Learning from Limited Heterogeneous Training Data: Meta-Learning for Unsupervised Zero-Day Web Attack Detection across Web Domains

Peiyang Li*
Tsinghua University & BNRist
li-py23@mails.tsinghua.edu.cn

Ye Wang*
Tsinghua University & BNRist
wangye22@mails.tsinghua.edu.cn

Qi Li✉
Tsinghua University
qli01@tsinghua.edu.cn

Zhuotao Liu
Tsinghua University
zhuotaoliu@tsinghua.edu.cn

Ke Xu
Tsinghua University
xuke@tsinghua.edu.cn

Ju Ren
Tsinghua University
renju@tsinghua.edu.cn

Zhiying Liu
Tencent
louiszyliu@tencent.com

Ruilin Lin
Tencent
ruilin@tencent.com

ABSTRACT

Recently unsupervised machine learning based systems have been developed to detect zero-day Web attacks, which can effectively enhance existing Web Application Firewalls (WAFs). However, prior arts only consider detecting attacks on specific domains by training particular detection models for the domains. These systems require a large amount of training data, which causes a long period of time for model training and deployment. In this paper, we propose RETSINA, a novel meta-learning based framework that enables zero-day Web attack detection across different domains in an organization with limited training data. Specifically, it utilizes meta-learning to share knowledge across these domains, e.g., the relationship between HTTP requests in heterogeneous domains, to efficiently train detection models. Moreover, we develop an adaptive preprocessing module to facilitate semantic analysis of Web requests across different domains and design a multi-domain representation method to capture semantic correlations between different domains for cross-domain model training. We conduct experiments using four real-world datasets on different domains with a total of 293M Web requests. The experimental results demonstrate that RETSINA outperforms the existing unsupervised Web attack detection methods with limited training data, e.g., RETSINA needs only 5-minute training data to achieve comparable detection performance to the existing methods that train separate models for different domains using 1-day training data. We also conduct real-world deployment in an Internet company. RETSINA captures on average 126 and 218 zero-day attack requests per day in two domains, respectively, in one month.

*Both authors contributed equally to this research.

CCS'23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

KEYWORDS

Web attack detection; meta-learning; zero-day attacks

ACM Reference Format:
Peiyang Li, Ye Wang, Qi Li✉, Zhuotao Liu, Ke Xu, Ju Ren, Zhiying Liu, and Ruilin Lin. 2023. Learning from Limited Heterogeneous Training Data: Meta-Learning for Unsupervised Zero-Day Web Attack Detection across Web Domains. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS'23).* ACM, New York, NY, USA, 16 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Web services suffer from various Web attacks (e.g., SQL injection). Web Application Firewalls (WAFs) [1, 3, 4, 53] have become the de facto Web attack defense mechanisms against the attacks. However, since WAFs detect Web attacks according to manually configured rules, the zero-day Web attacks generating unknown attack patterns can easily evade WAFs. Recently unsupervised machine learning systems [49, 60, 63] have been developed to complement existing WAFs to detect zero-day attacks. These systems learn patterns of benign Web requests and then identify zero-day attacks whose patterns deviate from benign requests.

However, these ML-based systems fail to align with the requirements of Web attack detection systems in large-scale, operational environments. First, an organization hosting Web services often maintains multiple Web domains for different Web applications. These domains normally exhibit heterogeneity because they enable different Web functionalities, e.g., Google Scholar and Gmail are two heterogeneous domains hosting different Google applications. However, existing methods [38, 40, 49, 60, 63, 66] only focus on developing the detection model for one specific domain, which are developed independently for one domain and can hardly be generalized to detect attacks on other domains. Second, the detection model requires frequent model retraining to avoid significant performance degradation [13, 14, 24, 30, 50] due to the Web services update [79]. According to our study on real-world Web services,

we find that most of the services are regularly updated, e.g., the average update interval of four real-world services in our study is only 3.79 days. It is difficult to retrain and update detection models due to the required long period of time of training data collection [60]. In addition, collecting such a large amount of production data incurs significant data preprocessing or privacy breach [43].

In this paper, we develop RETSINA, a novel meta-learning based framework for zero-day Web attack detection across multiple domains. RETSINA utilizes meta-learning to jointly train detection models for new deployed domains based on limited heterogeneous data. RETSINA exploits correlations between heterogeneous requests generated from various domains to build a universal detection model. Based on the universal detection model and limited requests from each domain, RETSINA generates a separate domain-specific detection model to detect attacks for the domain, which realize efficient model training and update for the domain.

However, it is challenging to utilize meta-learning to achieve zero-day Web attack detection for multiple domains. First, it is difficult to extract valid information from such requests due to the heterogeneity and complexity of processing HTTP requests across heterogeneous domains. Second, since lexical features of HTTP requests are not in the same semantic space among different domains [60, 63, 66, 74], we cannot easily capture feature correlations across Web domains. Third, it is not easy to transfer its knowledge to the domain-specific detection models according to the universal detection model because the structures of detection models vary for different domains.

In order to address these issues, we develop three key designs to enable knowledge sharing across heterogeneous domains. First, we develop an adaptive approach to facilitate semantic analysis of HTTP requests. It can automatically generate domain-specific strategies to eliminate redundant information that is not useful for attack detection and then produce valid tokens for training machine learning models. Second, we propose a feature representation approach for multi-domain requests. It performs token alignment operations by utilizing orthogonal transformations to measure the similarity of tokens across domains. Based on this similarity, it maps all tokens to the same feature space, which facilitates machine learning models to capture correlations and share knowledge across heterogeneous domains. Third, we develop a dedicated two-loop strategy for training the universal detection model, which updates parameters that can be inherited by the target domain. This ensures that the knowledge learned by the universal detection model is fully transferable to different target domains.

We evaluate RETSINA using four real-world datasets collected from four different domains that provide different types of Web services to demonstrate the effectiveness of RETSINA. In particular, we conduct a comparative analysis of RETSINA against two state-of-the-art unsupervised Web attack detection methods [60, 69], one of which is a knowledge-sharing-based framework that is proposed for multi-task security problems, and two supervised Web attack detection techniques [74, 81]. The experimental results demonstrate that RETSINA outperforms the existing methods with limited training data, e.g., RETSINA needs only 5-minute training data to achieve comparable detection performance for different domains to the existing methods that train separate models for different domains using 1-day training data. We perform ablation experiments to analyze how different modules of RETSINA contribute to the detection. We also deploy RETSINA in real-world production services in an Internet company. RETSINA detects on average 126 and 218 zero-day attack requests per day in two domains, respectively.

In summary, we make the following contributions:

- We propose RETSINA, an unsupervised framework to detect zero-day Web attacks by utilizing meta-learning, which is the first one to detect Web attacks across multiple domains with limited and heterogeneous data.
- We design an adaptive method to preprocess multi-domain Web requests, which automatically converts unstructured raw requests into structured token sequences for training and detection.
- We develop a multi-domain representation method to construct unified feature representations across different Web domains, which can be easily extended to enable detection for new domains.
- We conduct extensive evaluations with a total of 293M Web requests from 4 real-world domains owned by an Internet company. Evaluation results demonstrate the effectiveness of RETSINA for multiple heterogeneous Web domains with limited training data.
- We perform real-world deployment in real production and evaluate the performance for one month, detecting on average 126 and 218 zero-day attack requests per day, demonstrating the superiority of RETSINA with respect to effectiveness and robustness.

2 PROBLEM STATEMENT

We consider the scenario where an Internet company operates multiple frequently updated domains and consistently releases new domains. These domains exhibit heterogeneity since they have different functionalities to support different services. Meanwhile, these domains follow the same development specification and can share the same underlying interface (e.g., the authentication interface). Web attack detection models are needed for updated or newly developed domains. However, to keep up with the rapid domain release, only a limited number of data is available for these domains.

Our goal is to develop zero-day Web attack detection models for each *target domain* with limited training data, by leveraging the heterogeneous data collected from several *auxiliary domains*. While requests from the target domain are limited, each auxiliary domain has adequate request data. The developed detection model will work together with existing rule-based WAFs, i.e., detecting zero-day attacks that evade WAFs.

Similar to existing approaches (e.g., [60, 74]), we focus on analyzing URLs as well as the message bodies with the "x-www-form-urlencoded" content type, which covers a majority of Web attacks. To validate this issue, we manually analyze 352 Web vulnerabilities collected from a popular GitHub project [6], and find that 83.2% of these vulnerabilities can be detected by RETSINA. We do not consider the other types of message bodies since their formats are not restricted by the HTTP protocol. We also skip headers because they have limited information for attack detection, and more crucially, anomalies in headers can be identified by certain rules defined in traditional WAFs.

In this paper, we leverage the state-of-the-art unsupervised method, i.e., ZeroWall [60], as the basic detection model in our

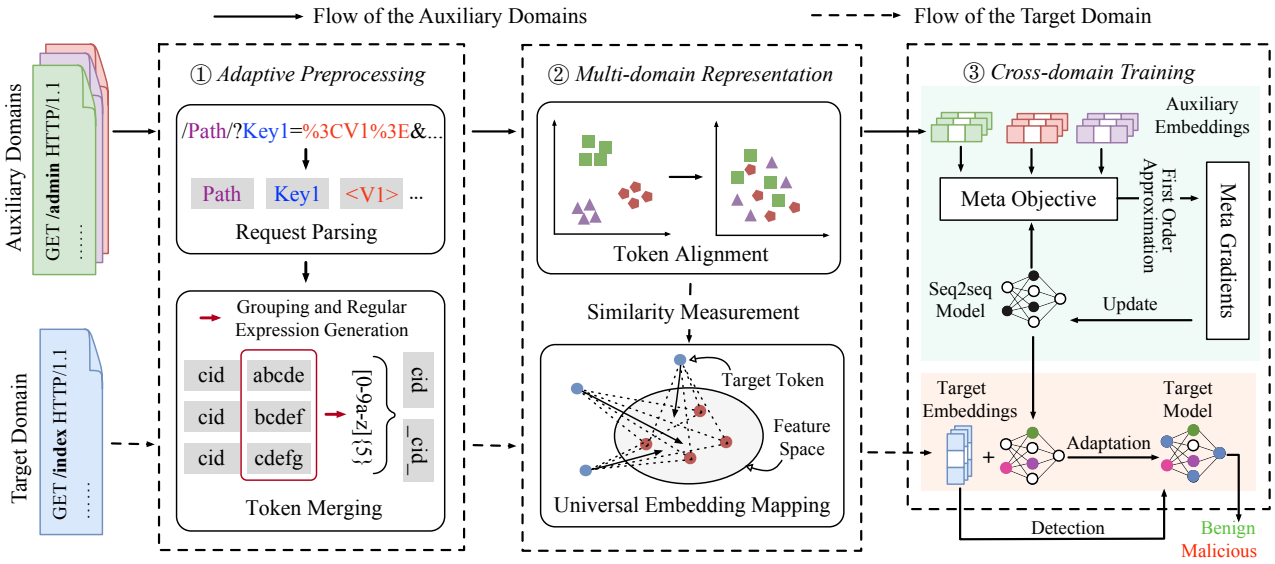


Figure 1: The overview of RETSINA.

framework. Zerowall leverages an embedding layer and a sequence-to-sequence (seq2seq) model to learn the benign pattern of Web requests by minimizing the reconstruction error between the input benign requests and the output reconstructed requests. Then it detects malicious Web requests whose patterns deviate from the benign pattern. In particular, the embedding layer converts each input sequence of requests into a vector sequence. The seq2seq model consists of a sequence-based encoder-decoder network that encodes a vector sequence into the latent feature space and decodes it back to a new vector sequence, and a generator that converts the vector sequence into the discrete output sequence.

3 FRAMEWORK OVERVIEW

We propose RETSINA for zero-day Web attack detection across heterogeneous domains with limited data. Our key insight is that semantic correlations exist between requests from different heterogeneous domains. The rationale behind semantic correlations is that domains operated by the same company follow the same development specification and share the same underlying interface. For example, the corporate data we collected for experiments established a series of cross-business parameter specifications for data governance. This is prevalent in other organizations. For example, multiple Google Web services including Google Cloud use five specific URL parameters with the prefix "utm" [2] to track user traffic. This insight indicates that the detection on one domain can benefit from the existing detection experiences from other domains.

Based on this insight, RETSINA leverages meta-learning to gain detection experience over models from multiple auxiliary domains. Since meta-learning learn patterns and relationships that are common across different domains, RETSINA is capable of training the detection models on target domains with limited data. RETSINA consists of the following three modules.

- **Adaptive Preprocessing.** Adaptive preprocessing converts each unstructured request into a structured token sequence, to facilitate semantic analysis during model training and detection. It first parses requests into token sequences based on punctuation and then merges tokens with inessential information according to the strategy automatically generated for each domain.
- **Multi-domain Representation.** Multi-domain representation projects tokens from heterogeneous domains into the same feature space in which the tokens with similar semantics are close. It chooses a domain as the base domain, and tokens in other domains are represented as a weighted sum of tokens in the base domain according to their semantic similarities.
- **Cross-domain Training.** Cross-domain training obtains a detection model for each target domain with limited training data. We first build a universal initial model using data from auxiliary domains and then adapt the model to the target domain using the limited data from that domain. Particularly, the universal initial model is trained to be adapted well to new domains by leveraging the idea of meta-learning.

Figure 1 shows the workflow of our framework from the perspective of auxiliary domains and target domains. For auxiliary domains, the adaptive preprocessing module converts requests from each domain into token sequences. Then the multi-domain representation module chooses a base domain among all domains and builds embeddings for all tokens according to the base domain. Using these embeddings, the cross-scenario training module obtains a universal initial model. For each target domain, the adaptive preprocessing module converts requests from this domain into token sequences. The multi-domain representation module builds embeddings for tokens in this domain according to the chosen base domain. Using these embeddings, in the cross-scenario training module, the universal initial model is then adapted to a target domain-specific model, on which we can perform online zero-day Web attack detection. It is worth noticing that, RETSINA finishes the computation

of auxiliary domains in advance, without knowing the target domain, thus enabling the detection model development for the target domain with limited data as well as a short development time.

4 DESIGN DETAILS

In this section, we present detailed designs of RETSINA.

4.1 Adaptive Preprocessing

Adaptive preprocessing aims at converting HTTP requests into token sequences, to facilitate semantic analysis during model training and attack detection. As mentioned earlier, HTTP requests are strings with complex semantics that vary across different domains. Therefore, adaptive preprocessing automatically generates domain-specific preprocessing strategies, to obtain token sequences with accurate semantics. In particular, for each domain, each request is first parsed into a token sequence. Then, by extracting the format of tokens, a preprocessing strategy, named as merging strategy, is automatically generated. It is applied to merge a number of tokens into fixed tokens. An example is shown in Figure 2.

Request Parsing. We parse each request into several parts according to the HTTP protocol and convert letters to lowercase. Specifically, we decode and parse a request into a path and multiple pairs of a key and a value in a query. Then we decode and split each part into tokens according to the punctuation (e.g., '/') and space.

Token Merging. We now merge tokens with similar semantics into the same tokens to obtain tokens with accurate semantics. We observe that frequently, the values of a key can be uncertain, and may even evolve with time, e.g., the number of distinct values of key "`androidId`" can be extremely huge. These values complicate the HTTP requests, but provide no additional benefits to attack detection since they have almost the same semantics. We regard these values as *inessential* tokens and merge them as the same tokens. In particular, we perform the following steps to merge these inessential tokens: i) We first group values by their keys to obtain each key's values. We identify keys whose number of distinct values is large, then for each such key, a regular expression is generated to match all its values for merging. Notably, the regular expression should be as small as possible to extract accurate semantics best. For example, for a set of values that only contains numbers, "`[0-9]+`" is generated instead of "`[0-9a-z]+`". For a simple implementation, we devise several common-used regular expressions, such as for numbers, and conduct auto selection among them. ii) After obtaining the regular expression of a key, for each of its values, we merge the value token in the token sequence using the same token (e.g., "`_aId_`" for key "`androidId`" in Figure 2) if the value matches its regular expression. Please refer to Appendix A for more details about this process.

In addition, we replace all low-frequency tokens using the same token "`_other_`", to allow our preprocessing to deal with previously unseen tokens. Note that the "`_other_`" token category generally implies that the requests are anomalous, since the low-frequency tokens with benign values have been merged according to our regular expressions. Given a collection of HTTP requests as our training data, besides identifying the regular expression during token merging, we also identify a *token set* which contains all tokens in the token sequences after request parsing and token merging. When new HTTP requests come, we parse these requests and merge tokens according to the regular expression. We also replace tokens that are not in the token set using "`_other_`".

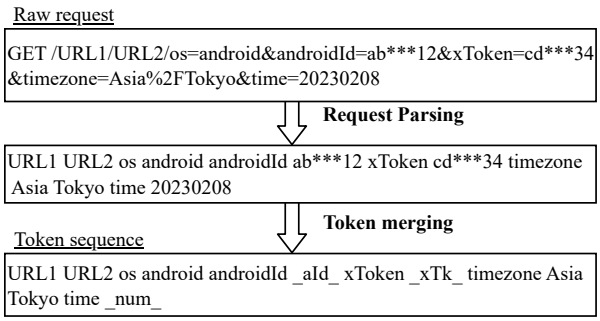


Figure 2: One example of adaptive preprocessing.

Our preprocessing automatically generates preprocessing strategies for each domain to merge inessential tokens. While accurately extracting semantics for each domain, it can be easily adapted to new domains without expertise effort.However, existing approaches use general preprocessing strategies for all domains, extracting inadequate semantics to differentiate abnormal when the strategies are too loose [38, 60], or retaining overly complex semantics to be learned and generalized when the strategies are too strict [49, 63, 74]. We provide a detailed example for comparison in Appendix A.

4.2 Multi-domain Representation

We now transform preprocessed token sequences into representations suitable for detection. Word embedding techniques use real-valued vectors to represent each token in such a way that tokens with similar semantics are expected to be closer in the continuous vector space [44]. Compared to conventional representations like one-hot embedding, they produce lower-dimensional embeddings with semantic similarity and are thus widely applied in natural language processing. However, tokens from different domains will be separately projected into different vector spaces[1], which overlooks the semantic similarity between different domains, rendering the knowledge distilling from one domain incapable to improve detection on other domains.

With the expectation that similar tokens across all domains can have close representations, we propose multi-domain representation, where tokens from any domain will be represented as a weighted sum of a universal token set. We observe that, if token sets of two domains overlap on a token, this overlapping token generally has similar semantics in both domains. The following three cases can lead to semantics similarity in overlapping tokens. i) Domain developers follow the same development specifications, e.g., two domains share several overlapping tokens when using the same API. ii) Developers use common tokens to express certain concepts. For example, even without development specifications, "`version`" typically describes a set of numbers identifying an update. Such tokens rarely incur semantic overloading. iii) Our adaptive preprocessing merges similar values into the same predefined tokens that

[1]Note that combining all tokens from multiple domains in one set is capable of producing representations in the same vector spaces. However, such a method cannot be extended to new domains because it requires obtaining representations of all domains at the same time.

Table 1: Notations

Notation	Explanation
Q^m	The token set of the m_{th} domain
T^m	The token sequences of the m_{th} domain
T_i^m	The i_{th} token sequence of T^m
D^m	HTTP requests of the m_{th} domain

represent the same semantics across different domains. Our empirical study on real-world data also confirms this observation. For example, we manually investigate the overlapping tokens between the two domains in our experiment data and find that 120 of the 145 overlapping tokens share similar semantics in both domains.

Based on the observation above, intuitively, we select one base domain and then, all these overlapping tokens shared between the base domain and each other domain can serve as the references for aligning the tokens of them. Note that we leverage orthogonal transformations to align tokens (see Equation (1)) so that a few overlapping tokens that do not share the same semantics have negligible impacts on token alignment [57]. Then, the similarity between any pair of tokens can then be computed and used to represent each token using the weighted sum of the universal token set (i.e., the token set of the base domain). Below we describe the details. Important notations are summarized in Table 1.

Token Alignment. We first align the tokens in different domains according to their semantic similarities. Among all M domains, we select a domain u as our base domain and align the tokens in each other domain with tokens in the base domain. Given the token sequences, we first compute the preliminary representations $\{W^i\}_{i=1}^M$ for tokens Q^i in each domain i separately. Here we adopt Word2vec [44] which mines contextual information within token sequences by predicting missing tokens from a surrounding window. Then for each domain v, we find a orthogonal linear transformation to project the preliminary representation of v to W^u vector space,

$$W^{v\to u} = O_v \cdot W^v,\ s.t. O_v^T O_v = I, \tag{1}$$

where I denotes the identity matrix. In other words, O_v^T can project tokens in domain u back into W^v vector space of domain v. We find the transformation by minimizing the squared reconstruction error on the overlapping tokens X_{uv} as follows:

$$\min_{O_v} \sum_{x\in X_{uv}}^{p} \left\| W^u(x) - O_v \cdot W^v(x) \right\|^2,\ s.t. O_v^T O_v = I, \tag{2}$$

Note that, we can easily identify these overlapping tokens by finding the intersection of token sets, i.e., $X_{uv} = Q^u \cap Q^v$, without resorting to manual effort. Unfortunately, Equation (2) cannot be optimized by simply applying gradient descent methods because of the orthogonal constraint for O_v. Instead, we use the singular value decomposition based method [57] under the condition that W^u and W^v are normalized. Readers can refer to the original paper for more details.

Universal Embedding Mapping. We now map tokens from different domains to a universal embedding space. According to the Equation (1) which aligns the preliminary representations of domain v with those of the base domain u, the similarity between a token t_v in domain v and a token t_u in domain u can be computed following the equation below:

$$S(t_v, t_u) = W^u(t_v) \cdot A \cdot W^{v\to u}(t_u), \tag{3}$$

where A is a learnable parameter initialized as an identity matrix and is jointly trained with the seq2seq model. With the token similarity, each token can be represented as the weighted sum of the token set Q^u of the base domain u, named as *universal embedding*:

$$U^v(t_v) = \sum_{t_i\in Q^u} \frac{e^{S(t_v,t_i)}}{\sum_{t_j\in Q^u} e^{S(t_v,t_j)}} \cdot E(t_i) \tag{4}$$

where the multiplicator on the left is the weight calculated by normalizing the similarity using a softmax function, and E is a matrix whose shape is $\|Q^u\| \times d$. We initialize E randomly and train it jointly with the seq2seq model.

Besides, a domain-specific embedding term is integrated to support representing the tokens derived from URL paths, e.g., token "CGI1" from "/URL1/URL2/CGI1…". Note that, these tokens are domain-specific and have no similar context structure with other tokens, thus can only be "memorized" by the models. Finally, by summing up the universal embedding U^v in Equation (4) and the domain-specific embedding E^v, our *multi-domain representation* for domain v is computed:

$$\tilde{U}^v(x) = U^v(x) + E^v(x), \tag{5}$$

where E^v is a matrix whose shape is $\|Q^v\| \times d$. E^v is initialized as a zero matrix and is trained only to obtain the detection model for a specified target domain.

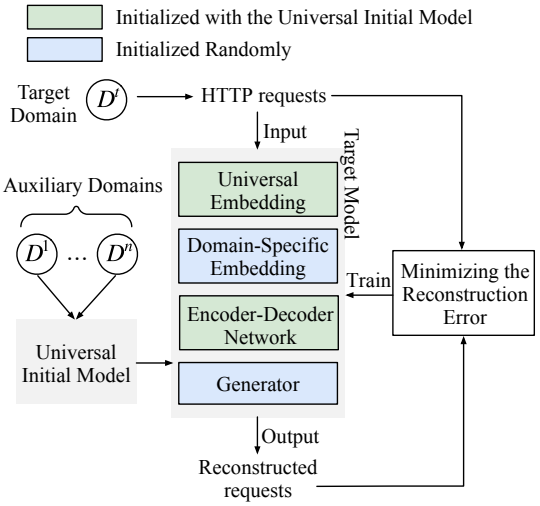


Figure 3: Training of the target model.

4.3 Cross-domain Training

Cross-domain training obtains a detection model for each target domain by combining the knowledge from both the target domain and the other auxiliary domains. It utilizes meta-learning to better exploit the knowledge from auxiliary domains. Specifically, we first build a universal initial model using training data from multiple auxiliary domains, with the expectation that it exploits several

Algorithm 1 Training the universal initial model

Input: Model f_θ; token sequences of M domains $T_M = \{T^1, T^2, \ldots, T^M\}$; learning rate α; step size s.
Output: Parameters θ^* of universal initial model.

1: Randomly initialize θ;
2: **while** f_θ is not convergent **do**
3: $\theta_{temp} \leftarrow \theta$;
4: $T^k \leftarrow$ Sample a domain k from T_M;
5: **for** $j = 1, 2, ..., s$ **do**
6: $b^k \leftarrow$ Sample a batch data from T^k;
7: Update θ according to Eq. (8) with b^k;
8: **end for**
9: $U_{T^k}(\theta) \leftarrow \theta$
10: $b^k \leftarrow$ Sample a batch data from T^k;
11: Compute g_{meta} according to Eq. (10) with b^k and $U_{T^k}(\theta)$;
12: $\theta \leftarrow \theta_{temp}$;
13: update θ according to Eq. (11) with g_{meta};
14: **end while**
15: $\theta^* \leftarrow \theta$;
16: **return** θ^*

auxiliary domains, yet provides better performance on any novel domain without resorting to a large amount of training data. To obtain the detection model for a specified target domain, the universal initial model is then adapted using the limited training data from that domain.

Universal Initial Model. Using the training data from multiple domains, we produce a model with parameters that can be adapted well to new domains by leveraging the idea of meta-learning [18]. Note that, though the meta-learning algorithm [18] is designed for supervised learning, it can be also applied to our unsupervised task that uses unlabeled data. This is because the objective of our task is to identify attacks by reconstructing each request, which is a supervisory signal. Formally, the loss $\mathcal{L}_{T^i}$ on training data T_i from domain i is defined as:

$$\mathcal{L}_{T^i}(\theta) = \sum_{T^i_j \in T^i} R(T^i_j, f_\theta(T^i_j)), \tag{6}$$

where f_θ is our model with the learnable parameters θ, T^i_j is the j-th token sequence of the domain i, and $R(\cdot)$ is the negative log-likelihood function, i.e., reconstruction error. Note that f_θ is composed of the multi-domain representation and the seq2seq model, and these two parts are optimized jointly. The details of the multi-domain representation are described in Section 4.2, and the seq2seq model, which consists of the encoder-decoder network and the generator, uses the same design as ZeroWall [60].

Given training data $T_M = \{T^1, T^2, \ldots, T^M\}$ from M domains, our objective function, denoted as the meta objective, aims to find the following parameters θ^*:

$$\theta^* = \underset{\theta}{argmin} \sum_{T^i \in T_M} \mathcal{L}_{T^i}(U_{T^i}(\theta)), \tag{7}$$

where $U_{T^i}(\cdot)$ is the parameter update operator using gradient descents on the training data of i-th domain. Equation (8) shows a step in the gradient descents:

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}_{T^i}(\theta), \tag{8}$$

where α is a hyperparameter representing the learning rate. $U_{T^i}(\theta)$ may contain several steps in Equation (8), but below we let $U_{T^i}(\theta)$ contain only one step for the simplicity of discussion, i.e., $U_{T^i}(\theta) = \theta'$. Intuitively, the meta objective ensures that a model initialized with θ^* can be trained (using gradient descents) to achieve the minimized loss under each domain.

We leverage gradient descents to solve this meta objective and denote the gradient for optimizing it as the meta gradient g_{meta}:

$$\begin{aligned} g_{meta} &= \sum_{T^i} \nabla_\theta \mathcal{L}_{T^i}(U_{T^i}(\theta)) \\ &= \sum_{T^i} \nabla_{U_{T^i}(\theta)} \mathcal{L}_{T^i}(U_{T^i}(\theta)) \nabla_\theta U_{T^i}(\theta) \\ &= \sum_{T^i} \nabla_{U_{T^i}(\theta)} \mathcal{L}_{T^i}(U_{T^i}(\theta)) \nabla_\theta (\theta - \alpha \nabla_\theta \mathcal{L}_{T^i}(\theta)) \\ &= \sum_{T^i} \nabla_{U_{T^i}(\theta)} \mathcal{L}_{T^i}(U_{T^i}(\theta)) (I - \alpha \nabla_\theta \nabla_\theta \mathcal{L}_{T^i}(\theta)). \end{aligned} \tag{9}$$

Since the high-order derivatives in g_{meta} are expensive to compute, we apply the first-order approximation [48] to the meta gradient, yielding the following approximation:

$$g_{meta} \approx \sum_{T^i} \nabla_{U_{T^i}(\theta)} \mathcal{L}_{T^i}(U_{T^i}(\theta)). \tag{10}$$

After obtaining g_{meta}, a naive way to optimize Equation (7) is to perform gradient descents iteratively according to g_{meta}. However, as shown in Figure 3, in our model the domain-specific embedding and the generator that creates the transformation between tokens and the vector space needs to be reinitialized for processing new domains. Therefore, the learned knowledge represented in these parameters cannot be retained [71] when training on new domains.

To address this issue, we do not update the parameters of domain-specific embedding and generator when applying g_{meta}, which enables us to "embed" the knowledge to the parameters that can be retained. Formally, the universal initial model is updated as follows:

$$\theta \leftarrow \theta - \alpha \cdot g_{meta} \odot \boldsymbol{K}, \tag{11}$$

where $\odot$ represents the element-wise multiplication and $\boldsymbol{K}$ is a parameter mask whose values are 0 for the parameters of the domain-specific embedding and the generator, and 1 otherwise.

Algorithm 1 shows the pseudocode of training the universal initial model. We first randomly initialize θ (line 1). Then we apply a two-loop strategy, consisting of an inner and an outer loop, to iteratively update θ. In lines 4-9, the inner loop updates θ using a randomly selected domain k and obtain $U_{T^i}(\theta)$, which simulates model training on any given target domain. In lines 10-13, the outer loop updates θ by computing g_{meta}. The training data for both the inner loop and outer loop are sampled independently from the randomly chosen domain. It is worth noting that, unlike the outer loop which does not update domain-specific parameters, all parameters are still eligible for updates in the inner loop.

Target Model. We take the universal initial model as the starting point to obtain a detection model for the target domain (denoted as the target model). Note that the structure of the target model is the same as that of the universal initial model. Specifically, the target model inherits the parameters of universal embedding and the encoder-decoder network from the universal initial model, and

reinitializes the domain-specific embedding and generator. Given limited data from the target domain, all learnable parameters of the target model are jointly optimized by minimizing the loss function defined in Equation (6).

5 EVALUATION

5.1 Settings

Datasets. As shown in Table 2, we collect four datasets from different domains provided by a world-leading Internet company: 1) OV is an online video platform for streaming media; 2) SNS provides social networking service; 3) SF is a sports forum; and 4) IdM offers an identity management service. All four domains serve different applications, thus providing good diversity for our evaluation across heterogeneous domains. Specifically, we collect the HTTP requests allowed by the company's signature-based WAF for two consecutive days for each domain. The data from the first and second days are used for training and testing, respectively. To collect limited training data, we select consecutive 5-minute data starting from the same time from the total (1-day) training dataset of each domain. For ease of data collection, we do not consider the intervals between the data collection for the auxiliary domains and the target domain. However, such intervals do not impact the model performance, because the semantic correlations between different domains are derived from the same development specifications that do not change over time. We confirm this in real-world deployment experiments (see Section 6).

Ground truth. The security operators collect the ground truth of attacks in the testing set using the following two approaches. First, they manually perform Web log analysis by investigating specific predefined keywords generated according to prior detected events. Second, they examine all requests detected by RETSINA and the baselines. A request is identified as an attack if it i) includes malicious payloads, e.g., a code snippet used for injection, or ii) does not follow the normal user behavior of Web applications, e.g., attempting to access administration interfaces. The rest of the requests are considered benign. Notably, the mislabeled attacks, if any, in our ground truth may lead to a higher reported recall, but they will not impact the fairness of comparisons with baselines.

Metrics. We evaluate the detection performance using precision (Pre), recall (Rec), and F1-score (F1), which are also widely used in previous work since zero-day Web attacks only comprise a very small proportion of all HTTP requests.

Baselines. We choose the two unsupervised anomaly detection methods ZeroWall and MTL. We also select the two supervised techniques SCNN and SRNN, to evaluate whether using some known attack requests yields a better detection system.

- **ZeroWall.** ZeroWall is the state-of-the-art unsupervised zero-day Web attack detection method. It parses requests according to punctuation and leverages a seq2seq model to achieve end-to-end detection [60]. We also test several widely-used unsupervised zero-day Web attack detection methods. For example, SAE [63] uses n-gram for tokenization, then leverages the stacked auto-encoder and the isolation forest for feature extraction and detection. However, these methods perform very poorly on our real-world datasets, and the F1-scores are almost 0. Similar results have also been reported in previous work [60].

Table 2: Statistics on 4 domains.

Domain	# of Requests	# of Attacks
Online Video (OV)	57.71M	392
Social Network Service (SNS)	84.72M	818
Sports Forum (SF)	13.85M	2,623
Identity Management (IdM)	136.78M	59

- **MTL.** Neither ZeroWall nor any existing methods for detecting Web attacks can utilize data from auxiliary domains. To enable more fair comparisons, we include a knowledge-sharing-based method for multi-task security problems. In particular, we modify the state-of-the-art framework in [69] and designate the modified method as MTL. MTL fully utilizes data from both the auxiliary and the target domains, thus learning similar knowledge to RETSINA. Specifically, it mixes data from multiple auxiliary domains and the target domain to train a model on the target domain. To prevent the effect of task imbalance (e.g., the target domain has fewer requests), we assign weights to the loss of requests from each domain in accordance with the number of requests from with limited data.
- **SCNN.** SCNN leverages a specially designed CNN to classify HTTP requests [74]. For better performance, we use the preprocessing technique of ZeroWall to produce token sequence. Since supervised methods require labeled data, we collect attack requests filtered by WAFs from a day for each domain and use them as malicious samples, i.e., their training data contains 1-day known attack requests and benign modifyrequests collected at different lengths of time.
- **SRNN.** SRNN utilizes the attention-based LSTM to classify sequence data [81]. The settings for preparing training data and preprocessing are the same as SCNN.

For unsupervised methods RETSINA, ZeroWall, and MTL, to make a fair comparison, we choose LSTM as the backbone of encoder-decoder networks following [60]. LSTM has been demonstrated to be more effective at modeling sequence data than GRU or classic RNNs [11, 17]. Moreover, Transformers may learn an identity function that reconstructs HTTP requests including attacks, and thus are ill-suited for the detection methods that identify attacks based on reconstruction errors [47, 64]. Our empirical study shows that, if we replace LSTM with Transformer, the F1-score of RETSINA significantly drops from 0.913 to 0.487 in domain OV with 5-minute training data.

Selection of the Base Domain. We select a domain among the auxiliary domains as the base domain if the domain has the most overlapping tokens with other domains. It allows us to maximize the overlap for token alignment and thus best utilizes the correlation between different domains for detection. We also conduct an empirical study in Appendix B and show that the selection of the base domain has limited impacts on the model performance.

Implementation. All the evaluations are conducted on a GPU server whose hardware environment is configured as 10-core vCPU, NVIDIA Tesla T10 GPU, and 40GB memory. Deep learning models are implemented in PyTorch 1.7.1 with CUDA 11.0 toolkit. We use Gensim to implement Word2vec.

Table 3: Overall performance comparisons when the collection time of training data is 5 minutes and 1 day respectively.

Time	Method	OV			SNS			SF			IdM		
		Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
5 min	ZeroWall	0.580	0.918	0.711	0.168	0.536	0.256	0.422	0.964	0.587	0.647	0.932	0.764
	MTL	0.777	0.623	0.692	0.268	0.833	0.406	0.498	0.955	0.655	0.845	0.831	0.838
	SCNN	0.080	0.940	0.147	0.174	0.845	0.288	0.084	0.699	0.150	0.377	0.966	0.543
	SRNN	0.040	0.338	0.072	0.070	0.972	0.131	0.073	0.717	0.133	0.177	0.983	0.300
	Ours	0.901	0.926	0.913	0.667	0.911	0.770	0.685	0.972	0.802	0.868	1.000	0.929
1 day	ZeroWall	0.924	0.866	0.894	0.810	0.599	0.719	0.802	0.830	0.816	0.879	0.864	0.872
	MTL	0.935	0.861	0.896	0.902	0.513	0.654	0.486	0.797	0.603	0.962	0.847	0.901
	SCNN	0.954	0.395	0.559	0.678	0.530	0.595	0.936	0.393	0.553	0.851	0.678	0.755
	SRNN	0.960	0.662	0.784	0.324	0.185	0.236	0.529	0.330	0.407	0.897	0.593	0.714
	Ours	0.921	0.884	0.902	0.895	0.819	0.855	0.922	0.893	0.907	1.000	0.915	0.956

Configurations. When evaluating the performance of one domain, we use the other three domains as auxiliary domains to perform MTL or cross-domain training of RETSINA. Note that, since the universal initial models only need to be trained once, they do not suffer from limited data, thus we train them using full datasets from auxiliary domains. For our method, we use the skip-gram to train the Word2vec embeddings. For the LSTM-based seq2seq model adopted in RETSINA, ZeroWall, and MTL, both the hidden size and embedding size are set to be 512. We use the Adam [32] with the learning rate of 0.001 as the optimizer and the learning rate decay is applied. To make an efficient computation on GPU, we use token-level dynamic batching to train the model. The token batch size is set to 4096.

Ethical Considerations. The data that we analyze only contains URLs and bodies of requests that are helpful for detection. Any fields containing sensitive user information have been removed. We also do not use the collected data to identify any individual. All datasets are stored on the company's servers and are accessed through an internship program. We conduct experiments in an isolated environment that has no impact on the production environment.

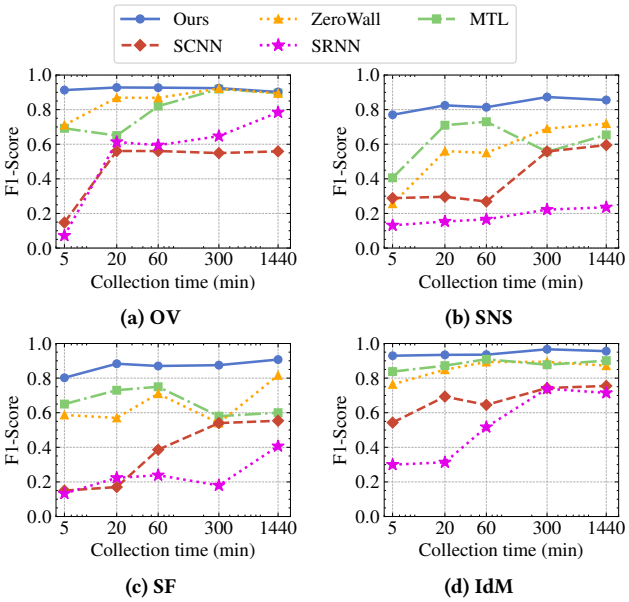


Figure 4: Detection performance with respect to the amount of training data.

5.2 Detection performance

First, we run experiments to evaluate whether our method outperforms baselines with limited training data. We use the 5-minute data for model training and test the trained model on the testing data. Table 3 shows the detection performance of our method and baselines. It can be seen that our method significantly outperforms baselines when the collection time is 5 minutes. Compared with ZeroWall, our method improves the F1-score by 21.5% at least. The results confirm the advantages of leveraging knowledge sharing when the training data is limited. We can further observe that such an improvement is mainly derived from precision (e.g., improved by 55.3% on OV), which indicates that our method effectively mitigates the false positives caused by the limited training data. Moreover, MTL can slightly improve model performance when the collection time is 5 minutes. For example, the F1-scores of SNS using MTL are improved from 0.256 to 0.406 compared with ZeroWall. However, this improvement is still far from ours. Overall, our method is more effective with limited Web training data.

Second, our method is also better than baselines when the collection time is 1 day. For example, the F1-scores of SNS and IdM are improved by 18.9% and 9.6% respectively compared with ZeroWall. This indicates that even with abundant training data, our method still helps to improve the generalizability of the trained model by combining the knowledge from other domains. Note that the results of MTL are even lower than that of ZeroWall when the collection time is 1 day. We analyze the main reason is that one or more auxiliary domains may have a greater impact than the target domain, which degrades the detection performance on the target domain.

Third, supervised methods SCNN and SRNN perform the worst when compared to unsupervised methods. For the 5-minute evaluation, both methods are ineffective due to insufficient data. For the 1-day evaluation, their recall is poor compared to their precision, indicating many attacks are undetected. This is because supervised methods depend on prior knowledge of attack requests and thus

Table 4: A benign request that is incorrectly recognized by baseline as an attack but is correctly recognized by RETSINA.

Request	.../send?xxxx_code=8**&bids=1*****76 &yyyyyy_code=18***6
Token Sequence	... send **xxxx_code _num_ bids _num_ yyyyyy_code _num_**

Table 5: Token sequence of two example requests that have a similar structure to that shown in Table 4. Example 1 and 2 are from OV and SNS respectively.

Example 1	watch_record_new callback _pbas_ **pagesize _num_ g_tk _num_ g_vstk _num_**
Example 2	cgi_qzshare **uin _num_ spaceuin _num_ g_iframe _num_**

fail to effectively detect zero-day attacks, which were previously unknown. Conversely, RETSINA is capable of detecting zero-day attacks because it detects attacks only according to learned benign patterns of Web requests.

Fourth, the detection performance varies across domains. For example, RETSINA performs worse on SNS and SF than on OV and IdM. This is mainly because the services running in different domains invoke different APIs with various parameters, and thus the complexity of benign patterns of these services varies. It is difficult for the detection model to learn these complicated patterns, which may result in more false positives and lower precision.

Figure 4 shows the F1-scores given different collection times of the training data. We observe that our method only needs 5-minute training data to reach the performance that the baseline method uses 1-day training data. Moreover, for all methods, model performance increases with collection time in most cases. If the collection time is shorter, the performance gap between the baseline method and our method will be larger. However, it is worth noting that more collection time does not always lead to better performance. For example, the F1-score of 1-day training data is slightly lower than that of using 5-minute training data on OV. This performance loss comes from recall (reduced from 0.926 to 0.884) according to the detailed breakup of performance shown in Table 3, which is mainly because the training data includes some noise samples that reduce the ability of the unsupervised model to detect zero-day attacks. We also discuss this phenomenon in the subsequent section.

We further perform a case study to investigate how our method helps to improve detection performance. We show a benign request that is misjudged by ZeroWall but correctly recognized by our method in IdM (note that some fields are masked for preserving anonymity and privacy) in Table 4. ZeroWall produces the false positive because the pattern of the token sequence "`xxxx_code _num_ bids _num_ yyyyy_code _num_`" is not included in the training data. However, requests with a similar pattern may exist in other domains (Table 5 shows two examples in OV and SNS, respectively), which allows our method to better learn the pattern of these requests and avoid the false positive.

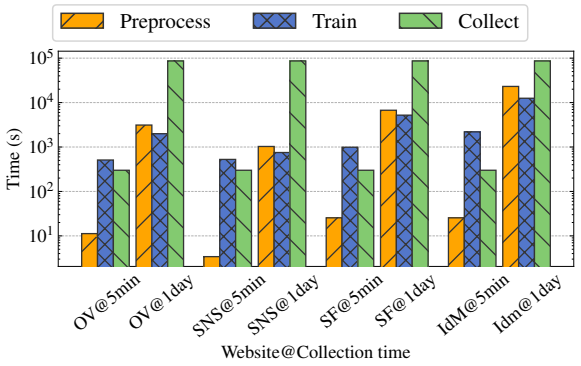


Figure 5: Development times of RETSINA with 5-minute training data and ZeroWall with 1-day training data.

5.3 Development Overhead

We now assess the development overhead of our proposed method. Considering that our proposed method with 5-minute training data achieves comparable detection performance to ZeroWall using 1-day training data, we compare the development times of these two settings. Figure 5 displays the comparison results in terms of collection time, preprocessing time, and training time, where training time refers to the time to train a convergent model. It can be seen that RETSINA is hundreds of times faster than ZeroWall for both collection and preprocessing. Meanwhile, the training time of RETSINA is only half that of ZeroWall. These results suggest that RETSINA is more efficient in developing new detection models than ZeroWall, which requires a large amount of training data.

We also examine the training time for universal initial models with the three auxiliary domains. Training universal initial models of OV, SNS, SP, and IdM takes 437, 487, 459, and 333 minutes, respectively. Note that universal initial models can be prepared in advance, thus not affecting the efficiency of developing new detection models.

5.4 Ablation Study

To further validate the design of RETSINA, we evaluate how different modules contribute to improving detection performance. First, we assess how adaptive preprocessing supports semantics analysis. We replace the adaptive preprocessing with the naive tokenization technique commonly used in previous work [60], denoting *ours without adaptive preprocessing*. Table 6 shows the comparison of detection performance. It can be seen that adaptive preprocessing improves the F1-scores under all settings. The improvements are significant on SNS, SF, and IdM when the collection times are 5 minutes and 1 day. For example, without adaptive preprocessing, the F1-score on SNS drops by 26.3%, and its corresponding precision and recall drop by 31.3% and 18.2%, respectively. Note that, the improvements brought by adaptive preprocessing in OV are minor since the requests in OV are relatively simple and can be mostly handled by naive tokenization techniques.

Second, we remove the multi-domain representation module and the cross-domain module. Note that, we treat the multi-domain representation module and the cross-domain training module as being inseparable in knowledge sharing based on meta-learning because

Table 6: Comparison with and without our adaptive preprocessing. A.P. is the abbreviation of adaptive preprocessing.

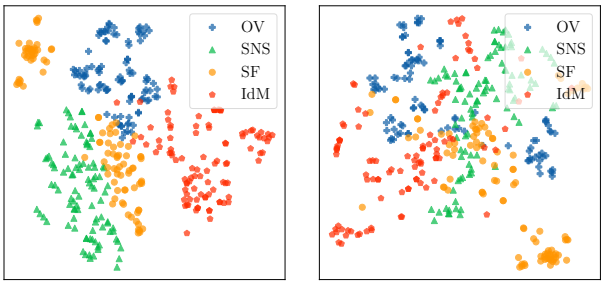
Time	Method	OV			SNS			SF			IdM		
		Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
5 min	Ours w/o A.P.	0.908	0.903	0.905	0.458	0.745	0.567	0.525	0.992	0.686	0.825	0.994	0.902
	Ours	0.901	0.926	0.913	0.667	0.911	0.770	0.685	0.972	0.802	0.868	1.000	0.929
1 day	Ours w/o A.P.	0.919	0.876	0.897	0.745	0.714	0.729	0.763	0.893	0.823	1.000	0.847	0.917
	Ours	0.921	0.884	0.902	0.895	0.819	0.855	0.922	0.893	0.907	1.000	0.915	0.956

Table 7: Comparison of different variants of our method with knowledge sharing. K.S. and T.L. are the abbreviations for knowledge sharing and transfer learning, respectively.

Time	Method	OV			SNS			SF			IdM		
		Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
5 min	Ours w/o K.S.	0.633	0.943	0.758	0.319	0.774	0.452	0.470	0.989	0.637	0.883	0.898	0.891
	Ours w/ T.L.	0.431	0.926	0.588	0.389	0.806	0.525	0.456	1.000	0.626	0.850	0.864	0.857
	Ours	0.901	0.926	0.913	0.667	0.911	0.770	0.685	0.972	0.802	0.868	1.000	0.929
1 day	Ours w/o K.S.	0.914	0.882	0.898	0.893	0.689	0.778	0.947	0.824	0.881	0.943	0.847	0.893
	Ours w/ T.L.	0.899	0.876	0.887	0.844	0.620	0.715	0.975	0.696	0.812	0.981	0.898	0.938
	Ours	0.921	0.884	0.902	0.895	0.819	0.855	0.922	0.893	0.907	1.000	0.915	0.956

1) multi-domain representation performs universal embedding mapping without changing the relationship between embeddings of tokens in the same domain, thus training a model using universal embedding from only one domain brings no benefit, and 2) cross-domain training requires inputs to be in the same feature space, which cannot be guaranteed without universal embedding. Therefore, we consider two variants for these two modules: 1) *ours without knowledge sharing*. We use the token sequences generated by the adaptive preprocessing module to train a seq2seq detection model and 2) *ours with transfer learning*. We leverage the common transfer learning paradigm as an alternative choice for knowledge sharing. A model is pre-trained using data from multiple auxiliary domains and is then fine-tuned to the target domain, where the embeddings of unknown tokens are randomly initialized. Table 7 shows the results. Under all settings, our method achieves the best performance with the multi-domain representation and the cross-domain training. Moreover, these two modules are especially helpful to boost detection performance when the data is limited. When these two modules are removed, F1-scores drop at most 41.3% and 9.0% with 5-minute and 1-day training data, respectively. Besides, the performance of transfer learning is even worse than without knowledge sharing in most cases, indicating that transfer learning is not effective for sharing knowledge among heterogeneous Web domains.

We further visualize the distribution of token embeddings in different domains to illustrate how our multi-domain representation helps to achieve knowledge sharing from heterogeneous Web data. We randomly select 100 tokens in each domain and project their embeddings into 2-dimension using t-SNE [62]. The embedding of t-SNE is initialized by principal component analysis, which is more globally stable than random initialization [5]. Figure 6 shows the results using our multi-domain representation and the Word2vec technique. Using Word2vec, embeddings from each domain form a cluster, which indicates the tokens of each domain are in different



(a) Word2vec **(b) Ours**

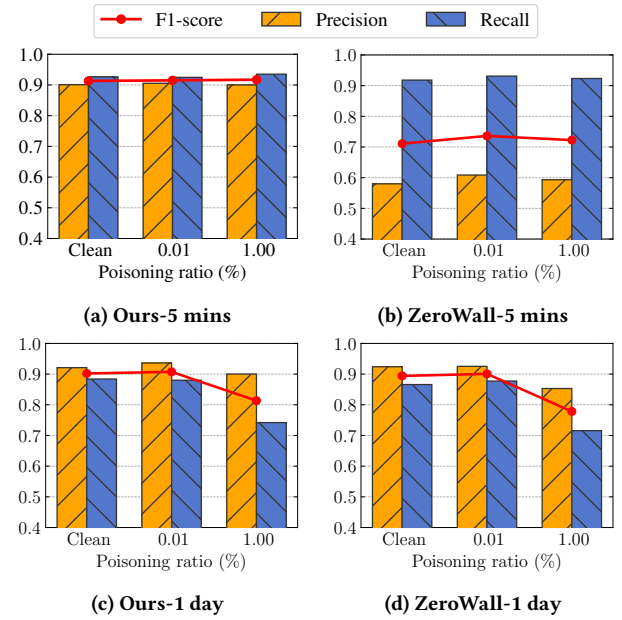
Figure 6: T-SNE Visualization of token embeddings using Word2vec and our method.

feature spaces. On the contrary, with multi-domain representation, tokens from the same domain are more dispersed and tokens from different domains are mixed. This suggests that multi-domain representation makes tokens from different domains share the same feature space, and thus the correlation between tokens is captured.

5.5 Robustness under Data Poisoning

In this section, we quantify how effective our method is against data poisoning. We realize data poisoning by generating and adding poison samples to the original dataset for model training. Specifically, the poison samples are automatically generated by a commercial Web security scanner[2]. The poison samples are added to the original dataset at different ratios, denoted as poison ratios η. We vary η over [0%, 0.1%, 1%]. We test both our method and ZeroWall using

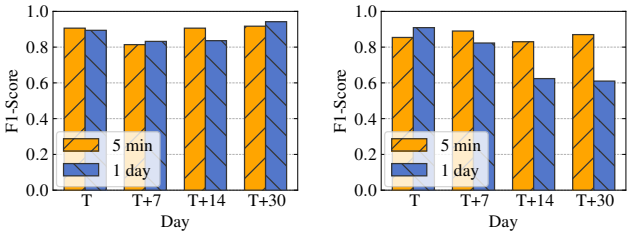
[2]The scanner presets over 3000 human-written attack instances which cover more than 40 types of common Web vulnerabilities including remote command execution, SQL injection, etc. To generate poison samples, it first collects a set of normal requests and substitutes query values in the HTTP request based on these instances.



(a) Ours-5 mins (b) ZeroWall-5 mins

(c) Ours-1 day (d) ZeroWall-1 day

Figure 7: Comparison of detection performance under data poisoning.



(a) OV (b) SNS

Figure 8: Performance of our method in online deployment when the training data is collected taking 5 minutes and 1 day respectively.

different collection time and present the comparison of detection performance on OV in Figure 7.

Overall, our method outperforms ZeroWall when dealing with data poisoning and is more stable. When the collection time is 5 minutes, both methods are stable when the poison ratio increases, but our method provides better detection performance. When the collection time is 1 day, the detection performance of both methods degrades if we increase the poison ratio to 1%. We can see that recall drops significantly, compared to precision. This is mainly because the model incorrectly learns the patterns of poisoning samples and identifies malicious samples with similar patterns as benign ones.

Comparing the results of 5-minute and 1-day evaluations, we surprisingly find that, under the same poisoning ratio, models trained with limited data are more robust against data poisoning. This is a result of the fact that, given the same poison ratio, the number of poison samples in 5-minute data is smaller than in 1-day data. To be more specific, when the number of poison samples is small, these samples behave like separate noises and are easily ignored by deep learning models [54, 68]. As the number of poison samples increases, it is more probable that these samples will cluster in the feature space and produce a larger norm of gradient that will be captured by the gradient descent algorithm [7].

6 REAL-WORLD DEPLOYMENT

To investigate how effective RETSINA is in a real production environment, we deployed it on the domains of OV and SNS in the company. The real-world deployment demonstrates the superiority of RETSINA in terms of effectiveness and robustness. Below, we first describe our deployment experience. Then, we present the results of the detection performance and our discoveries.

6.1 Real-World Deployment Experience

Since WAFs are effective in detecting known attacks, RETSINA works complementarily with WAFs rather than completely replacing them, to solely concentrate on zero-day attacks. We use the producer-consumer approach to detect online activity. In particular, the WAF acts as the producer, with the detection station, which deploys all of the detection models from each domain, acting as the consumer. Every request permitted by WAF will be mirrored to the detection station for detection. The detection result will be delivered to security operators for additional analysis.

Due to its large volume, real-world traffic is hard for deep learning models to process in a timely manner. We thus leverage the hash technique to avoid processing the same token sequence repeatedly. Since requests with the same structure but different valid values are preprocessed into the same token sequence, a hash table is built to store the hash values of token sequences and their detection results for all token sequences that have been processed beforehand. For example, the detection model only needs to process one of "`/path/?cid=abcde`", "`/path/?cid=bcdef`", "`/path/?cid=cdefg`" to return detection results for all three requests since they have the same token sequence "`path cid _cid_`'. After the hash tables in the preprocessing significantly reduce the number of token sequences needed to be predicted by the model, the preprocessing itself is the bottleneck for the system throughput. Therefore, we suggest assigning the majority of the CPU resources to the preprocessing concurrently. Note that, such a system has good scalability. It can be easily extended by adding multiple backend servers responsible for preprocessing or model prediction in a distributed way.

In practice, the universal initial models should be trained before target domain training. Therefore, we use the universal initial models developed in Section 5 that was trained more than 6 months ahead of our real-world deployment to train the target models. The experiment results in Section 6.2 show that the universal initial models are effective for detection, which confirms that the intervals between the data collection for the auxiliary domains and the target domain do not impact the model performance.

6.2 Real-World Detection Performance

Detection Results. In the deployment, we collect 5-minute data on the 0th, 7th, 14th, and 30th days respectively to train the target models. For OV and SNS, we detect 126 and 218 zero-day attack requests on average per day, respectively. We also compare the

Table 8: Examples of zero-day attacks in OV and SNS.

Domain	Example
OV	/***/***?_=166******782&callback=$%7B@var_dump(md5(158477632))%7D&channdlId=0
SNS	/***/***?desc=***&md=1&origin=***&pics=***&qc=$%7B@var_dump(md5(273657368))%7D;&showcount=0&site=&summary=***&url=https://***.com/***&where=10

models that are trained using 5-minute or 1-day data, respectively, in Figure 8. It can be seen that our method can achieve high detection performance using only 5-minute training data, and the F1-scores of 5-minute and 1-day training data are comparable in most cases. The results confirm the effectiveness of our method under limited training data in a real-world environment.

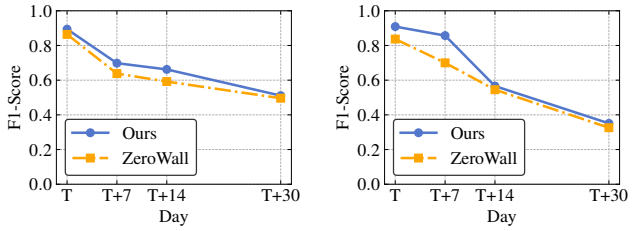
Interestingly, we observe that the results of 1-day experiments can be even worse than those of 5-minute experiments on the 14th and 30th days of SNS. We carefully examine the corresponding results and find that, the performance degrades since the training data is poisoned with many attack requests that the WAF is unable to identify. Conversely, the results of the 5-minute experiment behave normally, which is also consistent with the phenomenon observed in Section 5.5.

We also present two examples of zero-day attacks that are detected by our method but missed by WAF in Table 8. Though detected from OV and SNS, respectively, both attacks belong to the same attack type—command injection—where attackers aim to force the server to execute arbitrary codes. Benefiting from unsupervised learning, our method detects these attacks based on the learned patterns of benign requests. Conversely, WAFs usually detect command injection attacks based on keyword matching. It is unable to detect these attacks since it does not cover the keyword "`$%7B@var_dump`".

Detection under Concept Drift. Web applications are frequently updated, as was earlier mentioned. After the update, concept drift—the phenomenon that the detection performance of a trained model degrades when presented with previously unseen data due to the data update—occurs. We now explore the effectiveness of our approach against concept drift.

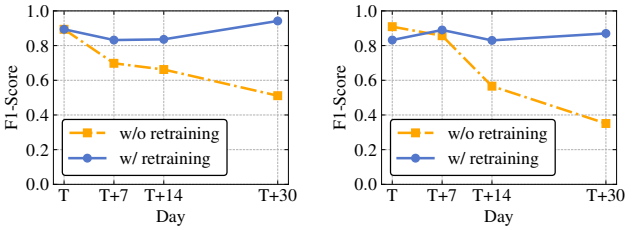
Firstly, we examine how RETSINA and the baseline perform against concept drift. We train models using the whole data from the 0th day and use them for subsequent testing without retraining. Figure 9 shows the comparison results. Overall, the detection performance of both our method and ZeroWall degrades. However, our method is more robust to concept drift and always performs better than ZeroWall. For example, for SNS, the F1-score drops by 0.052 in 7 days after training. In comparison, the decrease of ZeroWall is 0.137. This demonstrates that knowledge sharing enables our model to have better generalizability against concept drift. We also examine the corresponding results and find that concepts drift occurs when the Web services are updated, e.g., new APIs are added. Requests that invoke these new APIs are unknown to the old model, resulting in false positives.

Secondly, we investigate whether periodic retraining with 5-minute data provides better detection performance than without



(a) OV (b) SNS

Figure 9: Performance comparisons between our method and ZeroWall against concept drift.



(a) OV (b) SNS

Figure 10: The impact of retraining on detection performance.

retraining (i.e., using the model trained on the 0th day). Figure 10 shows the comparison results. In a word, the results of detection with retraining are stable over time and better than the results without retraining, which validates the necessity of updating the model for preserving the detection performance over time. On the contrary, detection performance degrades severely over time without retraining. For OV, the degradation is significant from the 7th day. For SNS, although the performance drop appears to be relatively small on the 7th day, the model performance still suffers a huge drop on the 14th day. Thus, for these two domains, it is necessary to update the model at least once a week.

7 DISCUSSION

Security against Adaptive Attacks. Given that RETSINA is a deep learning-based, adversaries may leverage two common techniques for adaptive attacks against deep learning models, i.e., adversarial examples [72] and data poisoning [46], to bypass our detection. We argue that these two techniques are less practical for our framework. For adversarial examples, existing works mainly study their impact on supervised classification tasks in computer vision [9] and natural language processing [34]. As discussed in earlier work, their extension to unsupervised tasks only performs with clustering methods [10, 70] or vanilla auto-encoders [20, 21]. It is still challenging to construct adversarial examples for HTTP requests in practice. First, it is impractical to construct adversarial samples in the input space of our detection model, which is discrete, since HTTP requests are treated as sequential data in RETSINA. The earlier research on inverse feature-mapping problem shows that attacks are still impractical even when an adversary has white box access to the model and constructs adversarial examples in the

feature space through optimization. Second, it is hard to generate adversarial samples while preserving the malicious semantics, and no existing constraint functions measure the similarity of HTTP requests.

Data poisoning requires a certain number of poisoned requests that can bypass the existing WAF, which has a cost. Generally, the number of malicious requests unfiltered by WAFs is relatively small and thus these requests have less impact on the unsupervised model that learns dominated patterns in training data. Actually, the datasets we evaluate are also directly collected from WAFs and are not completely clean, but as shown in Table 3 our method still achieves high detection performance across all settings. This validates that our method is not vulnerable to the small number of attack requests unfiltered by WAFs. Moreover, in section 5.5, we show that the model is more robust to poisoning when the number of training data is less given the fixed poison ratio. Recall that learning with limited training data is exactly the strength of our method. We also show that when the collection time of training data is 5 minutes our method will not be affected even if the poisoning ratio is 1%. All these results demonstrate the robustness of our method against data poisoning. We acknowledge that adversaries may launch more powerful attacks, e.g., [27, 55], where only a small number of carefully crafted samples can drastically degrade the model performance. However, to the best of our knowledge, no related work has been proposed for the system of Web attack detection. We leave it to future work.

Limitations. RETSINA has the following limitations. First, our method can only judge whether a request is an attack, but cannot provide finer-grained detection results. Thus, it still requires security operators to review the reported attacks. In future work, we plan to incorporate with clustering algorithm and perform fine-grained detection by automatically correlating the attack requests that have similar semantics. Second, we detect attacks by inspecting the payload of a single HTTP request, which cannot cover the context-based attacks, such as challenge collapsar (CC) attacks [42]. In future work, contextual information can be incorporated into our system to detect these attacks.

8 RELATED WORK

Web Attack Detection. Deploying WAFs [1, 3, 4, 53] is the commonly used way to detect Web attacks in the industry. However, these methods are rule-based (i.e., non-ML based) and cannot detect zero-day attacks that are not matched by rules. Recently, machine learning based methods have been proposed to improve detection performance. These methods work in a supervised or unsupervised manner. Supervised methods [15, 19, 38, 40, 41, 66, 76] let the classifier learn to discriminate malicious requests based on previously labeled data. However, due to the huge amount of Web traffic and the rarity of attacks, it is not practical to manually label Web data. In addition, supervised methods assume the knowledge about attacks during training, which can produce unreliable predictions when faced with zero-day attacks [25, 35].

Unlike supervised methods which learn from both benign requests and malicious requests, unsupervised methods only learn the profile of benign requests, and requests which deviate from the profile are identified as being malicious. Several methods explore the probability of using the variants of auto-encoder [49, 60, 63]. Vartouni *et al.* use character-level N-gram and apply stacked auto-encoder [63]. However, the classification model (i.e., isolation forest) directly uses the output of the encoder which is insufficient for request representation, thus providing limited performance. Park *et al.* propose to use a convolutional auto-encoder with character-level binary image transformation [49]. Tang *et al.* formulate the detection problem as a self-translation problem and apply recurrent seq2seq neural networks [60]. Our proposed solution also falls into the category of unsupervised methods and is designed to complement existing WAFs to detect zero-day attacks. However, in contrast to all methods above, our proposed detection model only requires limited training data and has good generalizability utilizing the knowledge sharing.

Machine Learning in Multi-task Scenario. Recently, a series of approaches have been proposed for multi-task learning in different fields [26, 45, 59, 69, 75]. A common goal of these researches is to learn a unified model to solve a collection of related tasks at the same time. Their key intuition is that learning one task can help improve the performance of other tasks. However, multi-task learning is not a good paradigm to solve our Web attack detection problem because Web applications may be updated frequently. More specifically, we must retrain the unified model when a domain is updated or a new domain comes, which is inefficient compared to updating the model for a single domain name. Moreover, we have compared our method with multi-task learning in our evaluations and demonstrated our advantages.

Machine Learning with Limited Training Data. Many works focus on improving the performance of machine learning when using limited training data [30, 31, 58, 61]. These methods can be grouped into two categories, i.e., data augmentation based and knowledge sharing based. Traditional augmentation methods are developed based on domain-specific expertise [33, 56, 67, 80]. Some new works design algorithms to analyze the feature distribution of available data and generate new data by linear [28, 36, 73] or non-linear [8, 29, 30, 52, 77] interpolations. However, to the best of our knowledge, none of these methods are applicable to generating HTTP request data. Knowledge sharing based approaches aim to leverage existing knowledge to boost the model performance on limited training data. Depending on the application scenario, different techniques are developed to achieve knowledge sharing, such as transfer learning [12, 22, 65], self-supervised learning [16, 37, 39] and meta-learning [23, 78]. Our method falls into this category. The key difference is that we propose specific and novel designs to better capture the correlation of HTTP requests in different domains.

9 CONCLUSION

We propose a novel framework, RETSINA, to detect zero-day Web attacks across multiple domains with limited training data by utilizing meta-learning. We develop a series of new designs to achieve knowledge sharing effectively. We evaluate RETSINA on 4 real-world datasets and demonstrate the effectiveness of RETSINA for multiple heterogeneous Web domains with limited training data.

REFERENCES

[1] 2023. *AQTRONiX WebKnight.* https://www.iis.net/downloads/community/2016/04/aqtronix-webknight

[2] 2023. *Google Help. Collect campaign data with custom URLs.* https://support.google.com/analytics/answer/1033863?hl=en#zippy=%2Cin-this-article
[3] 2023. *ModSecurity.* https://github.com/SpiderLabs/ModSecurity/
[4] 2023. *Naxsi.* https://github.com/nbs-system/naxsi/
[5] 2023. *Sklearn TSNE.* http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
[6] 2023. *Xray.* https://github.com/chaitin/xray/tree/master/pocs
[7] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. 2017. A closer look at memorization in deep networks. In *International conference on machine learning*. PMLR, 233–242.
[8] Francesco Calimeri, Aldo Marzullo, Claudio Stamile, and Giorgio Terracina. 2017. Biomedical data augmentation using generative adversarial neural networks. In *International conference on artificial neural networks*. Springer, 626–634.
[9] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. Ieee, 39–57.
[10] Anshuman Chhabra, Abhishek Roy, and Prasant Mohapatra. 2020. Suspicion-free adversarial attacks on clustering algorithms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3625–3632.
[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
[12] Dan C Cireşan, Ueli Meier, and Jürgen Schmidhuber. 2012. Transfer learning for Latin and Chinese characters with deep neural networks. In *The 2012 international joint conference on neural networks (IJCNN)*. IEEE, 1–6.
[13] Gabriela F Cretu, Angelos Stavrou, Michael E Locasto, Salvatore J Stolfo, and Angelos D Keromytis. 2008. Casting out demons: Sanitizing training data for anomaly sensors. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE.
[14] Gabriela F Cretu-Ciocarlie, Angelos Stavrou, Michael E Locasto, and Salvatore J Stolfo. 2009. Adaptive anomaly detection via self-calibration and dynamic updating. In *Recent Advances in Intrusion Detection: 12th International Symposium, RAID 2009, September 23-25, 2009. Proceedings 12*. Springer, 41–60.
[15] Saikat Das, Mohammad Ashrafuzzaman, Frederick T Sheldon, and Sajjan Shiva. 2020. Network intrusion detection using natural language processing and ensemble machine learning. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 829–835.
[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[17] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.
[18] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*. PMLR, 1126–1135.
[19] Mateusz Gniewkowski, Henryk Maciejewski, Tomasz R Surmacz, and Wiktor Walentynowicz. 2021. HTTP2vec: Embedding of HTTP Requests for Detection of Anomalous Traffic. *arXiv preprint arXiv:2108.01763* (2021).
[20] George Gondim-Ribeiro, Pedro Tabacof, and Eduardo Valle. 2018. Adversarial attacks on variational autoencoders. *arXiv preprint arXiv:1806.04646* (2018).
[21] Adam Goodge, Bryan Hooi, See-Kiong Ng, and Wee Siong Ng. 2020. Robustness of Autoencoders for Anomaly Detection Under Adversarial Impact.. In *IJCAI*. 1244–1250.
[22] Jiatao Gu, Hany Hassan, Jacob Devlin, and Victor OK Li. 2018. Universal neural machine translation for extremely low resource languages. *arXiv preprint arXiv:1802.05368* (2018).
[23] Jiatao Gu, Yong Wang, Yun Chen, Kyunghyun Cho, and Victor OK Li. 2018. Meta-learning for low-resource neural machine translation. *arXiv preprint arXiv:1808.08437* (2018).
[24] Shangbin Han, Qianhong Wu, Han Zhang, Bo Qin, Jiankun Hu, Xingang Shi, Linfeng Liu, and Xia Yin. 2021. Log-based anomaly detection with robust feature extraction and online learning. *IEEE Transactions on Information Forensics and Security* 16 (2021), 2300–2311.
[25] Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).
[26] He Huang, Haojiang Deng, Jun Chen, Luchao Han, and Wei Wang. 2018. Automatic multi-task learning system for abnormal network traffic detection. *International Journal of Emerging Technologies in Learning* 13, 4 (2018).
[27] Hai Huang, Jiaming Mu, Neil Zhenqiang Gong, Qi Li, Bin Liu, and Mingwei Xu. 2021. Data poisoning attacks to deep learning based recommender systems. *arXiv preprint arXiv:2101.02644* (2021).
[28] Hiroshi Inoue. 2018. Data augmentation by pairing samples for images classification. *arXiv preprint arXiv:1801.02929* (2018).
[29] Zubayer Islam, Mohamed Abdel-Aty, Qing Cai, and Jinghui Yuan. 2021. Crash data augmentation using variational autoencoder. *Accident Analysis & Prevention* 151 (2021), 105950.
[30] Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. 2020. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *2020 IEEE symposium on security and privacy (SP)*. IEEE, 1190–1206.
[31] Xiang Jiang, Mohammad Havaei, Gabriel Chartrand, Hassan Chouaib, Thomas Vincent, Andrew Jesson, Nicolas Chapados, and Stan Matwin. 2018. On the importance of attention in meta-learning for few-shot text classification. *arXiv preprint arXiv:1806.00852* (2018).
[32] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[33] Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv preprint arXiv:1805.06201* (2018).
[34] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271* (2018).
[35] Sainan Li, Qilei Yin, Guoliang Li, Qi Li, Zhuotao Liu, and Jinwei Zhu. 2022. Unsupervised contextual anomaly detection for database systems. In *Proceedings of the 2022 International Conference on Management of Data*. 788–802.
[36] Zhi Li, Jun Guo, Wenli Jiao, Pengfei Xu, Baoying Liu, and Xiaowei Zhao. 2020. Random linear interpolation data augmentation for person re-identification. *Multimedia Tools and Applications* 79, 7 (2020), 4931–4947.
[37] Junjie Liang, Wenbo Guo, Tongbo Luo, Honavar Vasant, Gang Wang, and Xinyu Xing. 2021. FARE: enabling fine-grained attack categorization under low-quality labeled data. In *Proceedings of The Network and Distributed System Security Symposium (NDSS)*.
[38] Jingxi Liang, Wen Zhao, and Wei Ye. 2017. Anomaly-based web attack detection: a deep learning approach. In *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*. 80–85.
[39] Chen Liu, Yanwei Fu, Chengming Xu, Siqian Yang, Jilin Li, Chengjie Wang, and Li Zhang. 2021. Learning a few-shot embedding model with contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8635–8643.
[40] Jiaxin Liu, Xucheng Song, Yingjie Zhou, Xi Peng, Yanru Zhang, Pei Liu, Dapeng Wu, and Ce Zhu. 2022. Deep anomaly detection in packet payload. *Neurocomputing* 485 (2022), 205–218.
[41] Tianlong Liu, Yu Qi, Liang Shi, and Jianan Yan. 2019. Locate-Then-Detect: Real-time Web Attack Detection via Attention-based Deep Neural Networks.. In *IJCAI*. 4725–4731.
[42] Xiaolin Liu, Shuhao Li, Yongzheng Zhang, Xiaochun Yun, and Jia Li. 2020. Challenge Collapsar (CC) Attack Traffic Detection Based on Packet Field Differentiated Preprocessing and Deep Neural Network. In *International Conference on Computational Science*. Springer, 282–296.
[43] Zhuotao Liu, Hao Zhao, Sainan Li, Qi Li, Tao Wei, and Yu Wang. 2021. Privilege-Escalation Vulnerability Discovery for Large-scale RPC Services: Principle, Design, and Deployment. In *ACM Asia CCS*.
[44] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[45] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3994–4003.
[46] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*. 27–38.
[47] Andrew Ng et al. 2011. Sparse autoencoder. *CS294A Lecture notes* 72, 2011 (2011), 1–19.
[48] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999* (2018).
[49] Seungyoung Park, Myungjin Kim, and Seokwoo Lee. 2018. Anomaly detection for HTTP using convolutional autoencoders. *IEEE Access* 6 (2018), 70884–70901.
[50] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, Lorenzo Cavallaro, et al. 2019. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th USENIX Security Symposium*. USENIX Association, 729–746.
[51] Xiaochang Peng, Chuan Wang, Daniel Gildea, and Nianwen Xue. 2017. Addressing the data sparsity issue in neural amr parsing. *arXiv preprint arXiv:1702.05053* (2017).
[52] Luis Perez and Jason Wang. 2017. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621* (2017).
[53] Stefan Prandl, Mihai Lazarescu, and Duc-Son Pham. 2015. A study of web application firewall solutions. In *International conference on information systems security*. Springer, 501–510.
[54] David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. 2017. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694* (2017).
[55] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. *Advances in neural information processing*

systems 31 (2018).
[56] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[57] Samuel L Smith, David HP Turban, Steven Hamblin, and Nils Y Hammerla. 2017. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *arXiv preprint arXiv:1702.03859* (2017).
[58] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. *Advances in neural information processing systems* 30 (2017).
[59] Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 231–235.
[60] Ruming Tang, Zheng Yang, Zeyan Li, Weibin Meng, Haixin Wang, Qi Li, Yongqian Sun, Dan Pei, Tao Wei, Yanfei Xu, et al. 2020. Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2479–2488.
[61] Hung-Yu Tseng, Hsin-Ying Lee, Jia-Bin Huang, and Ming-Hsuan Yang. 2020. Cross-domain few-shot classification via learned feature-wise transformation. *arXiv preprint arXiv:2001.08735* (2020).
[62] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).
[63] Ali Moradi Vartouni, Saeed Sedighian Kashi, and Mohammad Teshnehlab. 2018. 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.
[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[65] Dong Wang and Thomas Fang Zheng. 2015. Transfer learning for speech and language processing. In *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE, 1225–1237.
[66] Shanshan Wang, Qiben Yan, Zhenxiang Chen, Bo Yang, Chuan Zhao, and Mauro Conti. 2017. Detecting android malware leveraging text semantics of network flows. *IEEE Transactions on Information Forensics and Security* 13, 5 (2017).
[67] Jason Wei and Kai Zou. 2019. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196* (2019).
[68] Jiayun Xu, Yingjiu Li, and Robert H Deng. 2021. Differential training: A generic framework to reduce label noises for android malware detection. (2021).
[69] Teng Xu, Gerard Goossen, Huseyin Kerem Cevahir, Sara Khodeir, Yingyezhe Jin, Frank Li, Shawn Shan, Sagar Patel, David Freeman, and Paul Pearce. 2021. Deep entity classification: Abusive account detection for online social networks. In *30th USENIX Security Symposium (USENIX Security 21)*.
[70] Xu Yang, Cheng Deng, Kun Wei, Junchi Yan, and Wei Liu. 2020. Adversarial learning for robust deep clustering. *Advances in Neural Information Processing Systems* 33 (2020), 9098–9108.
[71] Kaichao You, Zhi Kou, Mingsheng Long, and Jianmin Wang. 2020. Co-tuning for transfer learning. *Advances in Neural Information Processing Systems* 33 (2020).
[72] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems* 30, 9 (2019), 2805–2824.
[73] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).
[74] Ming Zhang, Boyi Xu, Shuai Bai, Shuaibing Lu, and Zhechao Lin. 2017. A deep learning method to detect web attacks using a specially designed CNN. In *Neural Information Processing: 24th International Conference, ICONIP 2017, November 14–18, 2017, Proceedings, Part V 24*. Springer, 828–836.
[75] Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[76] Ya-Lin Zhang, Longfei Li, Jun Zhou, Xiaolong Li, and Zhi-Hua Zhou. 2018. Anomaly detection with partially observed anomalies. In *Companion Proceedings of the The Web Conference 2018*. 639–646.
[77] Panpan Zheng, Shuhan Yuan, Xintao Wu, Jun Li, and Aidong Lu. 2019. One-class adversarial nets for fraud detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 1286–1293.
[78] Wenbo Zheng, Chao Gou, Lan Yan, and Shaocong Mo. 2020. Learning to classify: A flow-based relation network for encrypted traffic classification. In *Proceedings of The Web Conference 2020*. 13–22.
[79] Zibin Zheng, Yilei Zhang, and Michael R Lyu. 2012. Investigating QoS of real-world web services. *IEEE transactions on services computing* 7, 1 (2012), 32–39.
[80] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2020. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 13001–13008.
[81] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*. 207–212.

	REs
Decimal numeral	`[0-9]+`
Hexadecimal numeral	`[0-9a-f]+`
Letters and digits	`[0-9a-z]+`
List of numbers	`(\d+\.)+\d+|(\d+|)+\d+|(\d+\:)+\d+|\d+`
...	

(a) Candidate regular expressions

```
/preload?vid=q0040fm4&timestamp=1600000009
&path=preload_play/play/
/preload?vid=r00ytp0z&timestamp=1690007079
&path=pages/play/
/preload?vid=y331kn5n&timestamp=1690807105
&path=pages/index/
...
```

(b) Examples of HTTP requests

Key	`vid`
Values	`q0040fm4`
	`r00ytp0z`
	`y331kn5n`
	`...`
RE	`[0-9a-z]{8}`

(c) RE generation for "`vid`"

Key	`timestamp`
Values	`1600000009`
	`1690007079`
	`1690807105`
	`...`
RE	`[0-9]{10}`

(d) RE generation for "`timestamp`"

Figure 11: Examples of regular expression (RE) generation. (a) lists several candidate regular expressions. (b) lists three original HTTP requests. (c) and (d) show the values and the generated regular expressions of keys "`vid`" and "`timestamp`", respectively.

A DETAILS FOR ADAPTIVE PREPROCESSING

We first illustrate details about generating regular expressions and merging tokens. Then we conduct a case study to demonstrate how our adaptive preprocessing differs from existing works.

We generate regular expressions by the following steps. First, given each key and its corresponding values, we enumerate all candidate regular expressions matching these values. Figure 11 (a) shows several candidate regular expressions we devise, which cover the majority of distinct values with inessential information. Note that, letters have been converted to lowercase before regular expression generation. Second, we select the smallest regular expression among the matched regular expressions, which allows us to best extract accurate semantics. Third, we also constrain the repetition of patterns in the regular expression according to the length of values. If values are of the same length x, we set a common length x (e.g., "`[0-9a-z]{7}`"). Otherwise, we set a minimum length x such that the lengths of all values are not smaller than x (e.g., "`[0-9a-z]{11,}`"). Note that, we relax the constraint of the proportion of values that can be matched by a generated regular expression, instead of requiring the regular expression to match all values, which can eliminate the impact of noises in the requests.

Generally, after obtaining the regular expression for a key, we merge the values of the key that match the regular expression using

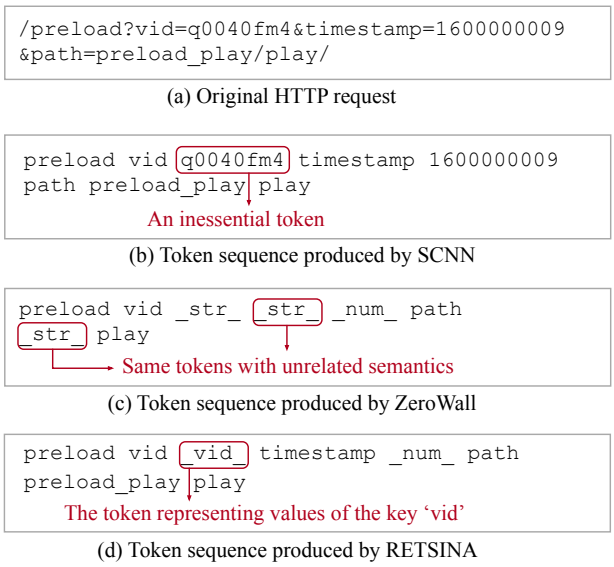


Figure 12: Comparison of different preprocessing techniques.

a placeholder "_{key}_", where "{key}" represents the name of this key. Especially when the generated regular expression of a key represents a decimal numeral or a hexadecimal numeral, its values share similar semantics across different domains. Therefore, its values will be merged using the same placeholders, e.g., "_hexnum_".

We now give examples of regular expression generation. Figure 11 (b) shows three examples of HTTP requests in a domain, where "vid" represents the identification number of items and "timestamp" represents the timestamp of requests. The number of distinct values of "vid" or "timestamp" could be exceedingly large. Several values and generated regular expressions of key "vid" and "timestamp" are presented in Figure 11 (c) and (d), respectively. For key "vid", all its values only match "[0-9a-z]+" among all candidate regular expressions and have a length of 8. Thus the regular expression of "vid" is "[0-9a-z]{8}". For key "timestamp", it can be seen that all its values match the first three candidate regular expressions, and all have a length of 10. We pick "[0-9]+" since it has the smallest set of elements (i.e., 0-9). Thus we generate "[0-9]{10}".

Our adaptive preprocessing merges tokens in the consideration that the removal of redundant information for HTTP requests helps better extract accurate semantics. Therefore, from the aspect of token merging, we categorize the preprocessing techniques in existing works into two types: i) preprocessing that considers only specific cases of token merging [49, 63, 74] and ii) preprocessing with general strategies for token merging [38, 60]. Among our baselines, SCNN and ZeroWall are representative methods of these two categories respectively and we compare our method with them.

Figures 12 (b)-(d) present the token sequences produced by different preprocessing methods given the first HTTP request in Figure 11 (b). Since type I preprocessing does not take into account the removal of redundant information from HTTP requests or only considers simplistic cases, they may result in a large number of tokens, such as q0040fm4 and other possible "vid" values. These inessential tokens would degrade the detection performance due to the data sparsity issue [51]. Type II preprocessing leverages simple but generic strategies for preprocessing, such as replacing all strings of length over 8 with the same token. This strategy can merge different values of "vid", but it may also incorrectly replace other tokens, such as "timestamp" and "preload_play". While it is possible to limit the replacement to only the "vid" field based on expert experience, such an approach is not extensible. Instead, our method addresses the above issues by automatically finding merging strategies for each key, which helps to identify and remove inessential tokens in a more systematic and scalable way. Our preprocessing recognizes that most of the values of "vid" are strings of length 8 and replaces these strings with the same token.

Table 9: Performance comparisons when choosing different base domains. The symbol "*" denotes that the base domain is selected using our method.

Target Domain	Time	Base Domain	Pre	Rec	F1
OV	5 min	SNS	0.896	0.929	0.912
		SP	0.890	0.921	0.905
		* IdM	0.901	0.926	**0.913**
	1 day	SNS	0.932	0.885	**0.908**
		SP	0.919	0.853	0.885
		* IdM	0.921	0.884	0.902
IdM	5 min	* OV	0.868	1.000	**0.929**
		SNS	0.895	0.864	0.879
		SP	0.889	0.949	0.918
	1 day	* OV	1.000	0.915	**0.956**
		SNS	0.948	0.932	0.940
		SP	0.963	0.881	0.920

B SELECTION OF THE BASE DOMAIN

We evaluate the impact of the selection of base domains on domain OV and IdM. Specifically, we change base domains and compare their performance. The results are shown in Table 9. It can be seen that the selection of base domains has limited impacts on the model performance, and the fluctuation of F1-scores is within 0.05. Moreover, our method selects the base domain with the best performance in most cases.