0.755	0.923	0.804	0.898	0.987	0.915	0.963	0.990	0.972	0.894	0.943	0.953
WebSpotter w/ FINER	0.867	0.952	0.890	0.917	0.983	0.933	0.975	0.995	0.981	0.961	0.983	0.966
WebSpotter w/o T.F.	0.965	0.978	0.970	0.977	0.990	0.975	0.940	0.976	0.944	0.923	0.952	0.917
WebSpotter w/o L.M.	0.948	0.982	0.959	0.969	0.983	0.971	0.989	0.935	0.947	0.976	0.904	0.921
Original WebSpotter	0.968	0.980	0.972	0.990	0.986	0.986	0.987	0.993	0.988	0.966	0.982	0.970

† The results of the Jaccard Index exhibit a trend similar to that of the F1 score, and those results are presented in Appendix F.

machine learning methods, including CADE [22], LEMNA [7], SHAP [53], and vanilla gradients (VG) [54]. We implement these methods following the settings described in prior works, and the details are provided in Appendix D. Table III presents the comparison results, demonstrating that the original WebSpotter consistently achieves the best performance across all datasets. The improvements are particularly significant on the CSIC dataset. For instance, the F1-score of the original WebSpotter reaches 0.972, whereas the F1-score of *WebSpotter with SHAP* drops significantly to 0.537. This occurs because the CSIC dataset is relatively simple, and detection models tend to be overfitting on it. As a result, the outputs produced by these interpretability methods are biased. In contrast, our method effectively mitigates this issue by performing linear interpolation. A similar overfitting issue is also observed on the PKDD dataset. However, the localization performance on the PKDD dataset remains relatively high for baselines, since in the PKDD dataset, some non-malicious payloads are replaced with random strings, which introduces a shortcut for identifying malicious payloads.

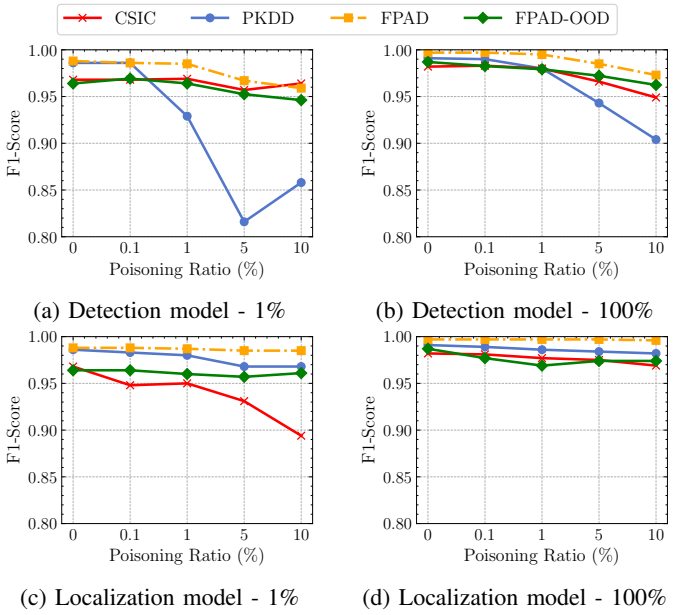
Second, we consider alternative designs for the HTTP-structure alignment module. This module deploys a two-stage aggregation scheme, applying different aggregation techniques at each stage. To assess the effectiveness of this design, we first consider a simple single-stage aggregation strategy that sums the embedding importance scores directly to obtain the MSU importance. In addition, we also consider FINER [9] which proposes to aggregate low-level interpretability results by calculating their L1 norm. These two variants are denoted as *WebSpotter with Single-stage Summation* and *WebSpotter with Finer*, respectively. As shown in Table III, both variants perform worse than the original WebSpotter. Specifically, our method improves the F1-score by at most 21% compared to the single-stage summation approach and 9% compared to FINER. These results demonstrate that our two-stage aggregation scheme is better suited for computing the MSU importance.

Third, we evaluate the malicious payload localization module. We consider two variants for this module: i) *WebSpotter without textual features*, which only builds the importance features of MSUs for the localization model. ii) *WebSpotter without localization models*, which directly uses the MSU importance scores to perform localization, bypassing the localization model. Specifically, we calculate importance scores for all MSUs within an HTTP request and determine a threshold by summing the average and standard deviation of these scores. MSUs with importance scores above this threshold are considered as malicious. The original WebSpotter effectively improves the localization accuracy compared with only considering the importance features, showing that the textual features provide valid information to complement the MSU importance. Compared with *WebSpotter without localization models*, our scheme also provides valid improvements in all datasets. This is because not all MSUs with high importance scores are necessarily malicious, and only relying on a threshold-based strategy is inadequate for precise localization. We present a representative example in Appendix F to further illustrate this issue.

D. Robustness under Data Poisoning

We evaluate the robustness of WebSpotter against data poisoning attacks by considering poisoning both the detection model and the localization model. Specifically, we simulate poisoning by manipulating samples in the training dataset of both models. For the detection model poisoning, we replace the class labels of requests with incorrect labels. For the localization model poisoning, we modify the location ground truth of requests, reassigning malicious labels to random MSUs in the request. We vary the poisoning ratios over [0.1%, 1%, 5%, 10%] and conduct experiments for both models when the labeling overhead is set to 1% and 100%. Note that, both kinds of poisoning attacks are hard to launch in practice, especially the localization model poisoning, as the ground truth is mainly provided by security operators.

Figure 8 presents the localization performance under detection model poisoning and localization model poisoning. Under detection model poisoning in Figures 8a and 8b, it can be seen that, except for PKDD, the localization performance only degrades slightly when the ratio increases. On the PKDD dataset, the localization performance drops significantly with a larger poisoning ratio, especially when the labeling overhead is 1%. This is because, in the PKDD dataset, benign payloads are anonymized into random strings. The detection model is



(a) Detection model - 1% (b) Detection model - 100%

(c) Localization model - 1% (d) Localization model - 100%

Fig. 8: Localization performance under detection and localization model poisoning with 1% and 100% labeling overhead.

more likely to incorrectly capture the relationship between features and maliciousness when learning from poisoning samples created from these payloads. Under localization model poisoning in Figures 8c and 8d, the performance generally remains stable compared to detection model poisoning, with the exception of the 1% data poisoning on the CSIC dataset. Moreover, when the labeling overhead is 100%, the impact of poisoning is less significant; e.g., the performance only drops by 1.3% on the CSIC dataset even if the poisoning ratio is 10%. We argue that more training data enables the localization model to learn robust features instead of spurious ones from clean samples, thereby reducing the interference from poisoned samples. Overall, WebSpotter demonstrates robustness against poisoning. Additionally, a larger training dataset can enhance the robustness of the WebSpotter.

E. Evaluations on Real-world Attacks

We also evaluate WebSpotter on real-world Web attacks collected from recently reported CVEs and two real-world Web applications sourced from a cloud provider to investigate the effectiveness of WebSpotter in practical scenarios.

Collection from CVEs. We collect over 750 exploits from CVEs reported between 2021 and 2024, and perform evaluation on these attacks. We manually annotate the collected attacks to determine the ground-truth locations of malicious payloads and construct the CVE dataset. Specifically, the collected CVE attacks are divided into two groups: the first group comprises CVEs reported in 2021, while the second group consists of remaining CVEs. The data of the first group is combined with all training data of FPAD to form the CVE training set. We train the attack detection model and the localization model based on this CVE training set. Finally, we evaluate the localization performance on the second group, i.e., those CVEs reported between 2022 and 2024.

TABLE IV: Localization performance on CVE attacks.

Labeling Overhead	Method	Pre	Rec	F1	Jac
100%	LTD	0.365	0.849	0.470	0.356
	SSD	0.241	0.891	0.355	0.239
	REG	0.241	0.320	0.254	0.224
	CLS	0.433	0.977	0.550	0.429
	Ours	0.895	0.920	0.891	0.861
1%	LTD	0.357	0.821	0.456	0.350
	SSD	0.212	0.886	0.315	0.209
	CLS	0.430	0.967	0.544	0.422
	Ours	0.861	0.926	0.868	0.827

TABLE V: Classification performance and localization performance of WebSpotter on two real-world applications.

Application	Classification			Localization			
	Pre	Rec	F1	Pre	Rec	F1	Jac
G1	0.930	0.845	0.872	0.923	0.984	0.945	0.930
G2	0.981	0.960	0.970	0.913	0.976	0.928	0.912

Table IV shows the results. Our method achieves F1-scores of 0.87 and 0.89 when the labeling overhead is 1% and 100%, both significantly outperforming the baselines. These results demonstrate the effectiveness of our method in handling real-world attacks. Interestingly, we observe that the performance of CLS does not exhibit a significant improvement when the labeling overhead increases from 1% to 100%. The performance difference between CLS and WebSpotter is also larger compared to those results on public datasets or the FPAD dataset.The main reason is that the distribution of testing data and training data presents a more significant difference, whereas CLS is not robust in this case. Specifically, different CVEs may target different Web components and their requests exhibit high variability, leading to the presence of numerous unseen but normal payload patterns in the testing set. However, CLS only relies on the textual features of MSUs to locate malicious payloads, which makes it ineffective in handling these unseen patterns of normal payloads.

Collection from Real-world Web applications. We collaborate with one of the three major ISPs in China, and collect HTTP requests of two Web applications (denoted as G1 and G2) for more than four days based on its cloud services. G1 and G2 are applications for e-government services hosted by the service provider. Security experts of the cloud service provider manually label all attack requests within the data and identify the positions of malicious payloads. The statistics of the data collected from these two applications are presented in Appendix A. For each application, the collected attack requests are divided into two subsets. The first subset, comprising attack requests collected on the first day, is combined with the FPAD training set to train the detection model. The remaining attack requests are then used as testing requests to evaluate the localization performance of WebSpotter.

We report both the classification and localization perfor-

TABLE VI: Detection performance of WAFs before and after generating new WAF rules.

Dataset	Pre		Rec		F1	
	Before	After	Before	After	Before	After
FPAD	0.999	0.996	0.742	0.934	0.852	0.964
CVE	0.999	0.992	0.840	0.947	0.913	0.969
G1	0.998	0.999	0.358	0.983	0.527	0.991
G2	0.999	0.999	0.744	0.996	0.853	0.998

mance on these testing requests in Table V. It can be observed that WebSpotter still demonstrates high localization performance when dealing with attacks on real-world web applications, achieving F1 scores of 0.945 and 0.928 on the two applications, respectively. Interestingly, although the detection model demonstrates relatively weak classification performance on G1, WebSpotter still demonstrates effective localization capabilities on the same dataset. This is primarily because the testing requests contain previously unseen attack patterns, causing the model to misclassify the attack types. Nevertheless, WebSpotter is still capable of accurately locating the malicious payloads under such conditions, as the MSU importance scores of malicious payloads remain markedly distinct from those of benign payloads.

F. Comparison with LLMs

Recently, some research shows the potential of large language models (LLMs) [55] in various security tasks [56], [57], [58], including Web security [59], [60]. We evaluate whether LLMs can achieve goals similar to WebSpotter, i.e., locating malicious payloads in HTTP requests. To this end, we craft a prompt to guide the LLMs and apply in-context learning [61] to incorporate the location-labeled training data. The details of our prompt is available in Appendix H. We use 10 in-context examples to perform evaluations considering the constraints of LLM context length and computational overhead. Note that, we also explore the use of more in-context examples, such as 1% of the attack requests from the training set, but observe negligible performance improvements, which is also consistent with the observation of existing studies [62].

The evaluation results are shown in Figure 9. Our method achieves the best overall performance compared to LLMs across all datasets. On the FPAD and FPAD-OOD datasets, GPT4o-mini performs the best among the LLMs and its F1-score is close to WebSpotter. On the CSIC and PKDD datasets, LLMs perform significantly worse. We observe that, compared to WebSpotter, LLMs demonstrate inferior localization performance on non-injection attacks (e.g., forceful browsing [63]). We attribute this to two reasons. First, non-injection attacks are less prevalent than injection attacks (e.g., SQL injection and XSS) and consequently appear less frequently in the training corpora of LLMs. This limits the capability of LLMs to identify such attacks. Second, LLMs lack the knowledge of benign request patterns which are helpful for locating the malicious payloads. Our experimental results demonstrate that even when equipping LLMs with in-context learning to learn these patterns, their performance is still inadequate. Effectively enabling LLMs to learn these patterns remains an open challenge. Moreover, LLMs are notably slower during inference. For example, on the FPAD dataset, our method processes each request in 0.1011 seconds, while Qwen2-72B, and LLaMA3-70B require 12.3907 and 15.8604 seconds, respectively.

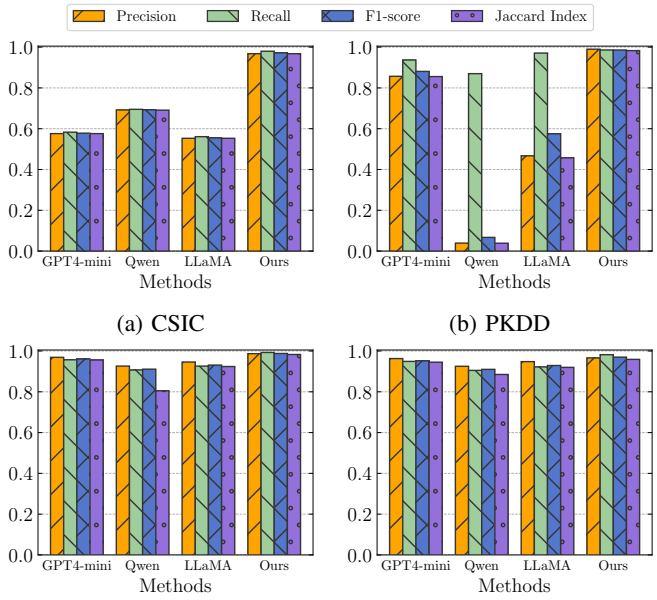
G. Effectiveness of the Rule Generation

In this section, we evaluate the effectiveness of the generated WAF rules on the FPAD, CVE, G1 and G2 datasets. We select the Coraza WAF [34], a widely used open-source WAF engine, for the experiments. The WAF is configured with the OWASP Core Rule Set (CRS) which is a generic WAF rule set and has been adopted by many commercial cloud services [64], [65], [66] as their fundamental configurations.

We conduct the experiments according to the following procedure. First, we employ the machine learning model and the WAF separately to detect attacks. Then we identify the attack requests that are detected by the machine learning methods but missed by the WAF. Subsequently, new WAF rules are generated based on these attack requests. Finally, we add these newly generated rules to the rule set and report the detection performance of the WAF before and after adding the new rules. Note that the labeling overhead is set to 1% for rule generation, and the normal requests of FPAD are added into the CVE, G1 and G2 dataset for detection performance evaluation because these datasets only contain attack requests. As shown in Table VI, the results show that the generated rules can significantly improve the attack detection rate of the WAF. For example, on the FPAD dataset, the generated rules improve the recall by 26% (from 0.741 to 0.934). It can also be observed that the generated rules may incur a small number of false positives (FPs) for WAFs. For example, the precision on the FPAD dataset drops from 0.999 to 0.996. These FPs arise from limitations in the localization method. Specifically, the localization method occasionally misclassifies some benign payloads as malicious. Consequently, our rule generation method is unable to extract effective malicious patterns from these incorrectly labeled benign payloads, resulting in rules that produce false positives. As mentioned in Section VI, the generated rules are not directly deployed into WAFs. Instead, they serve as the references for the security operators who will address this small number of FPs when deriving the final rules.

VIII. Discussion

Robustness against adversarial inputs. Some research [67], [68], [69], [70], [71] proposes to craft adversarial inputs to evade Web attack detectors. We evaluate two state-of-the-art methods, i.e., WAF-A-MoLE [70] and AdvSQLi [71], that generate adversarial SQL injection requests to evade DL-based detection. We select 194 and 500 SQL injection requests from the CSIC and FPAD datasets, respectively, and leverage the two methods to generate the adversarial inputs. The results show that none of these adversarial inputs can evade our detector, and WebSpotter remains effective in identifying malicious payloads. The reason is that both methods attack detectors



(a) CSIC (b) PKDD

(c) FPAD (d) FPAD-OOD

Fig. 9: Localization performance of LLMs.

that analyze each HTTP field independently. In contrast, WebSpotter adopts the end-to-end detection architecture, which can learn and analyze richer semantics of HTTP requests (e.g., the interactions and dependencies between HTTP fields) and thus is more robust against adversarial inputs.

Applicability of WebSpotter. WebSpotter requires white-box access to the DL-based detector to perform gradient-based embedding attribution. This requirement is justified, as localization is designed to complement detection, and typically both functions are managed by the same operational team. Nevertheless, WebSpotter can be adapted for black-box scenarios by substituting the gradient-based attribution with perturbation-based or surrogate model-based methods that rely solely on the detector's output. We have evaluated such adaptations in our ablation study (Section VII-C), examining two specific variants: "WebSpotter w/ CADE", a perturbation-based method, and "WebSpotter w/ LEMNA", a surrogate model-based method. The experimental results show that, although both alternative approaches slightly reduce localization accuracy compared to the original WebSpotter, they consistently outperform baseline methods.

Moreover, WebSpotter is compatible with DL-based detectors that process the entire HTTP request as a unified input and utilize preprocessing and embedding modules to derive numerical features. This architecture is common in existing literature [20], [3], facilitating computing the MSU importance score through the two-stage aggregation method. Conversely, a minority of detectors (e.g., [72]) analyze HTTP fields independently rather than modeling them collectively, which places them outside the scope of our study. Such methods compromise the semantic coherence of HTTP requests and have been shown to be susceptible to adversarial attacks [70], [71].

Labeling overhead. We assume that a small set of location-labeled attack requests (e.g., 1% of the training dataset) are available to train the localization model. This is because labeling of security-related data imposes significant costs [31]. For instance, the G1 dataset, collected from real-world environments across five days, requires two security experts to invest over 16 working hours to analyze attacks and generate the corresponding location labels. WebSpotter can significantly reduce this labeling effort by a factor of 100, while maintaining performance comparable to or even surpassing existing methods (see Figure 7). Note that this example pertains to a single Web application, whereas in practice, an organization typically manages multiple applications. Furthermore, considering the dynamic nature of Web applications and continuously evolving attack behaviors, regular labeling of newly collected samples is critical to ensure model accuracy and relevancy. Both of these factors significantly amplify the labeling cost in real-world environments.

Moreover, several studies published in top-tier security venues also relied on labeled samples with a similar size to ours. For example, existing works [73], [74], [75] utilized a minimum of 20, 50, and 250 labeled samples, respectively.

IX. Related Work

Web attack detection. Many machine learning-based methods have been proposed for detecting Web attacks. These methods extract features from HTTP requests in an end-to-end manner [4], [20], [76], [77], [78], [3], and then perform predictions using either supervised or unsupervised models. Unsupervised models [79], [4], [20], [80], [81] learn the patterns of benign HTTP requests and identify requests that deviate from these patterns, while supervised methods [14], [82], [83], [77], [76] separately learn the patterns of benign requests and various types of attack requests.

Classification with localization. Classification with localization is a machine learning task. It has been widely studied in the field of computer vision [84], [85], [86], [87], [49], to not only classify objects within an image but also accurately determine their locations. The key idea of existing methods is to add an additional regression model to the original classification model, which outputs a set of values to indicate the object location. To the best of our knowledge, LTD [21] is the only related method for Web attack detection with localization. LTD adopts a method similar to Faster R-CNN [86] to identify suspicious regions in HTTP requests, and then classifies the payloads of these regions to detect attacks. All these methods require a large number of location annotations, which are challenging to obtain due to substantial time and expertise required.

Interpretable deep learning. Recently, many works have been proposed to improve the interpretability of deep learning models [9], [88]. A common goal of these studies is to identify the contribution of each input feature to the prediction of a model. These methods can be grouped into the following categories. Back-propagation-based methods [54], [89], [10] propagate the prediction scores of the models back to the input features through layers to assess their importance. A common and simple method is to calculate the gradient at

each layer. and propagate it using the chain rule. However, such methods are hard to be extended to the problem space because the feature extraction is often not differentiable. We address this issue in our HTTP-structure alignment module. Surrogate model-based methods [7], [24], [11] attempt to fit the original model by training a simpler model. The main drawback of these methods is that the surrogate models may not accurately capture the complex behavior of the original models. Although some studies [7] advocate for increasing the complexity of surrogate models, their performance remains inferior to that of the original models. Perturbation-based methods [12], [90], [91] modify each feature and compute the changes in model prediction. Intuitively, modifying critical features will significantly impact the output. How to perturb the features is a key question considered in existing works. Common techniques include replacing features with a reference value [22] or permuting input features [23]. However, these techniques were initially developed for images or text and are not well-suited for HTTP requests.

X. Conclusion

We propose WebSpotter, a novel framework that achieves interpretable deep learning-based Web attack detection by localizing malicious payloads within HTTP requests. We develop a series of novel designs to achieve precise localization. We also demonstrate how to automatically generate WAF rules based on the localization results produced by WebSpotter. We evaluate WebSpotter on two public datasets and our newly constructed dataset, demonstrating its effectiveness and practical applications in mitigating attacks.

Ethics Considerations

This paper aims to defend against Web attacks, and does not introduce any security risks. To ensure a comprehensive evaluation of our method, we utilized data collected from real-world Web applications. The data collection was conducted by our industrial partner, and all personally identifiable information has been removed. We processed the data in an isolated environment and used it only for research purposes.

Acknowledgment

We thank the anonymous reviewers for their valuable comments. This work is supported in part by the National Key Research and Development Program of China under Grant 2023YFB3107400, the National Natural Science Foundation of China under Grant 62132011, 62472247, 62425201, 62521002, U24B20185, T2442014, U21B2018, U2441240 ("Ye Qisun" Science Foundation) and 62441238, the Shaanxi Province Key Industry Innovation Program under Grant 2023-ZDLGY-38 and 2021ZDLGY01-02, and the Ant Group Postdoctoral Programme. Qi Li is the corresponding author.

References

[1] L. U. Maheswari, G. Srivalli, G. Shivani, G. S. Nikitha, and K. Kaveri, "A novel web attack detection system for internet of things via ensemble classification," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 14, no. 03, pp. 834–845, 2023.

[2] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1963–1971, 2019.

[3] Z. Guo, Q. Shang, X. Li, C. Li, Z. Zhang, Z. Zhang, J. Hu, J. An, C. Huang, Y. Chen *et al.*, "Web-ftp: A feature transferring-based pre-trained model for web attack detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 37, no. 3, pp. 1495–1507, 2025.

[4] R. Tang, Z. Yang, Z. Li, W. Meng, H. Wang, Q. Li, Y. Sun, D. Pei, T. Wei, Y. Xu *et al.*, "Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks," in *IEEE INFOCOM*, 2020.

[5] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: A qualitative study of SOC analysts' perspectives on security alarms," in *USENIX Security*, 2022, pp. 2783–2800.

[6] F. Wei, H. Li, Z. Zhao, and H. Hu, "xNIDS: Explaining deep learning-based network intrusion detection systems for active intrusion responses," in *USENIX Security*, 2023, pp. 4337–4354.

[7] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *ACM CCS*, 2018, pp. 364–379.

[8] D. Han, Z. Wang, W. Chen, Y. Zhong, S. Wang, H. Zhang, J. Yang, X. Shi, and X. Yin, "Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications," in *ACM CCS*, 2021.

[9] Y. He, J. Lou, Z. Qin, and K. Ren, "Finer: Enhancing state-of-the-art classifiers with feature attribution to facilitate security analysis," in *ACM CCS*, 2023, pp. 416–430.

[10] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning*. PMLR, 2017, pp. 3145–3153.

[11] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, "Ai/ml for network security: The emperor has no clothes," in *ACM CCS*, 2022, pp. 1537–1551.

[12] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *CVPR*, 2017, pp. 3429–3437.

[13] (2023) Cve-2023-26469. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-26469

[14] M. Zhang, B. Xu, S. Bai, S. Lu, and Z. Lin, "A deep learning method to detect web attacks using a specially designed cnn," in *Neural Information Processing: 24th International Conference (ICONIP)*. Springer, 2017, pp. 828–836.

[15] Y. Fang, Y. Qiu, L. Liu, and C. Huang, "Detecting webshell based on random forest with fasttext," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, 2018, pp. 52–56.

[16] S. Hao, J. Long, and Y. Yang, "Bl-ids: Detecting web attacks using bi-lstm model based on deep learning," in *International conference on security and privacy in new computing environments*. Springer, 2019, pp. 551–563.

[17] R. Fielding, M. Nottingham, and J. Reschke, "Rfc 9112: Http/1.1," 2022.

[18] T. Berners-Lee, R. Fielding, and L. Masinter, "Rfc 3986: Uniform resource identifier (uri): Generic syntax," 2005.

[19] Flatten json. [Online]. Available: https://github.com/amirziai/flatten

[20] P. Li, Y. Wang, Q. Li, Z. Liu, K. Xu, J. Ren, Z. Liu, and R. Lin, "Learning from limited heterogeneous training data: Meta-learning for unsupervised zero-day web attack detection across web domains," in *ACM CCS*, 2023, pp. 1020–1034.

[21] T. Liu, Y. Qi, L. Shi, and J. Yan, "Locate-then-detect: Real-time web attack detection via attention-based deep neural networks." in *IJCAI*, 2019, pp. 4725–4731.

[22] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and explaining concept drift samples for security applications," in *USENIX Security*, 2021, pp. 2327–2344.

[23] A. Fisher, C. Rudin, and F. Dominici, "All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously," *Journal of Machine Learning Research*, vol. 20, no. 177, pp. 1–81, 2019.

[24] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?" explaining the predictions of any classifier," in *ACM SIGKDD*, 2016, pp. 1135–1144.

[25] B. Jin, Z. Zhou, and J. Zou, "On the saturation phenomenon of stochastic gradient descent for linear inverse problems," *SIAM/ASA Journal on Uncertainty Quantification*, vol. 9, no. 4, pp. 1553–1588, 2021.

[26] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 3319–3328.

[27] S. Ding, H. Xu, and P. Koehn, "Saliency-driven word alignment interpretation for neural machine translation," in *Proceedings of the Fourth Conference on Machine Translation*, 2019, pp. 1–12.

[28] M. Srivastava, T. Hashimoto, and P. Liang, "Robustness to spurious correlations via human annotations," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9109–9119.

[29] Z. Nussbaum, J. X. Morris, B. Duderstadt, and A. Mulyar, "Nomic embed: Training a reproducible long context text embedder," *arXiv preprint arXiv:2402.01613*, 2024.

[30] A. Kusupati, G. Bhatt, A. Rege, M. Wallingford, A. Sinha, V. Ramanujan, W. Howard-Snyder, K. Chen, S. Kakade, P. Jain *et al.*, "Matryoshka representation learning," *NeurIPS*, vol. 35, pp. 30 233–30 249, 2022.

[31] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath, "Throwing darts in the dark? detecting bots with limited data using neural data augmentation," in *IEEE S&P*. IEEE, 2020, pp. 1190–1206.

[32] (2024) SecLang syntax. [Online]. Available: https://coraza.io/docs/seclang/syntax

[33] (2024) Modsecurity. [Online]. Available: https://modsecurity.org

[34] (2024) Coraza. [Online]. Available: https://coraza.io

[35] (2024) Haproxy waf. [Online]. Available: https://www.haproxy.com/solutions/web-application-firewall

[36] (2024) Ibm security verify access waf. [Online]. Available: https://www.ibm.com/docs/en/sva/10.0.8?topic=firewall-overview

[37] (2024) Vmware avi load balancer waf. [Online]. Available: https://docs.vmware.com/en/VMware-Avi-Load-Balancer/30.2/WAF-Guide/GUID-0BB09D0C-D2A0-459F-9402-EE1485DFEBA1.html

[38] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting." in *OSDI*, vol. 4, 2004, pp. 4–4.

[39] C. Kreibich and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," *ACM SIGCOMM computer communication review*, vol. 34, no. 1, pp. 51–56, 2004.

[40] F. Nielsen and F. Nielsen, "Hierarchical clustering," *Introduction to HPC with MPI for Data Science*, pp. 195–211, 2016.

[41] (2024) Http dataset csic 2010. [Online]. Available: https://web.archive.org/web/20220106004901/http://www.isi.csic.es/dataset/

[42] C. Raıssi, J. Brissaud, G. Dray, P. Poncelet, M. Roche, and M. Teisseire, "Web analyzing traffic challenge: description and results," in *Proceedings of the ECML/PKDD*, 2007, pp. 47–52.

[43] (2024) Cve-2021-44228. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228

[44] (2024) Payloadsallthethings. [Online]. Available: https://github.com/swisskyrepo/PayloadsAllThe-Things

[45] (2024) xray. [Online]. Available: https://github.com/chaitin/xray

[46] (2024) Burp bounty. [Online]. Available: https://burpbounty.net

[47] T. Liu, Y. Qi, L. Shi, and J. Yan, "Locate-then-detect: Real-time web attack detection via attention-based deep neural networks," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.

[48] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017, pp. 1251–1258.

[49] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *ECCV*. Springer, 2016, pp. 21–37.

[50] J. M. H. Noothout, B. D. D. Vos, J. M. Wolterink, E. M. Postma, and I. Igum, "Deep learning-based regression and classification for automatic landmark localization in medical images," *IEEE Transactions on Medical Imaging*, 2020.

[51] C. H. Lampert, M. B. Blaschko, and T. Hofmann, "Beyond sliding windows: Object localization by efficient subwindow search," in *CVPR*. IEEE, 2008, pp. 1–8.

[52] O. O. Koyejo, N. Natarajan, P. K. Ravikumar, and I. S. Dhillon, "Consistent multilabel classification," *NeurIPS*, 2015.

[53] M. Scott, L. Su-In *et al.*, "A unified approach to interpreting model predictions," *NeurIPS*, vol. 30, pp. 4765–4774, 2017.

[54] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[55] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *NeurIPS*, pp. 27 730–27 744, 2022.

[56] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large language model guided protocol fuzzing," in *NDSS*, 2024.

[57] J. He and M. Vechev, "Large language models for code: Security hardening and adversarial testing," in *ACM CCS*, 2023, pp. 1865–1879.

[58] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *IEEE S&P*. IEEE, 2023, pp. 2339–2356.

[59] R. Fang, R. Bindu, A. Gupta, Q. Zhan, and D. Kang, "Llm agents can autonomously hack websites," *arXiv preprint arXiv:2402.06664*, 2024.

[60] ——, "Teams of LLM agents can exploit zero-day vulnerabilities," *arXiv preprint arXiv:2406.01637*, 2024.

[61] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, B. Chang *et al.*, "A survey on in-context learning," in *EMNLP*, 2024, pp. 1107–1128.

[62] Y. Hao, Y. Sun, L. Dong, Z. Han, Y. Gu, and F. Wei, "Structured prompting: Scaling in-context learning to 1,000 examples," *arXiv preprint arXiv:2212.06713*, 2022.

[63] (2018) Forceful browsing. [Online]. Available: https://capec.mitre.org/data/definitions/87.html

[64] (2024) Cloudflare waf. [Online]. Available: https://developers.cloudflare.com/waf/managed-rules/reference/owasp-core-ruleset/

[65] (2024) Cloud armor. [Online]. Available: https://cloud.google.com/armor/docs/rule-tuning

[66] (2024) Microsoft azure waf. [Online]. Available: https://learn.microsoft.com/en-us/azure/web-application-firewall/ag/application-gateway-crs-rulegroups-rules

[67] M. Hemmati and M. A. Hadavi, "Bypassing web application firewalls using deep reinforcement learning." *ISeCure*, vol. 14, no. 2, 2022.

[68] D. Appelt, C. D. Nguyen, A. Panichella, and L. C. Briand, "A machine-learning-driven evolutionary approach for testing web application firewalls," *IEEE Transactions on Reliability*, 2018.

[69] M. Amouei, M. Rezvani, and M. Fateh, "Rat: Reinforcement-learning-driven and adaptive testing for vulnerability discovery in web application firewalls," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3371–3386, 2021.

[70] L. Demetrio, A. Valenza, G. Costa, and G. Lagorio, "Waf-a-mole: evading web application firewalls through adversarial machine learning," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 1745–1752.

[71] Z. Qu, X. Ling, T. Wang, X. Chen, S. Ji, and C. Wu, "Advsqli: Generating adversarial sql injections against real-world waf-as-a-service," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2623–2638, 2024.

[72] Waf-brain. [Online]. Available: https://github.com/BBVA/waf-brain

[73] A. S. Li, A. Iyengar, A. Kundu, and E. Bertino, "Revisiting concept drift in windows malware detection: Adaptation to real drifted malware with minimal samples," in *NDSS*, 2025.

[74] Y. Chen, Z. Ding, and D. Wagner, "Continuous learning for android malware detection," in *USENIX Security*, 2023, pp. 1127–1144.

[75] Y. Qing, Q. Yin, X. Deng, Y. Chen, Z. Liu, K. Sun, K. Xu, J. Zhang, and Q. Li, "Low-quality training data only? a robust framework for detecting encrypted malicious network traffic," in *NDSS*, 2024.

[76] J. Liang, W. Zhao, and W. Ye, "Anomaly-based web attack detection: a deep learning approach," in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, 2017.

[77] M. Gniewkowski, H. Maciejewski, T. R. Surmacz, and W. Walentynowicz, "Http2vec: Embedding of http requests for detection of anomalous traffic," *arXiv preprint arXiv:2108.01763*, 2021.

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

[79] A. M. Vartouni, S. S. Kashi, and M. Teshnehlab, "An anomaly detection method to detect web attacks using stacked auto-encoder," in *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*. IEEE, 2018, pp. 131–134.

[80] S. Park, M. Kim, and S. Lee, "Anomaly detection for http using convolutional autoencoders," *IEEE Access*, 2018.

[81] A. Moradi Vartouni, M. Teshnehlab, and S. Sedighian Kashi, "Leveraging deep neural networks for anomaly-based web application firewall," *IET Information Security*, vol. 13, no. 4, pp. 352–361, 2019.

[82] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1096–1109, 2017.

[83] J. Liu, X. Song, Y. Zhou, X. Peng, Y. Zhang, P. Liu, D. Wu, and C. Zhu, "Deep anomaly detection in packet payload," *Neurocomputing*, vol. 485, pp. 205–218, 2022.
[84] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.
[85] R. Girshick, "Fast r-cnn," in *CVPR*, 2015, pp. 1440–1448.
[86] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *NeurIPS*, vol. 28, 2015.
[87] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016, pp. 779–788.
[88] A. Adadi and M. Berrada, "Peeking inside the black-box: a survey on explainable artificial intelligence (xai)," *IEEE access*, vol. 6, pp. 52 138–52 160, 2018.
[89] J. Li, X. Chen, E. Hovy, and D. Jurafsky, "Visualizing and understanding neural models in nlp," *arXiv preprint arXiv:1506.01066*, 2015.
[90] J. Li, W. Monroe, and D. Jurafsky, "Understanding neural networks through representation erasure," *arXiv preprint arXiv:1612.08220*, 2016.
[91] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*. Springer, 2014, pp. 818–833.
[92] J. He, K. Chen, G. Meng, J. Zhang, and C. Li, "Good-looking but lacking faithfulness: Understanding local explanation methods through trend-based testing," in *ACM CCS*, 2023, pp. 431–445.

APPENDIX

A. Dataset Statistics

The statistics of datasets are shown in Table VII. Note that, FPAD-OOD involves only the testing set and does not contain normal requests. We test the localization performance on FPAD-OOD based on the training data of FPAD.

The statistics of real-world attacks collected from two Web applications are shown in Table VIII.

TABLE VII: Statistics of our experimental datasets.

Dataset	Class	# Training	# Testing	# Testing Attacks
CSIC	2	25701	8576	3935
PKDD	8	40092	10024	3041
FPAD	5	38179	16363	10771
FPAD-OOD	4	-	21306	21306

TABLE VIII: Statistics of the real-world attacks collected from two Web applications.

Application	Collection Time	# Attack requests
G1	5 Days	5.87K
G2	18 Days	1.83K

B. Construction Details of the FPAD Dataset

The FPAD dataset directly inherits the normal requests from the CSIC dataset, while the attack requests are regenerated. To generate valid attacks, we collect more than 1580 malicious payloads from existing Web attack scanners and open-source projects. These malicious payloads can be applied to normal requests through various techniques, such as concatenation and substitution, to form new attack requests. We utilize an existing Web security tool, Burp Bounty, which provides off-the-shelf implementations for this purpose. Moreover, data transformation techniques are applied to enhance the diversity of attack requests. Specifically, all parameters along with their potential values are extracted from the non-malicious fields in both CSIC normal and abnormal requests, and assigned to the generated attack requests. Finally, we perform dataset splitting and create the FPAD training set, FPAD testing set, and FPAD-OOD set. In the training set and testing set, the attack requests are generated based on the same set of malicious payloads, but they differ in their normal payloads. For the OOD dataset, we use a separate collection of malicious payloads. These payloads are carefully selected based on expert experience and exhibit greater complexity and distinct characteristics compared to those in the training set.

TABLE IX: Detection performance with different structures.

Model	# Params	Dataset	Pre	Rec	F1
TextCNN	1.37M	FPAD	0.999	0.999	0.999
		FPAD-OOD	0.973	0.967	0.970
Bi-LSTM	4.26M	FPAD	0.999	0.999	0.999
		FPAD-OOD	0.951	0.935	0.943
FastText	0.06M	FPAD	0.955	0.955	0.954
		FPAD-OOD	0.932	0.895	0.913

TABLE X: Localization performance of WebSpotter with different model structures.

Method	FPAD			FPAD-OOD		
	Pre	Rec	F1	Pre	Rec	F1
TextCNN 1%	0.987	0.993	0.988	0.962	0.981	0.964
Bi-LSTM 1%	0.947	0.988	0.960	0.934	0.974	0.944
FastText 1%	0.948	0.986	0.960	0.946	0.977	0.954
TextCNN 100%	0.997	0.997	0.997	0.993	0.987	0.988
Bi-LSTM 100%	0.995	0.997	0.996	0.985	0.986	0.985
FastText 100%	0.995	0.996	0.995	0.990	0.989	0.989

C. Selection of the Detection Model

We also evaluate two other DL-based Web attack detection models in addition to TextCNN, i.e., Bi-LSTM [16] and FastText [15]. Table IX shows their attack detection performance on FPAD and FPAD-OOD datasets. It can be seen that TextCNN achieves the best detection performance, especially on the FPAD-OOD dataset. Consequently, we select TextCNN for evaluations in Section VII. Table X further shows the localization performance of WebSpotter on these detection models. It can be observed that our method consistently demonstrates high localization performance across various models, which confirms its generality and effectiveness.

D. Implementation details of Existing methods

This section provides the implementation details of existing methods mentioned in our work. We first describe the implementation details of baseline methods, including LTD, SSD, REG and CLS.

LTD. LTD [47] proposes a payload locating network (PLN) to identify malicious payloads within HTTP requests. PLN is inspired by the region proposal techniques in image processing [86]. It utilizes a sliding window mechanism to generate a set of candidate regions (namely anchors) on the feature map and employs two sibling convolutional layers to predict the

position and classification confidence of each region. The output includes the coordinates of candidate regions and their corresponding probabilities of containing a malicious payload. We follow the core settings of the original paper. Each input is embedded into a tensor of shape $1000 \times 8 \times 1$ and processed to generate a 32×8 feature map. Then, $32 \times 75 = 2400$ anchors are generated based on the feature map. During the training phase, we also balance the data by limiting the proportion of negative to positive anchors to a maximum ratio of 3:1.

SSD. SSD [49] is originally designed for object localization, and we adapt it for Web attack detection based on the design of LTD. SSD generates multi-scale feature maps by applying a series of down-sampling blocks and then produces anchors on each feature map. Unlike LTD, SSD directly predicts the final class label for each anchor, rather than performing a binary classification to determine whether it contains malicious payloads.

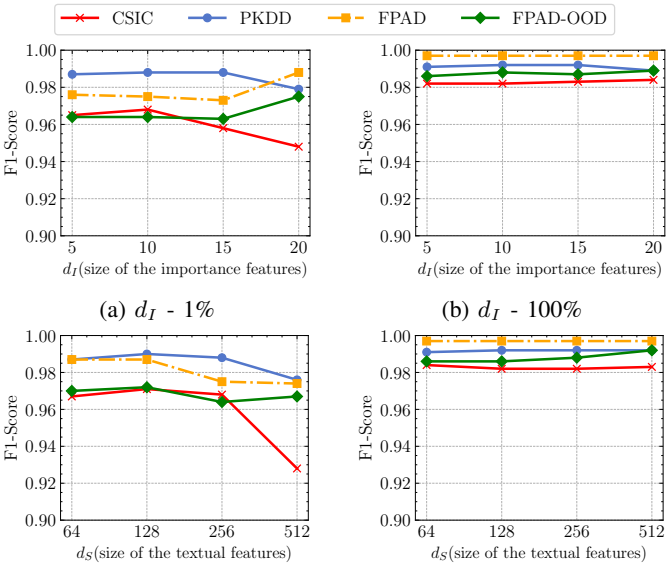
REG. REG[50] formulates a regression task to predict the locations of malicious payloads as a set of string indices. We enhance TextCNN with two fully connected heads, enabling it to simultaneously perform the regression task for malicious payload localization and the classification task for attack detection.

CLS. CLS[51] treats each MSU as an independent entity and assesses its potential maliciousness. It first segments the HTTP request into MSUs and extracts textual semantic features for each MSU using the Nomic-embed technique. A random forest classifier is then trained based on these features to predict whether each MSU contains a malicious payload. The model architecture and hyperparameters used in CLS are identical to those in WebSpotter.

The localization results of LTD, SSD, and REG consist of a set of string indices representing the start and end positions of malicious payloads within the HTTP request. To process these predicted regions, we first apply Non-Maximum Suppression (NMS) to eliminate overlapping and redundant regions. These regions are then mapped to minimal semantic units (MSUs). Specifically, an MSU is identified as malicious if its intersection over union (IoU) with a predicted region exceeds a predefined threshold, which is set at 0.7. If a predicted region does not sufficiently overlap any MSU, the MSU with the highest IoU to this region is identified as malicious. Moreover, considering that WebSpotter leverages full classification labels during training the detection model, we replace the embedding layers of these three baselines with the trained embedding layer from the detection model to ensure a fair comparison.

Next, we elaborate on the implementation details of interpretability methods referenced in our ablation study.

SHAP. SHAP[53] uses Shapley values from cooperative game theory to quantify feature contributions. We use KernelSHAP, which is a model-agnostic method that approximates Shapley values using synthetic samples, and set the number of synthetic samples to 500. We follow the previous studies [92] to perturb all embedding features associated with each token and obtain token importance scores. Then, we utilize the protocol-level aggregation from our method to further compute the MSU importance score.



(a) d_I - 1% (b) d_I - 100%

(c) d_S - 1% (d) d_S - 100%

Fig. 10: Impact of d_I and d_S on localization performance with 1% and 100% labeling overhead.

LEMNA. LEMNA[7] is a surrogate model-based method that uses a mixture regression model with fused lasso to approximate local decision boundaries of deep learning models. We follow the original paper and set the number of synthesized samples to 500. The fused lasso regression is implemented using the genlasso package in R. Additionally, we group all embedding features associated with each token into a unit and calculate an importance score for each token.

CADE. CADE[22] identifies important features by modifying the features of samples and observing the output changes. Following this idea, we replace each original token of a request with a padding token and compute the importance score of each token by measuring the differences in the model's output category prediction probabilities.

E. Hyperparameter Sensitivity

To evaluate the hyperparameter sensitivity of WebSpotter, we analyze the impact of two key parameters on localization performance: the size d_I of the importance features and the size d_S of the textual semantic features.

We vary d_I from 5 to 20 in Figures 10a and 10b. d_I has a negligible influence on the localization performance. For example, the CSIC dataset showed the largest performance change, with only a 0.02 difference between the best and worst cases. Increasing d_I does not always result in improved localization performance, since the localization model primarily depends on the importance score of the target MSU, with sibling MSU scores providing supplementary information. The performance remains stable as long as d_I is sufficiently large to include the most relevant sibling MSUs.

TABLE XI: Jaccard Index of various variants of WebSpotter.

Method	CSIC	PKDD	FPAD	FPAD-OOD
WebSpotter w/ CADE	0.884	0.978	0.957	0.941
WebSpotter w/ SHAP	0.439	0.956	0.925	0.902
WebSpotter w/ Lemna	0.817	0.897	0.948	0.932
WebSpotter w/ VG	0.876	0.906	0.957	0.947
WebSpotter w/ S.S.	0.755	0.894	0.961	0.938
WebSpotter w/ FINER	0.867	0.910	0.973	0.955
WebSpotter w/o T.F.	0.965	0.970	0.923	0.890
WebSpotter w/o L.M.	0.948	0.961	0.930	0.898
Original WebSpotter	0.968	0.983	0.983	0.959

GET /tienda1/miembros/editar.jsp?modo=registro&login=%22+AND+%221%22%3D%221&password=trariloNgO&nombre=Gelsomino&apellidos=Zuazo+Suela&email=gopi_belcher%40bahiadelasirenas.mq&dni=15074727K&direccion=Baixada+Can+Llobet+124%2C+&ciudad=San+Miguel+de+Bernuy&cp=32089&provincia=Baleares+%28Illes%29&ntc=1285539651029120&B1=Registrar HTTP/1.1
…

Fig. 11: An example showing the limitation of *"WebSpotter without localization models (L.M.)"*, where the malicious payloads located by *"WebSpotter without L.M."* are highlighted in red and the `email` parameter is incorrectly identified.

We vary d_S from 64 to 512 in Figures 10c and 10d. When the labeling overhead is 100%, the localization performance remains stable. At 1% labeling overhead, stability is maintained for embedding sizes below 256, but performance degrades as d_S increases to 512. This is because machine learning models with high complexity require more training data. If the labeling overhead is too low, the available training data becomes insufficient to effectively optimize the models.

F. Additional Results of the Ablation Study

We provide the results of Jaccard Index of ablation study in Table XI. These results exhibit a trend similar to that of the F1-scores shown in Table III, and demonstrate the effectiveness of each individual module of WebSpotter.

We present a representative example in Figure 11 to further illustrate why "WebSpotter without localization models" is inadequate for precise localization. In this example, although the detection model correctly identifies the request as an attack, it considers the `email` field to have a significant impact on the prediction, which is mainly because the content of `email` parameter is unknown for the detection model and contains some special characters. Consequently, the `email` parameter is incorrectly identified as malicious. In contrast, the original WebSpotter successfully recognizes the `email` parameter as benign by incorporating the textual semantics of MSUs.

G. Computational Overhead

We evaluate the training and inference time of different methods when the labeling overhead is set to 1%. As shown in Figure 12, our method and CLS have computational overheads of the same order of magnitude, while SSD and LTD incur

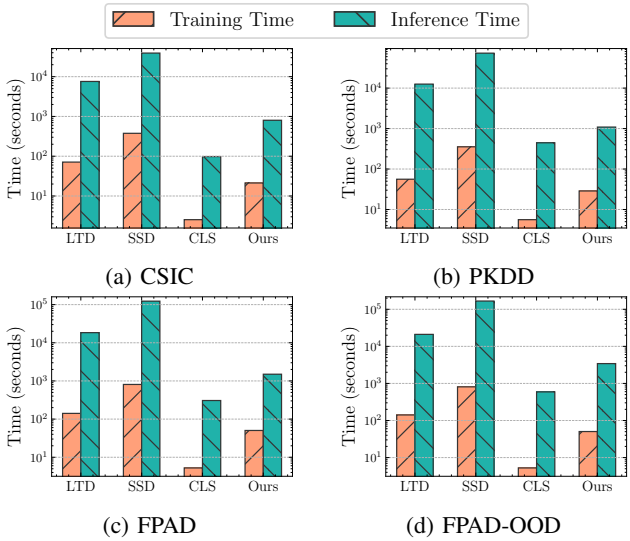


Fig. 12: Computational overhead of different methods.

Prompt

Definition: Your task is to locate malicious payloads in an HTTP request. The HTTP request will be divided into minimal semantic units (MSUs), and the input is an array of strings, where each string represents an MSU of the HTTP request. The output should be a dictionary in JSON format, where the key is a string from the input array, and the value is 0 or 1. A value of 0 indicates that the unit does not contain malicious payloads, and a value of 1 means otherwise.

Example-1:
Input: ["POST", "/tienda1", "/<marquee loop=1 width=0 onfinish=alert(1)>", "/anadir.jsp", "id=2", "nombre=Iber", "precio=5003", "cantidad=64", "B1=Entrar"]
Output: {"POST": 0, "/tienda1": 0, "/<marquee loop=1 width=0 onfinish=alert(1)>": 1, "/anadir.jsp": 0, "id=2": 0, "nombre=Iber": 0, "precio=5003": 0, "cantidad=64": 0, "B1=Entrar": 0}

Example-2 :

...

User Input : {input}

Fig. 13: Prompt for LLM-based malicious payload localization.

costs at least one order of magnitude higher. For example, on the FPAD dataset, the inference time per attack request is 0.1011 seconds for our method, 0.0413 seconds for CLS, 1.1659 seconds for LTD, and 6.7864 seconds for SSD. Compared to CLS, our method requires additional time to compute the MSU importance. The higher computational cost of SSD and LTD can be attributed to the need to handle a large number of candidate regions, e.g., LTD generates approximately 2400 candidate regions per request.

H. LLM Prompt

The prompt for LLM-based malicious payload localization is shown in Figure 13.

APPENDIX A
ARTIFACT APPENDIX

Our artifact includes the source code for WebSpotter, along with the scripts for setting up the environment, the datasets used in the experiments (including our newly constructed datasets), and detailed instructions for reproducing the main results presented in the paper.

A. Description & Requirements

1) How to access: The source code and datasets used in our study are publicly available at (i) our GitHub repository: *https://github.com/Sec-AI-research/WebSpotter* and (ii) Zenodo: *https://doi.org/10.5281/zenodo.16978408*.

2) Hardware dependencies: Our artifact needs a machine with at least 16GB RAM. Although the experiment can be run directly on the CPU, it is highly recommended to use a system with an NVIDIA GPU. We used the NVIDIA GeForce RTX 4090 GPU with 24GB VRAM. This artifact is compatible with other NVIDIA GPUs that provide sufficient memory (approximately 12GB VRAM).

3) Software dependencies: The artifact uses Python 3.9. All required packages are listed in our GitHub repository at requirements.txt, which can be used to directly construct the Python environment via pip.

4) Benchmarks: This artifact uses the TextCNN model together with the FPAD and FPAD-OOD datasets for evaluation. We have made all datasets publicly available. For more details, please refer to Section VII.A of the paper.

B. Artifact Installation & Configuration

To streamline the setup process, we provide a script for automatic environment configuration. After cloning the GitHub repository, navigate to the project root directory and run the following commands to create the environment:

```
conda create -n webspotter python=3.9
conda activate webspotter
pip install -r requirements.txt
```

C. Experiment Workflow

The experiments of our artifact consists of three main stages: (i) training a Web attack detection model, (ii) computing the importance scores of minimal semantic units (MSUs) within HTTP requests, and (iii) training a localization model to identify the locations of malicious payloads.

D. Major Claims

- **(C1):** WebSpotter achieves highly accurate malicious payload localization when the labeling overhead is set to 1%. This is supported by experiment (E1), with evaluation results reported in Table I of the paper.
- **(C2):** WebSpotter maintains stable localization performance as the labeling overhead changes. This is demonstrated in experiment (E2), with trends shown in Figure 7 of the paper.
- **(C3):** WebSpotter exhibits strong generalization capability in handling unseen attack patterns. This is validated through experiment (E3), with detailed results provided in Table I and Table II of the paper.

E. Evaluation

1) Experiment (E1): [Localization Performance of WebSpotter with 1% Labeling Overhead] [2 human-minutes + 40 compute-minutes]: This experiment demonstrates WebSpotter's ability to achieve accurate malicious payload localization using only 1% of location-labeled attack requests for training. The evaluation results are reported in Table I.

[Preparation] First, train a basic DL-based Web attack detection model. We provide a TextCNN model for this purpose. The training command is:

```
python classification/run.py --tmp_dir
    datasets/FPAD --tmp_model tmp_model --
    dataset fpad
```

Then, compute the importance scores of minimal semantic units (MSUs), which are required for training and evaluating the localization model. The following two commands generate the importance scores for the training and testing sets, respectively:

```
python localization/post_explain/
    run_explain.py --model_path tmp_model/
    textcnn-700-FPAD-512-None-42.pth --
    outputdir post_explain_result/fpad/
    train --dataset fpad --test_path
    datasets/FPAD/train.jsonl

python localization/post_explain/
    run_explain.py --model_path tmp_model/
    textcnn-700-FPAD-512-None-42.pth --
    outputdir post_explain_result/fpad/
    test --dataset fpad --test_path
    datasets/FPAD/test.jsonl
```

[Execution] Train the localization model using 1% of location-labeled attack requests, and evaluate its localization performance. The command is as follows:

```
python localization/binary_based/run.py --
    feature_method score_sort_with_textemb
     --dataset fpad --train_path
    post_explain_result/fpad/train/train.
    jsonl_withscore --test_path
    post_explain_result/fpad/test/test.
    jsonl_withscore --output_path
    binary_result/fpad_0.01 --sample_rate
    0.01
```

[Results] The evaluation metrics will be printed in the terminal and the localization results for each request will be saved in `binary_result/fpad_0.01`. Expected metrics for FPAD dataset include Precision > 0.98, Recall > 0.98, and F1-score > 0.98.

2) Experiment (E2): [Localization Performance under Varying Labeling Overhead] [2 human-minutes + 40 compute-minutes]: This experiment evaluates WebSpotter's performance when the labeling overhead changes. The results are reported in Figure 7 of the paper.

[Preparation] Evaluators can reuse the detection model and MSU importance scores generated in (E1).

[Execution] Train the localization model using different proportions of location-labeled attack requests, and evaluate its localization performance. The labeling overhead can be controlled via the `--sample_rate` argument, such as 0.01, 0.1, 0.5, and 1. An example command is as follows:

```
python localization/binary_based/run.py --
    feature_method score_sort_with_textemb
     --dataset fpad --train_path
    post_explain_result/fpad/train/train.
    jsonl_withscore --test_path
    post_explain_result/fpad/test/test.
    jsonl_withscore --output_path
    binary_result/fpad_0.5 --sample_rate
    0.5
```

[Results] The evaluation metrics will be printed in the terminal and the localization results for each request will be saved in `output_path`. Expected metrics for above example include Precision > 0.99, Recall > 0.99, and F1-score > 0.99.

3) Experiment (E3): [Localization Performance of WebSpotter on Unseen Attacks] [1 human-minutes + 20 compute-minutes]: This experiment reproduces the WebSpotter on the FPAD-OOD dataset to evaluate its generalization capability when applied to unseen attack patterns. The results are shown in Table I and Table II of the paper.

[Preparation] Evaluators can reuse the detection model trained in (E1).

[Execution] First, compute the importance scores of minimal semantic units (MSUs) on the FPAD-OOD dataset. Run the following command:

```
python localization/post_explain/
    run_explain.py --model_path tmp_model/
    textcnn-700-FPAD-512-None-42.pth --
    outputdir post_explain_result/fpad-ood
    /test --dataset fpad-ood --test_path
    datasets/FPAD-OOD/test.jsonl
```

Then, run WebSpotter on the FPAD-OOD dataset using the following command:

```
python localization/binary_based/run.py --
    feature_method score_sort_with_textemb
     --dataset fpad-ood --train_path
    post_explain_result/fpad/train/train.
    jsonl_withscore --test_path
    post_explain_result/fpad-ood/test/test
    .jsonl_withscore --output_path
    binary_result/fpad_ood --sample_rate
    0.01
```

[Results] The evaluation metrics will be printed in the terminal and the localization results for each request will be saved in `binary_result/fpad_ood`. Expected metrics for FPAD-OOD dataset include Precision > 0.96, Recall > 0.96, and F1-score > 0.96.

F. Customization

The above experiments are conducted using the FPAD dataset. To switch to other dataset, evaluators can modify the corresponding command-line arguments, such as `--dataset`, `--train_path`, and `--test_path`, along with the relevant file or directory paths. Also, the proportion of location-labeled training data used for the localization model can be adjusted via the `--sample_rate` argument.