Web-FTP: A Feature Transferring-Based Pre-Trained Model for Web Attack Detection

Zhenyu Guo, Qinghua Shang, Xin Li, Chengyi Li, Zijian Zhang, *Senior Member, IEEE*, Zhuo Zhang, Jingjing Hu, Jincheng An, Chuanming Huang, Yang Chen, and Yuguang Cai

***Abstract*—Web attack is a major threat to cyberspace security, so web attack detection models have become a critical task. Traditional supervised learning methods learn features of web attacks with large amounts of high-confidence labeled data, which are extremely expensive in the real world. Pre-trained models offer a novel solution with their ability to learn generic features on large unlabeled datasets. However, designing and deploying a pre-trained model for real-world web attack detection remains challenges. In this paper, we present a pre-trained model for web attack detection, including a pre-processing module, a pre-training module, and a deployment scheme. Our model significantly improves classification performance on several web attack detection datasets. Moreover, we deploy the model in real-world systems and show its potential for industrial applications.**

***Index Terms*—Web attack detection, pre-trained model, transfer learning.**

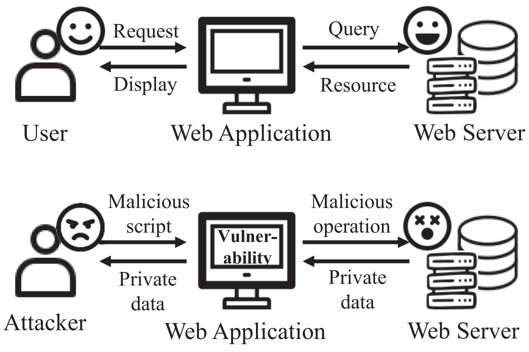


Fig. 1. Accessing and attacking Web Applications.

I. Introduction

THE phenomenal development of internet technology is transforming the way of industry, playing a significant role in our daily lives. Indeed, it is now regarded as one of the most important utilities, along with water, gas, and electricity. Web applications allow people to access or manage the information they need with a local area network (LAN) or wide area network (WAN), which can be regarded as a distributed large-scale information retrieval system based on a client-server model. In this system, users initiate their access requests in the client's web application. The clients convert them into uniform resource locators (URLs) according to the Internet Protocol (IP) and send them to the server. The server retrieves the relevant resource information according to the received URLs and feeds them back to the clients. The clients then returned to the user to satisfy their needs, thus forming complete access to the web resources.

This information system forms the backbone of numerous databases and data management platforms that provide services to businesses, enterprises, and government agencies. However, these web-based applications and databases are vulnerable to attacks. As technology advances, attackers are employing increasingly sophisticated methods to exploit various system vulnerabilities. Fig. 1 shows how web data can be accessed or attacked.

For example, attackers often target the HTTP protocol, which dominates web communication. They may tamper with HTTP requests or inject malicious HTML code into websites with security vulnerabilities, enabling them to carry out web-based attacks. This not only threatens the data security of individual users but also poses significant challenges for managing internet data effectively. As a result, detecting and preventing web attacks has become a critical priority in internet security and data protection. Researchers are continuously refining detection techniques to stay ahead in the ever-evolving battle between attackers and defenders.

A. Motivation

Current web attack detection methods rely heavily on large amounts of labeled data, which serve as supervised signals to guide models in learning the relevant features in web traffic for detection. While these methods identify known attacks efficiently, they struggle to detect new, unseen threats because they have not been trained on the features of these emerging attacks. To address this issue, datasets must be continuously updated, and

Received 15 May 2024; revised 28 September 2024; accepted 25 November 2024. Date of current version 5 February 2025. This work was supported in part by the NSFC under Grant 92270125 and Grant 62276024, and in part by the National Key Research and Development Program of China under Grant 2023YFB2704702. Recommended for acceptance by H. Hu. *(Zhenyu Guo and Qinghua Shang are contributed equally to this work.) (Corresponding authors: Zijian Zhang; Zhuo Zhang.)*

Zhenyu Guo, Xin Li, and Jingjing Hu are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: zhenyuguo@bit.edu.cn; xinli@bit.edu.cn; hujingjing@bit.edu.cn).

Qinghua Shang, Zhuo Zhang, Jincheng An, Chuanming Huang, Yang Chen, and Yuguang Cai are with QAX Security Center, Qi-AnXin Group Inc., Beijing 100000, China (e-mail: shangqqinghua@gmail.com; zhangzhuo@qianxin.com; anjincheng@qianxin.com; josjoy0413@gmail.com; gm.chenyang@gmail.com; caiyuguang@qianxin.com).

Chengyi Li and Zijian Zhang are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: chengyili@bit.edu.cn; zhangzijian@bit.edu.cn).

Digital Object Identifier 10.1109/TKDE.2024.3512793

models have to undergo frequent retraining. However, given the vast diversity and complexity of web attack traffic, obtaining sufficient labeled data for all types of attacks is impractical. Therefore, exploring detection methods that do not depend on large-scale labeled data is essential.

Fortunately, the development of pre-trained models presents a promising solution. Initially designed for natural language processing (NLP) tasks, pre-trained models learn general feature representations from large-scale, unlabeled datasets through pre-training tasks. These models can then be fine-tuned for specific tasks using a relatively small amount of labeled data, allowing them to adapt effectively to specialized tasks. The generic features learned during the pre-training phase are independent of class labels, leading to excellent generalization performance.

In the context of web attack detection, we can leverage pre-trained models to learn general features from large-scale web traffic data. By fine-tuning these neural networks with a small amount of labeled data from known web attack types, we can develop models capable of effectively detecting both known and novel web attacks with minimal dependence on large-scale labeled datasets.

B. Technology Challenges

To develop pre-trained models capable of detecting web attacks and applying them in the real world, we need to address three key technical challenges:

C1: While pre-trained models do not require large amounts of labeled data, they still rely on extensive web traffic data for pre-training to learn generic feature representations. However, the sample sizes in existing web attack datasets are often too small for the pre-training process to capture effective features, which poses a significant challenge.

C2: Most pre-trained models are designed for natural language sequence data, yet web traffic data differs significantly from natural language in terms of disambiguation and semantics. Applying NLP processing methods and models directly to web traffic data will not effectively capture the unique features required for detecting web attacks. Therefore, it is crucial to design model architectures and pre-training tasks specifically tailored for web traffic, which presents a challenge.

C3: The real-world web traffic environment often differs from the conditions under which pre-trained models are tested. Additionally, pre-trained models are typically larger and more complex than conventional supervised learning models, leading to challenges in deploying them in industrial settings. Developing an effective strategy for deploying pre-trained models in these environments is another critical challenge.

C. Our Contributions

In this paper, we propose a novel approach to address the challenge of insufficient web attack data for supporting pre-trained models (C1). We identify implicit semantic similarities between code search and web attack detection in terms of both data and tasks. Leveraging this insight, we suggest a cross-domain migration pre-training strategy that uses large-scale code data for pre-training. We introduce a new pre-trained framework for the model, which is then fine-tuned with a small amount of web attack data. This enables the generic sequence features learned by the pre-trained model to be applied to web attack detection.

To tackle the challenge of adapting models for web traffic data (C2), we develop a pre-trained framework tailored to the sequence features of web attack data. This framework includes a data pre-processing module and a generative adversarial training module with an alternative mask pre-training task.

For real-world application (C3), we design a comprehensive three-stage model deployment scheme that is optimized for industrial platforms. This deployment strategy has been tested and successfully implemented on specific commercial platforms, yielding significant practical results and demonstrating notable social impact. Our results highlight the potential of AI technology to serve both industry and society effectively.

In summary, this paper makes the following contributions:

- We propose a cross-domain migration pre-trained method to address the lack of web attack annotation and pre-training data. Our approach uses a code dataset to pre-train the model, which is then fine-tuned with a small amount of web attack data. This allows the model to learn a generic sequence feature representation suitable for web attack detection.
- We introduce a comprehensive pre-trained framework for web attack detection. This framework includes a data preprocessing module, a generative adversarial training module, and an alternative mask pre-training task. It effectively mines deep sequence features from web attack data, enabling the model to detect both known and unknown web attacks, thereby significantly enhancing its generalization ability and applicability.
- We propose a pre-trained model deployment scheme for web attack detection to demonstrate the practicality of our model. We tested it with a real-world traffic monitoring system, using real-time data to reflect the conditions encountered in industrial applications.
- Our model exhibits robust performance on both publicly available datasets and real-world traffic data, demonstrating that our approach provides tangible, reliable outcomes suitable for practical implementation.

The rest of this paper is organized as follows. First we recall the related works in Section II and briefly describe Web Attacks and Code Search in Section III. Then, the detection model is proposed in Section IV. Following that, Section V describes how we deploy the model in the real world. Section VI shows the performance of our model on public datasets and in the real world. Finally, the conclusion is drawn in Section VII.

II. Related Work

The key issue in classification against web attacks is how to learn the features of the data. Existing research can usually be divided into two main categories: statistical feature-based and deep feature-based methods.

Statistical feature-based web attack classification methods generally use statistical algorithms to pre-process and extract features from the dataset and use classical machine learning methods for classification. As early as 2011, Komiya R et al. [1] proposed a classifier for detecting SQL injection and XSS attacks, which uses TD-IDF for weight calculation after manually specifying the separator, and uses a support vector machine, simple Bayesian network, and k-NN to classify the attacks, and achieves more than 90% classification accuracy on a private dataset. Li et al. [2] improved this by using Word2Vec and TF-IDF to extract features and boost to classify the attacks and achieved more than 94% classification accuracy on several datasets. However, the practical use of this method depends on statistical modeling of behavior and demands a significant amount of labeled data. It is unable to learn deep attack pattern features and is unsuitable for identifying intricate attack patterns [3].

Deep feature-based web attack classification methods can fully exploit the deep features behind the data using neural network models and use them for classification. The main feature of neural network models is that they propagate information in hidden layers that remember information that has been processed before, thus bringing the structural advantage for processing the time-series information. Gong et al. [4] argued that the labeling errors of web attack datasets affect the model accuracy and introduced model uncertainty into deep learning-based web attack classification, so they used CNN networks as classifiers and achieved more than 95% classification accuracy on both the private dataset and the CSIC public dataset. Liang et al. [5] proposed a web attack classification model using two RNN networks, one supervised and the other unsupervised. They used CNN networks as classifiers and achieved more than 95% classification accuracy on two private datasets and the CSIC public dataset. Tian et al. [6] proposed a distributed deep learning model applied on edge devices, which is modified for the ResNet network and combined with the word2vec method to achieve more than 99% classification accuracy on the CSIC dataset. Tekerek et al. [7] used a bag-of-words model (CBOW) to extract features from web requests, convert them into images, and classify them using a CNN, achieving up to 97% classification accuracy on the CSIC2010v2 dataset.

However, both statistical and deep learning methods rely heavily on the supervised signals provided by large amounts of high-confidence labeled data to learn effective features, which are extremely expensive. The learned features are strongly correlated with the quality of the annotations, which can easily introduce biased information. These problems become more apparent as attack schemes become more innovative.

In recent years, pre-trained Transformer-based models, such as BERT, have been used for word embedding for web attack classification due to their excellent performance in NLP. Seyyar et al. [8] used BERT for word embedding and Multi-Layer Perceptron (MLP) for classification and achieved over 99% accuracy on the CSIC 2010 dataset. Jú nior et al. [9] also used BERT for word embedding and BiLSTM for classification and achieved over 99% accuracy on several datasets. Bokolo [10] experimented with DistilBERT and achieved over 80% accuracy. Gniewkowski et al. [11], [12] used RoBERTa for word embedding and several machine learning methods in a downstream classification task, achieving over 99% accuracy on several datasets.

However, all of these methods borrow part of the pre-trained model to generate vector representations. A more complete pre-trained method for web attack traffic has not yet appeared, mainly because pre-trained models often require a large amount of data for training before they can learn their inherent generic features, and the size of the current publicly available web attack dataset is difficult to satisfy the demand; on the other hand, web attack traffic, unlike general NLP data, has its unique sequence features and structural features, so existing pre-trained models cannot receive web data as input for pre-training, and other solutions must be found.

III. Preliminary

In this section, we will explore the different common types of web attacks, analyze specific examples, and discuss the challenges faced by current web attack detection methods.

A. Web Attack

A web attack targets a user's online behavior or a device such as a web server. This includes actions like implanting malicious code, modifying website permissions, and obtaining private information about website users. As mentioned in the previous section, web application security is crucial for any web-based business. Ensuring the security of web applications is essential to prevent unauthorized access, misuse, modification, or destruction.

Common web attack methods include *XSS* attacks, *SQL injection* attacks, and *DDoS* attacks.

XSS (Cross-Site Scripting) is one of the most common attack methods in web applications. The principle behind XSS attacks involves the use of forms commonly found on the front end of web applications to submit information to the back end or server. Attackers often embed malicious scripting programs in these forms. For example:

```
<input type = “text” name = “user” value = “cbuc”/>
```

This script gets executed by the web application to submit the username entered by the user in the form, but if the attacker enters not a normal string but the malicious script:

```
“/><script>alert(“bingo”) </script><! - ”
```

At this point, the submitted script would be:

```
<input type=“text” name=“user” value=“”/>
<script>alert(“bingo”) </script><!-” />
```

Then the server detects the value exception, redirects the page, and executes the command $alert(“bingo”)$. This will cause a warning box to pop up. With a few modifications, the attacker can do far more than just display an alert box. The malicious script can steal the user's cookies, usernames, and passwords, or hijack the front-end page, thereby compromising the user's information security. This highlights the serious risks posed by XSS attacks and underscores the importance of robust web application security measures.

SQL injection is the most common attack on web databases. Attackers disguise SQL commands as normal request parameters and pass them to the server, tricking it into executing malicious SQL commands to achieve unauthorized access or other malicious purposes. For example, in an attack log, an attacker might input the following:

```
anadir.jsp?id=2&nombre=Jam%F3n+Ib%E9rico&
precio=85&cantidad=%27%3B+DROP+TABLE+
usuarios%3B+SELECT+*+FROM+datos+
WHERE+nombre+LIKE+sofia
```

A SQL injection vulnerability exists in the $cantidad$ field, where an attacker writes content that exploits the vulnerability. Once the web server receives the segment, it parses and executes the associated SQL statement:

```
DROP TABLE usuarios;
SELECT * FROM datos WHERE nombre LIKE sofia
```

This attack request allows the attacker to simultaneously query a segment of data stored in the web server and delete the specified table, preventing the server from operating normally. By writing other statements, the attacker can even bypass the server's authentication process to take over the server remotely.

DDoS attack is the most powerful and effective method of attack known as Distributed Denial of Service. It requires the use of a public network and a huge number of computer devices as a platform for launching malicious requests. This makes it difficult for hosts to cope with the large number of requests in a short time, paralyzing the target host. Depending on the content of the request, it can be subdivided into three main types of attack: SYN Flood, DNS Query Flood, and HTTP Flood. In a SYN Flood Attack, a large number of IP addresses are forged to send SYN messages to the server. In a DNS Query Flood Attack, ports and client IPs are forged to send a huge number of domain name resolution requests. In an HTTP Flood Attack, a large number of anonymous HTTP proxies are forged to send malicious requests. These attacks are designed to overwhelm the system with a large number of meaningless query commands in a short time, occupying a significant amount of system resources and slowing down the back-end business processing system.

Although various web attacks may exhibit different behaviors on the client side, they all rely on communication with the server to transmit attack commands. This communication traffic often contains critical information, such as specific attack instructions, which can be inadvertently executed by the server when parsing the request packets, resulting in the intended effects of the attack.

By carefully analyzing the features of specific fields in network traffic data, we can identify different attack methods. For example, as with the earlier example of SQL injection, the specific traffic packet information corresponds to the nature of the attack. Therefore, thorough analysis of traffic data can help uncover distinct attack patterns, aiding in the detection and mitigation of web attacks.

B. Code Search

Code search is the task of retrieving relevant code given a natural language query. This requires bridging the gap between the language used in code, which is often abbreviated and highly technical, and natural language, which is better suited for describing vague concepts and ideas.

Researchers usually build code search tools with text-based queries or semantic analysis. Text-based queries perform code search according to a relevancy algorithm (e.g., TF-IDF) that matches the query with candidate code snippets. However, this approach struggles to comprehend long natural language queries. On the other hand, semantic analysis captures the syntactic and semantic features of the code using a learning model that learns the relationship between code and query from large-scale datasets. As a result, semantic analysis has become an attractive alternative to traditional text-based models.

Formally, code search aims to retrieve the corresponding code snippet $x \in X$ that matches the semantics of the query y. The semantic analysis has an encoder to represent the code snippet x as code vector V_x:

$$V_x = E_x(x) \tag{1}$$

where $E_x(\cdot)$ is the neural network to encode source code. Then, the natural language query y is mapped to the same semantic space as V_x, represented as query vector V_y:

$$V_y = E_y(y) \tag{2}$$

where $E_y(\cdot)$ is another neural network to encode natural language queries. Finally, the similarity value in the matching step is calculated as:

$$Similarity = S\left(V_x, V_y\right) \tag{3}$$

where $S(\cdot)$ is a similarity function between code vector V_x and query vector V_y. By maximizing the similarity function, we can get the most relative code to a given description.

In short, the current mainstream approach to the Code Search task is to match the semantic features of query statements and code. The main idea involves designing the encoder E to efficiently extract deep semantic features from the codes.

IV. Detection Model

An overview of the proposed Web-FTP framework is depicted in Fig. 2. The pipeline of Web-FTP can be divided into three parts: preprocessing, pre-training, and fine-tuning.

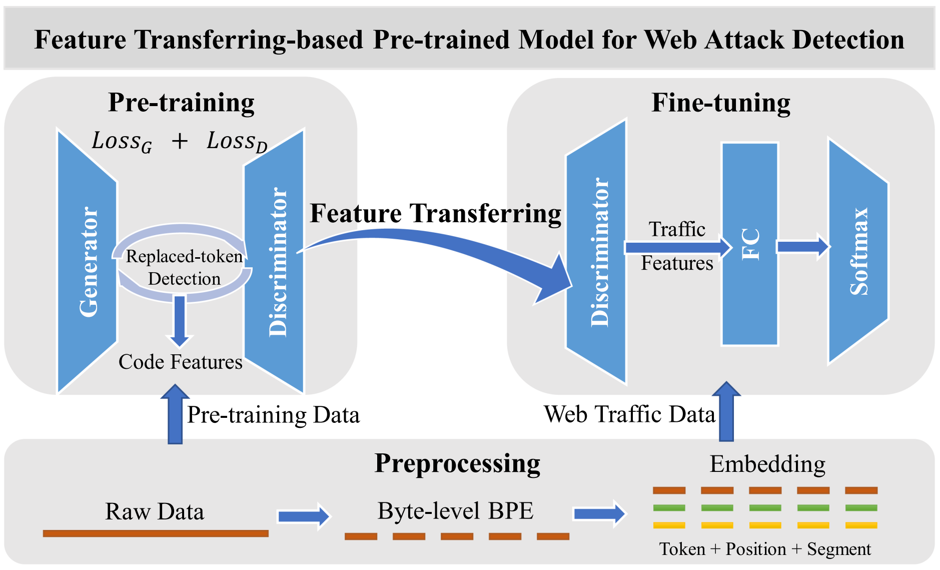


Fig. 2. Overview of Web-FTP, illustrating the three modules as preprocessing, pre-training, and fine-tuning.

TABLE I
Comparison of the Size of the Public Web Attack Datasets and the Pre-Trained Model Datasets

Dataset	Positive	Negative	All	Size
CSIC 2010	72000	25065	97065	3.22MB
PKDD 2007	35006	15110	50116	31.9MB
SR-BH 2020	525195	382619	907814	416MB
FWAF	1294531	48126	1342657	22.9MB
Wikipedia	-	-	$\geq$ 2500000000	90GB
Bookcorpus	-	-	74004228	5.8GB
CICIDS2017	-	-	3119345	51.1GB
ISCXVPN2016	18745956	3977556	22723512	24.6GB

A. *Dataset for Pre-Training*

Pre-trained models can eliminate the need for labeled data but also require sufficient unlabelled data for feature learning. The most famous pre-trained model for sequential tasks is BERT [13]. Its pre-training data comes from two datasets: Wikipedia and Bookcorpus [14]. Wikipedia has 90 GB of data and Bookcorpus has 5.8 GB of data. The pre-training data of ET-BERT [15], a variant of BERT, comes from the public datasets CICIDS2017 [16] and ISCXVPN2016 [17], making it an ideal source for pre-training a network traffic classification model [18].

And for the Web Malicious Attack Detection task, currently recognized public datasets for web attacks include CSIC 2010 [19], ECML-PKDD 2007 [20], SR-BH 2020 [21] and FWAF [22]. The sizes of these datasets are shown in Table I.

From Table I, we can observe that the largest publicly available web attack dataset contains 1342657 entries. However, this scale is insufficient to achieve effective pre-training for a model of this nature. For instance, BERT used BooksCorpus (800M words) [23] and English Wikipedia (2500M words) for pre-training. The relatively small size of web attack datasets severely limits the model's ability to learn general sequence features. Therefore, we need to find data that is more suitable for pre-training the model.

To address this, we are investigating web attack data samples to generate new ideas for suitable pre-training data. Web attack data consists of multiple components, and as an example, we have selected a sample from the CSIC 2010 dataset, represented as follows:

```
GET http://localhost:8080/tienda1/publico/anadir.jsp?
&id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&
cantidad=%27%3B+DROP+TABLE+usuarios%3B+
SELECT+*+FROM+datos+WHERE+nombre+LIKE
+%27%25&B1=A%F1adir+al+carrito HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5;
Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+
xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=B92A8B48B9008CD29F622
A994E0F650D
Connection: close
```

One web traffic record reveals a clear SQL injection vulnerability in the GET request. The attacker attempts to exploit this vulnerability, as discussed in the Preliminary report. The core attack statement is:

```
'; DROP TABLE usuarios; SELECT * FROM datos
WHERE nombre LIKE '
```

TABLE II
DATASETS COMMONLY USED IN CODE SEARCH

Dataset	Samples	Type of Code
StaQC	119519	SQL
SWIM	$\geq$ 3000000	C#
CodeSearchNet	6452446	Java, JavaScript, Python, Ruby, PHP
DeepCom	485812	Java
DeepCodeSearch	16262602	Java
GitHub-Android	787000	Android

This SQL code is the core element that turns the entire traffic packet into a malicious attack. It is not displayed directly in the traffic but is embedded into the request following network protocol rules and then encoded into the packet. This tactic circumvents standard string-matching detection methods.

The key to identifying web attacks is detecting function-specific code inserted into otherwise normal web requests. When the web server parses the request packet, it executes this malicious code, producing the desired effect for the attacker. This is the fundamental process through which web attacks are executed.

Therefore, it is essential to detect whether packets are encoded with such malicious code and whether they contain the specific data needed to realize the semantics of a web attack. This underscores the strong correlation between detecting malicious code and identifying malicious web attacks.

Web requests can be considered a form of code. While their syntax differs from traditional programming languages, they consist of structured strings that convey specific semantics according to defined rules. Web data can be viewed as blocks of code with distinct semantic functions executed by the web server to perform corresponding operations—similar to tasks in a complete code structure. By learning the semantic features of these code blocks, machine learning models can identify different semantic patterns and determine the function of a code block. If the block's function resembles that of known web attack code, we can confidently classify it as a web attack. This approach allows us to transform the task of learning the generic semantic features of web traffic data into learning the semantic features of code blocks. By doing so, pre-trained models can learn both contextual and semantic features of code statements, providing a strong data representation capability within the code domain. Once pre-trained on large code datasets, the model can be fine-tuned on a relatively small web attack dataset, which benefits from the model's excellent representation learning abilities. This enables us to transfer the generic features learned in the code domain to the web attack domain, allowing for effective representation of specific attack patterns in the web traffic data.

In summary, pre-training the model on a code semantic dataset effectively addresses the issue of limited web attack data, preventing the model from underfitting due to insufficient data volume.

We have selected several well-known public datasets for our code search tasks, including StaQC [24], SWIM [25], CodeSearchNet [26], DeepCom [27], DeepCodeSearch [28], and GitHub-Android [29]. The sizes of these datasets are shown in Table II.

These datasets, StaQC, DeepCom, and GitHub-Android, have less than 1 million data volumes and are therefore unsuitable for use as pre-training datasets. Similarly, SWIM and DeepCodeSearch data are focused on C Sharp and Java respectively, and their code semantic structures are relatively homogeneous, which will reduce the generalization of the model if we use these two datasets for pre-training. Using these two datasets for pre-training will reduce the model's generalization ability and cause accuracy issues when faced with web attack traffic with multiple code structures, particularly when dealing with code structures not seen in the training set. We have chosen the CodeSearchNet dataset as our pre-training dataset. It contains about 6000000 pieces of code data including JavaScript, PHP, and Python code. Table III shows the dataset statistics in CodeSearchNet, which are in the form of code source code, preserving the before-and-after structural associations. This allows the model to fully learn the contextual features among structured texts from them, laying the foundation for subsequent migration to the Web domain and fine-tuning.

TABLE III
THE STATISTICAL INFORMATION OF CODESEARCHNET

	Number of Functions	
	w/documentation	All
Go	347789	726768
Java	542991	1569889
JavaScript	157988	1857835
PHP	717313	977821
Python	503502	1156085
Ruby	57393	164048
All	2326976	6452446

B. Preprocessing

After selecting CodeSearchNet as our pre-training dataset, we considered how best to process the data so that the model could effectively learn deeper semantic information from both code and web attack data.

Existing approaches typically extract URLs from web attacks and feed them into sequence models to extract features. For example, with the previously mentioned attack dataset, current methods focus solely on the URL + Payload portion of the HTTP request, discarding other fields. This approach overlooks potentially valuable feature information that could be present in those omitted fields. While this may not significantly impact supervised learning approaches—where strong labeling information provides guidance—the absence of supervisory signals in pre-trained models necessitates incorporating as much feature information as possible. By doing so, we can capture more correlations and improve the model's overall effectiveness. Therefore, our model aims to accept the entire HTTP request packet as input, rather than just the URL + Payload fields.

Following the rules of the web data, the CodeSearchNet dataset requires preprocessing before being fed into the model. Instead of treating each piece of code as a simple sequence of data, we treat each block of code as a sequential unit because it represents a collection of multiple code segments with full semantics and functionality (e.g., a complete function definition). This allows us to draw a parallel between a web request packet and a code block, as both encapsulate a complete action with similar semantic functionality.

Each request packet or code block must be preprocessed to ensure it conforms to a format the pre-trained model can receive seamlessly. To accomplish this, we employed character-level Byte Pair Encoding (BPE) (as referenced in Neural Machine Translation of Rare Words with Subword Units) [23] to segment and encode both the code blocks and request packets [30]. For each token, we introduced three different embedding dimensions to capture its features: Token Embedding, Position Embedding, and Segment Embedding.

In the case of CodeSearchNet, we treat a complete function along with its descriptive text as a single segment. The sequential order of each function determines its position in the input data, corresponding to a distinct segment embedding.

The syntax rules of programming languages dictate that code data should be divided into shorter statements by line breaks, then into words by space characters, and longer words into shorter words, i.e., tokens, according to the operators in the longer words. The space character is the main separator in this process. The description text is a natural language sequence, so we use RoBERTa's Tokenizer [31] to divide it into words and get Token. We train our model with a larger byte-level BPE [23] vocabulary, and each word corresponds to a token in the larger byte-level BPE vocabulary. The position of this token in the vocabulary serves as the token embedding for this word. We superimpose the three embeddings to get the preprocessed code data representation. In the same function, the order of occurrence of each word indicates its position in this one text segment, corresponding to a separate Position Embedding. We build a vector with a sequence of positive integers incremented by 1 to represent the order of the tokens in the corresponding positions.

The same word always has the same Token Embedding, regardless of where it appears. The same word in the same function always has the same Segment Embedding but may have a different Position Embedding due to different positions. The same word in different functions may have the same Token Embedding and Position Embedding due to the same positions. Token Embedding and Position Embedding, but have different Segment Embedding.

For web data, we define a complete traffic data segment and design three kinds of embedding. Unlike code data, web traffic data sequences are much longer. For example, the URL sequences appeared earlier, and at this time, using the space character as a participle for segmentation is not feasible. Therefore, we consider the Traffic sequences to be treated as a piece of natural language. They are directly disambiguated using RoBERTa's Tokenizer, resulting in a token and a vocabulary. The vocabulary is specific to web data and differs from that used for code data. We obtain Segment Embedding and Position Embedding in the same way as we do for code data. We also superimpose the three embeddings to get the preprocessed Web data representation.

The specific coding process is illustrated in Fig. 3.

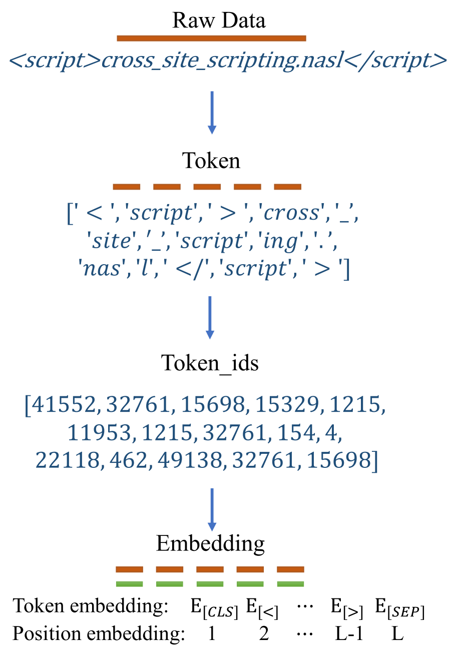

Raw Data
<script>cross_site_scripting.nasl</script>
Token
[' < ','script',' > ','cross','_',
'site','_','script','ing','.',
'nas','l',' </','script',' > ']
Token_ids
[41552, 32761, 15698, 15329, 1215,
11953, 1215, 32761, 154, 4,
22118, 462, 49138, 32761, 15698]
Embedding
Token embedding: E[CLS] E[<] ··· E[>] E[SEP]
Position embedding: 1 2 ··· L-1 L

Fig. 3. How to preprocess web traffic data into inputs.

C. Pre-Training

In the pre-trained module, we leverage the ELECTRA algorithm [32] and adopt the replaced-token detection task to learn the deep sequence features of the web attack data as we consider both traffic packets and code data as textual sequential data. The ELECTRA framework is a textual encoder with strong feature representation capabilities for textual data and performs excellently on related tasks.

The ELECTRA algorithm is a prominent pre-trained framework designed for sequence data, where the pre-training is divided into two parts: generation and discrimination. The generator performs a mask prediction task similar to BERT, and the discriminator utilizes a separate encoder to ascertain whether the token predicted by the generator matches the masked token. Together, these two parts form a replaced-token detection task.

In our Web-FTP framework, the generator receives token embedding from the preprocessing module and randomly replaces 15% of the input traffic with the MASK token, which is the token to be predicted. The generator's encoder transforms this input into a vector representation in the hidden space, subsequently predicting the token at the replaced position and outputting a probability distribution using a softmax layer:

$$\Pr_G(x_t \mid \boldsymbol{x}) = \text{softmax}\left(E(x_t)^{\mathrm{T}} h_G(\boldsymbol{x})_t\right) \tag{4}$$

where $\boldsymbol{x}$ denotes the input sequence, x_t denotes the token of the position t to be predicted, $E(x_t)$ denotes the embedding vectors, and $h_G(\boldsymbol{x})_t$ denotes a given position in the vector representation from generator G. The loss function of the generator is:

$$L_G = -\frac{1}{m}\sum_{i=1}^{m} \log\left(\Pr_G\left(\text{MASK}_i = \text{Token}_i|\boldsymbol{x}\right)\right) \tag{5}$$

where m denotes the total number of masked tokens, MASK_i and Token_i denote the MASK and the token that should be presented at the i-th position respectively.

After the generator predicts the masked token, the discriminator determines whether the token it has generated matches the original one. In this way, the discriminator is essentially performing a binary classification task:

$$D(\boldsymbol{x}, t) = \text{sigmoid}\left(W h_D(\boldsymbol{x})_t\right) \tag{6}$$

where W denotes the parameters of the discriminator and h_D denotes the vector representation from discriminator D. The loss function of the discriminator is therefore:

$$\begin{aligned} L_D = & -\frac{1}{m}\sum_{i=1}^{m} \log\left(\Pr_D(x_i \mid \text{Token}_i = \text{MASK}_i)\right) \\ & + \log\left(1 - \Pr_G\left(x_i \mid \text{Token}_i \neq \text{MASK}_i\right)\right) \end{aligned} \tag{7}$$

The overall loss function for replaced-token detection is:

$$L = L_G + L_D. \tag{8}$$

The ELECTRA framework effectively learns feature representations of sequence data by training a generator and a discriminator with the encoder structure of BERT.

D. Fine-Tuning

The pre-trained model learns the features of the code data. To better serve the downstream tasks, we must fine-tune our model. Specifically, we need to transfer the generic features and representational capabilities acquired during pre-training on the code domain to the web attack data domain. In the pre-training phase, we use code data from multiple programming languages to ensure that the pre-trained representation is independent of any specific programming language. This allows the model to capture semantic information from generic structures within the sequences.

The fine-tuned model differs from the pre-trained model. We utilize the Discriminator component of the pre-trained model as a backbone, followed by a fully connected network layer and a Softmax layer. This combination forms a detection model for web attack data. The Discriminator, having been trained on code data, acts as an efficient encoder that extracts rich semantic features from the web attack data by representing the generic features of the code in vector form. The fully connected layer then synthesizes and processes these feature vectors, while the Softmax layer produces the final detection results for web attack data.

For fine-tuning, we use 1.5 GB of real-world traffic data captured from an industrial platform belonging to a prominent cybersecurity company. This data, stored as PCAP packets, is used to design a binary classification task. The packets are fed into the constructed fine-tuning model. The preprocessing method previously discussed transforms the web attack data into token sequences, which are then input into the fine-tuning model to obtain a binary classification result. This result determines whether the model identifies the traffic as web attack data or not. Cross-entropy loss is calculated between the predicted result and the actual web data label, and the fully connected network layer parameters are updated accordingly. This process is repeated until the model converges, completing the fine-tuning phase and producing a model capable of web attack detection.

The fine-tuning model built in this way offers two key advantages. First, it transfers the powerful representational ability of the Discriminator to the web data domain, creating a dedicated classifier for web attack detection. Second, it only uses a portion of the pre-trained model's architecture, significantly reducing the number of parameters and enhancing its suitability for real-world deployment.

V. Real-World Application

We now describe how to deploy the trained model for web attack detection tasks in real-world environments.

Real-world network environments are far more complex than public datasets. The volume of traffic data is immense, and the transmission rate is extremely fast, making data collection for detection a significant challenge. Simply intercepting network traffic directly is not a viable option, as it would negatively impact the user experience. We must account for the downstream user's experience while considering that attackers might detect the model's deployment. Therefore, minimizing the model's impact on the network environment is crucial. Additionally, deep learning models typically have slower detection rates, making them unsuitable for handling real-time detection requirements on their own. Thus, it is necessary to strike an optimal balance between practicality and accuracy.

To address these challenges, we designed a sensor-based Threat Sensitive System, where the core module is a network traffic sensor. The sensor module is a Linux-based device used for traffic collection and analysis. It is highly sensitive to network traffic, and capable of sampling and collecting the necessary data for detection, thereby mitigating the problem of traffic data collection. We deploy the sensor in a bypass mode within the network infrastructure, ensuring that we do not intercept the full network traffic for analysis. Instead, we sample portions of the traffic, which minimizes the impact on the user's network environment. By strategically placing these network traffic sensors at multiple locations within a protected network, we can gather large amounts of traffic and threat data without disrupting the user experience. This data is then sent to a centralized system responsible for threat analysis, enabling us to comprehensively assess the cybersecurity status of the entire network.

The final step involves balancing detection efficiency and accuracy. The deployed model operates in a two-step process: an initial inspection using a machine learning model, followed by re-inspection with a deep learning model. The initial inspection employs a simple tri-gram Bayesian classification model to make a preliminary assessment of whether traffic samples are likely to be malicious. Samples that cannot be conclusively judged by this model are sent to the trained deep learning model for further analysis. This two-stage approach significantly reduces the computational load on the system, enhancing overall efficiency while maintaining a high level of detection accuracy.

Therefore, the deployed system consists of three key components as shown in Fig. 4:

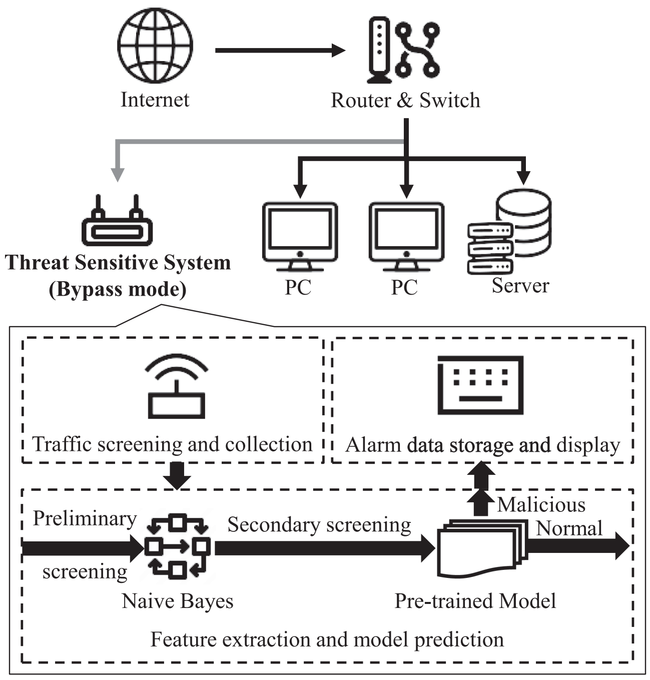

Internet
Router & Switch
Threat Sensitive System (Bypass mode)
PC
PC
Server
Traffic screening and collection
Alarm data storage and display
Preliminary screening
Naive Bayes
Secondary screening
Malicious
Normal
Pre-trained Model
Feature extraction and model prediction

Fig. 4. Deploy Web-FTP in real-world network.

1) Traffic screening and collection: The network data sensor first analyzes traffic by dividing it into sessions using a quintuple segmentation method. It then applies protocol and whitelist filtering to identify eligible traffic, extracts relevant fields, and queues the data for transmission to the machine learning detection module.
2) Feature extraction and model prediction: After receiving session field data, feature extraction is performed to obtain the session's packet distribution and payload features. A tri-gram Naïve Bayes model serves as the initial filter, using packet distribution features to quickly identify suspicious traffic. The deep learning model proposed in this paper is then applied as a secondary detection layer, focusing on the payload features of the suspicious traffic. The detection results are classified into two types: 0 represents benign (white) traffic, and 1 indicates a web attack. Information regarding traffic labeled as 1 is sent to the alerting module.
3) Alarm data storage and display: The module merges the sessions and stores them in the database to record alarm traffic information. The alarm results are also sent to the front end for display.

By implementing this system, we address the challenges of real-world deployment, enabling effective evaluation of the model's performance in practical applications and supporting regulatory traffic assessment.

VI. Evaluation

We have conducted extensive experiments on several web attack traffic datasets using Python programming language and Google Tensorflow deep learning library to implement the proposed model. We ran models on a machine with Tesla-V100, 16 GB memory, and Ubuntu18.04. In all experiments, the best performance was reported. The results demonstrate the superiority of our model.[1]

A. Metrics

We regard web attack traffic detection a a binary classification task. Therefore, We first considered $accuracy(acc)$ as a metric to evaluate the model.

$Accuracy$ is the ratio of the true positives (TPs) and true negatives (TNs) to the sum of samples (the TPs, TNs, false positives (FPs) and the false negatives (FNs)):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

However, we hope that our detecting method can find as much web attack traffic as possible when deploying the model in the real world, achieving a high $recall$.

$Recall$ is the ratio of the TPs to the sum of the TPs and the FNs, which represents the completeness of the classification:

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

At the same time, we do not hope that the method easily identifies normal traffic as attack traffic, which means the model has great $precision$.

$Precision$ means the ratio of the TPs to the sum of the TPs and the FPs, which represents the confidence of the classification:

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

Therefore, we choose the $F1 - score$ to evaluate the results, and it can be shown as:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{12}$$

$F1$ is the harmonic mean of the $precision$ and $recall$. It can comprehensively consider them to better measure the detection capability of the method than a single one. The higher the F1 score, the stronger the detection ability of our method for web attack traffic.

What's more, we also introduce AUC. AUC means the $Area\ Under\ the\ Curve$. It is a single scalar value representing the ROC performance. ROC is a receiver operating features graph that visualizes models based on their performance [33]. It is a two-dimensional graph in which the $True\ Positive\ Rate(Recall)$ is plotted on the Y axis and the $False\ Positive\ Rate$(the percentage of negatives incorrectly classified in total negatives) is plotted on the X axis. After drawing this curve, we can calculate the area of the area enclosed by the X axis. Since the AUC is the portion of the area under the ROC curve, it is always between 0 and 1.0, and it shows the relative trade-off between true positives and false positives of our model.

[1]The public data and the deployed code underlying this article are available at https://github.com/chyn0907/WebFTP

TABLE IV
COMPARISON WITH SUPERVISED LEARNING MODELS ON PUBLICLY AVAILABLE DATASETS

Method	**CSIC2010**				**ECML-PKDD 2007**			
	Recall	AUC	ACC	F1	Recall	AUC	ACC	F1
Logistic Regression	9.16%	71.57%	76.69%	16.71%	59.56%	72.32%	71.47%	55.73%
Adaboost	8.96%	69.78%	76.64%	16.37%	56.98%	81.77%	82.19%	65.86%
Random Forest	81.43%	97.17%	92.12%	84.06%	82.16%	96.52%	92.99%	87.60%
XGBoost	77.83%	97.86%	93.25%	85.47%	**84.12%**	**96.97%**	**93.66%**	**88.88%**
Web-FTP	**99.53%**	**99.98%**	**99.88%**	**99.77%**	76.20%	93.77%	92.03%	85.22%
Method	**SR-BH 2020**				**FWAF**			
	Recall	AUC	ACC	F1	Recall	AUC	ACC	F1
Logistic Regression	84.62%	92.59%	84.15%	81.82%	19.75%	94.49%	97.10%	32.63%
Adaboost	84.89%	95.49%	86.64%	84.27%	52.08%	96.90%	97.80%	62.76%
Random Forest	98.46%	99.87%	98.47%	98.19%	92.20%	99.15%	99.61%	94.45%
XGBoost	97.03%	99.69%	96.98%	96.44%	90.41%	99.79%	99.56%	93.57%
Web-FTP	**99.93%**	**99.98%**	**99.44%**	**99.34%**	**99.90%**	**99.99%**	**99.99%**	**99.89%**

TABLE V
ABLATION STUDY OF KEY COMPONENTS ON PUBLICLY AVAILABLE DATASETS

Method	**CSIC2010**				**ECML-PKDD 2007**			
	Recall	AUC	ACC	F1	Recall	AUC	ACC	F1
(w/o) Pre-training	78.94%	98.36%	94.09%	87.19%	58.04%	85.77%	85.95%	71.30%
(w/o) Tokenizer	82.40%	97.23%	94.76%	88.92%	**88.09%**	**98.12%**	**95.75%**	**92.59%**
Web-FTP	**99.53%**	**99.98%**	**99.88%**	**99.77%**	76.20%	93.77%	92.03%	85.22%
Method	**SR-BH 2020**				**FWAF**			
	Recall	AUC	ACC	F1	Recall	AUC	ACC	F1
(w/o) Pre-training	98.07%	99.89%	98.76%	98.53%	96.00%	99.79%	99.84%	97.71%
(w/o) Tokenizer	98.96%	99.98%	99.23%	99.08%	97.42%	99.26%	99.90%	98.59%
Web-FTP	**99.93%**	**99.98%**	**99.44%**	**99.34%**	**99.90%**	**99.99%**	**99.99%**	**99.89%**

B. Detection on Public Datasets

First, we compare the performance of the proposed model with a variety of supervised models on these datasets, including logistic regression [34], random forest [35], AdaBoost [36] and XGBoost [37]. The Table IV shows the results.

From the Table IV, we can see that our approach can achieve higher accuracy on classification tasks than many supervised learning models on multiple datasets. Especially, the accuracy of our model is higher than the best performers in the baseline and even higher by about 14% on the CSIC 2010, which demonstrates that our model can learn generic web attack traffic features and that these features are critical for the classification task; On ECML-PKDD 2007, ET-Muff is slightly inferior to XGBoost, it is because ECML-PKDD 2007 poses a severe imbalance in data, and all methods suffer severely from it. Meanwhile, the data in ECML-PKDD 2007 is not real-world traffic, as some fields have been randomly altered [21]. This introduces corruption, distorting the otherwise complete semantic information of the Web attacks it contains. Additionally, 10% of these attacks are out of context within the dataset. While they may resemble real attacks, they are ineffectual as they are constructed blindly and do not target the appropriate entities [38]. These spurious attack data introduce false semantic features, potentially misleading the model and decreasing accuracy, particularly when fine-tuning the model with real attack data. Although the accuracy of our method is not superior to XGBoost, it still reached the state-of-the-art compared to supervised learning methods.

In addition, our model achieves remarkable accuracy and an F1 score of more than 99% on datasets except for ECML-PKDD, while other methods have a wider range of variation, e.g., XGBoost is about 10% less accurate on the CSIC 2010 than on the others. These experiments show that even if we cannot learn specific data features from the labeled data as supervised learning models do, our model can still achieve better classification results by learning generic features during pre-trained. In practice, our model can deal with the complex composition of web attack traffic and work as a stable classification in varieties of situations.

C. Ablation Study

We show ablation results to verify the contribution of the pre-training data and the tokenizer we mentioned before. In the table, we trained these models separately and performed experiments on publicly available datasets.

Table V presents the experimental findings, with "w/o Pre-training" indicating the absence of code data pre-training and the direct use of the ELECTRA model trained on NLP data as a feature extractor for web attack detection. "w/o Tokenizer" means not using byte-level BPE but using character-level BPE in the data pre-processing procedure. "Web-FTP" refers to the model we propose as a complete solution.

It is clear that pre-training the model with code data significantly improves classification accuracy and the F1 score. This demonstrates that the domain features of code data align more closely with web attacks than those of NLP data. By fine-tuning on a limited amount of data, the model can efficiently learn relevant features and transfer them effectively to the web attack detection, resulting in a substantial performance boost.

We have also observed that enhancing the tokenizer and incorporating payload features improve the model's ability to represent web attack traffic features to varying degrees, thereby

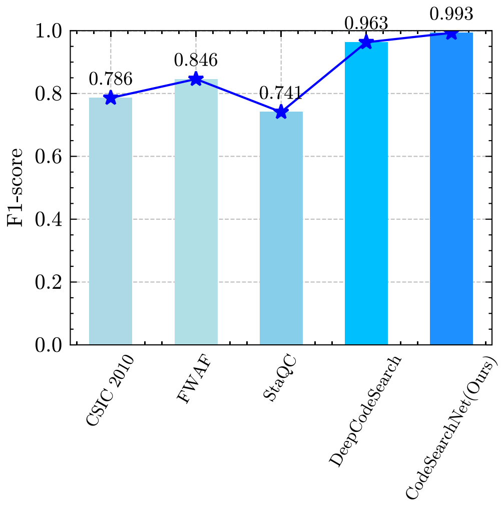


Fig. 5. Detection results of Web-FTP with different datasets for pre-training.

enhancing detection performance. However, it is important to note that the enhanced tokenizer performed less effectively on the ECML-PKDD dataset compared to the original version. This suggests that careful consideration should be given when selecting token segmentation methods for specific data types.

Furthermore, we selected alternative datasets for pre-training, and we aim to determine the impact of utilizing existing traffic datasets or other code datasets for pre-training on traffic detection. To pre-train the model, several publicly available traffic and code datasets were selected, as previously mentioned. Following this, the model was tested on the SR-BH 2020 dataset for the classification task. The experimental results are presented in Fig. 5.

The figure clearly shows that the models pre-trained using the CSIC and FWAF datasets are not as effective as DeepCodeSearch and CodeSearchNet because the amount of data in CSIC and FWAF is not sufficient for the models to learn the common sequence features adequately, which affects the accuracy. The CSIC dataset is demonstrably less effective than FWAF, due to the significantly smaller amount of data it contains. Meanwhile, DeepCodeSearch has more data than CodeSearchNet, but its classification accuracy is slightly lower. We believe that DeepCodeSearch focuses on Java data, leading to the lack of generalization performance compared to CodeSearchNet, which contains a wider range of code data. Therefore, we strongly recommend using the CodeSearNet dataset as the pre-training dataset.

On the other hand, It is clear that while StaQC and CSIC are similar in terms of data volume, their results are not as good as CSIS. This proves that the semantic features in the code data cannot completely replace the traffic data features. If there is a large-scale publicly available traffic dataset in the future, it can be used for model pre-training, and better detection results can be achieved.

D. Performance in Real World

To evaluate the model's performance in the real world, we deployed it on the industrial platform of a prominent cybersecurity company and conducted traffic detection experiments in a live environment.

TABLE VI
RESULTS IN REAL-WORLD DATA

True Label	Test Label	
	Normal	Attack
Normal	463816(TP)	909(FN)
Attack	567(FP)	1480981(TN)
Accuracy	99.87%	
Precision	99.88%	
Recall	99.80%	
F1-Score	99.84%	

TABLE VII
DETAILS OF DEPLOYED MODEL

	Model	
Layers	12	12
Batch_Size	256	128
Inference Time	130ms	70ms
Minimum Inference Time	124.4ms	66.5ms

We tested the model during a three-hour network attack and defense exercise. The exercise simulated a remote network attack on a web server located in an office that stored important data and functioning normally. Two groups participated: one group acted as network attackers, using various remote attack methods to send malicious requests to the network where the server was located, aiming to compromise the server's web environment and steal sensitive data. The other group acted as defenders, deploying and maintaining our detection model, and initiating defense measures once web attack traffic was detected.

Our model was deployed on a machine equipped with an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20 GHz and Ubuntu 18.04. During the exercise, the model monitored a total of 1946273 traffic data points, 464725 of which were malicious samples. Following our deployment scenario, we detected and analyzed this data, with the results presented in the accompanying table.

As widely recognized, real-world network traffic is considerably more complex than the datasets used in public benchmarks. Despite this complexity, our model achieves over 99% accuracy in detecting network attacks, as shown in Table VI, demonstrating the potential of Web-FTP for effective real-world deployment.

Furthermore, we tested the model's actual inference time. Table VII shows the designed inference time(ms) and minimum time for each batch of data of the deployed models. The model's inference speed varies when the batch size is different, but the number of flows that can be inferred stays around 1800-2000/s, which is the throughput of our model.

Our model is deployed on the QAX SkyEye New Generation Threat Sensitive System[2] (referred to as the SkyEye system). It processes about 800 sessions per second.[3] and accurately detects known and unknown advanced network attacks against hosts and servers in the network. Based on network traffic and terminal EDR logs, machine learning, deep learning, a rule engine, and other technologies are applied to achieve this. It also combines with other components to block threats promptly. A wide range

[2]Introduction to QAX SkyEye: https://en.qianxin.com/product/detail/118
[3]Introduction to Probe (Flow Sensor Hardware): http://ztrhmall.com/goods.php?id=129517

of industries, including banking, government, energy, education, and transport use our products for their cyberspace security.

VII. Conclusion

In this paper, we present Web-FTP, a feature transfer-based pre-trained model for web attack detection. The model is capable of learning generic feature representations of web attack traffic from large-scale unlabeled data through pre-training. By employing a modified tokenizer and feature transfer-based pre-training, the model's ability to capture traffic features is significantly enhanced, leading to improved classification accuracy across various publicly available encrypted traffic datasets and a real-world industrial platform. Both comparison and ablation experiments demonstrate that Web-FTP outperforms existing methods in classification tasks.

Looking ahead, we plan to conduct a comprehensive analysis of the multimodal features available in the fusion module of Web-FTP and further evaluate the model's effectiveness in defending against specific attack types.

Acknowledgment

The authors are grateful to anonymous reviewers for their fruitful comments, corrections, and inspiration to improve this paper.

References

[1] R. Komiya, I. Paik, and M. Hisada, "Classification of malicious web code by machine learning," in *Proc. 3rd Int. Conf. Awareness Sci. Technol.*, 2011, pp. 406–411.

[2] J. Li, H. Zhang, and Z. Wei, "The weighted Word2vec paragraph vectors for anomaly detection over HTTP traffic," *IEEE Access*, vol. 8, pp. 141 787–141 798, 2020.

[3] X. Liu and J. Liu, "Malicious traffic detection combined deep neural network with hierarchical attention mechanism," *Sci. Rep.*, vol. 11, no. 1, 2021, Art. no. 12363.

[4] X. Gong, J. Lu, Y. Zhou, H. Qiu, and R. He, "Model uncertainty based annotation error fixing for web attack detection," *J. Signal Process. Syst.*, vol. 93, pp. 187–199, 2021.

[5] J. Liang, W. Zhao, and W. Ye, "Anomaly-based web attack detection: A deep learning approach," in *Proc. 2017 VI Int. Conf. Netw. Commun. Comput.*, 2017, pp. 80–85.

[6] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 1963–1971, Mar. 2020.

[7] A. Tekerek, "A novel architecture for web-based attack detection using convolutional neural network," *Comput. Secur.*, vol. 100, 2021, Art. no. 102096.

[8] Y. E. Seyyar, A. G. Yavuz, and H. M. Ünver, "An attack detection framework based on BERT and deep learning," *IEEE Access*, vol. 10, pp. 68 633–68 644, 2022.

[9] L. S. R. Júnior, D. Macêdo, A. L. Oliveira, and C. Zanchettin, "LogBERT-BiLSTM: Detecting malicious web requests," in *Proc. Int. Conf. Artif. Neural Netw.*, Springer, 2022, pp. 704–715.

[10] B. G. Bokolo, L. Chen, and Q. Liu, "Detection of web-attack using DistilBERT, RNN, and LSTM," in *Proc. 11th Int. Symp. Digit. Forensics Secur.*, 2023, pp. 1–6.

[11] M. Gniewkowski, H. Maciejewski, T. R. Surmacz, and W. Walentynowicz, "HTTP2vec: Embedding of HTTP requests for detection of anomalous traffic," 2021, *arXiv:2108.01763*.

[12] M. Gniewkowski, H. Maciejewski, T. Surmacz, and W. Walentynowicz, "Sec2vec: Anomaly detection in HTTP traffic and malicious URLs," in *Proc. 38th ACM/SIGAPP Symp. Appl. Comput.*, 2023, pp. 1154–1162.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv: 1810.04805*.

[14] Y. Zhu, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," 2015, *arXiv:1506.06724*.

[15] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proc. ACM Web Conf.*, 2022, pp. 633–642.

[16] I. Sharafaldin et al., "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[17] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related," in *Proc. 2nd Int. Conf. Inf. Syst. Secur. Privacy*, 2016, pp. 407–414.

[18] R. Panigrahi and S. Borah, "A detailed analysis of CICIDS2017 dataset for designing intrusion detection systems," *Int. J. Eng. Technol.*, vol. 7, no. 3.24, pp. 479–482, 2018.

[19] H. Nguyen, C. Torrano-Gimenez, G. Álvarez, S. V. Petrovic, and K. Franke, "Application of the generic feature selection measure in detection of web attacks," in *Proc. Int. Conf. Comput. Intell. Secur. Inf. Syst.*, 2011, pp. 25–32.

[20] C. Raïssi, J. Brissaud, G. Dray, P. Poncelet, M. Roche, and M. Teisseire, "Web analyzing traffic challenge: Description and results," in *Proc. ECML/PKDD*, 2007, Art. no. 6.

[21] T. S. Riera, J. R. B. Higuera, J. B. Higuera, J. Martínez-Herráiz, and J. A. S. Montalvo, "A new multi-label dataset for web attacks CAPEC classification using machine learning techniques," *Comput. Secur.*, vol. 120, 2022, Art. no. 102788.

[22] Fwaf machine learning-driven web application firewall, 2017. [Online]. Available: https://github.com/faizann24/Fwaf-Machine-Learning-driven-Web-Application-Firewall/

[23] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," 2015, *arXiv:1508.07909*.

[24] Z. Yao, D. S. Weld, W.-P. Chen, and H. Sun, "StaQC: A systematically mined question-code dataset from stack overflow," in *Proc. 2018 World Wide Web Conf.*, 2018, pp. 1693–1703.

[25] M. Raghothaman, Y. Wei, and Y. Hamadi, "SWIM: Synthesizing what i mean: Code search and idiomatic snippet synthesis," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 357–367.

[26] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "CodeSearchNet challenge: Evaluating the state of semantic code search," 2019, *arXiv: 1909.09436*.

[27] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation with hybrid lexical and syntactical information," *Empirical Softw. Eng.*, vol. 25, pp. 2179–2217, 2020.

[28] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 933–944.

[29] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "When deep learning met code search," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 964–974.

[30] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. 2020 Conf. Empirical Methods Natural Lang. Process. Syst. Demonstrations*, 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[31] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv: 1907.11692*.

[32] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training text encoders as discriminators rather than generators," 2020, *arXiv: 2003.10555*.

[33] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.

[34] R. Chiramdasu, G. Srivastava, S. Bhattacharya, M. S. K. Reddy, and T. Gadekallu, "Malicious URL detection using logistic regression," in *Proc. 2021 IEEE Int. Conf. Omni-Layer Intell. Syst.*, 2021, pp. 1–6.

[35] Y. Ding, L. Wang, H. Zhang, J. Yi, D. Fan, and B. Gong, "Defending against adversarial attacks using random forest," in *Proc. 2019 IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops*, 2019, pp. 105–114.

[36] A. Shahraki, M. Abbasi, and Ø. Haugen, "Boosting algorithms for network intrusion detection: A comparative evaluation of real AdaBoost, Gentle AdaBoost and modest AdaBoost," *Eng. Appl. Artif. Intell.*, vol. 94, 2020, Art. no. 103770.

[37] C.-T. Yang, Y.-W. Chan, J.-C. Liu, E. Kristiani, and C.-H. Lai, "Cyberattacks detection and analysis in a network log system using XGboost with ELK Stack," *Soft Comput.*, vol. 26, pp. 5143–5157, 2021.

[38] T. Heitz and M. Roche, "Analyzing web traffic ECML/PKDD 2007 discovery challenge," 2007. [Online]. Available: https://www.lirmm.fr/pkdd2007-challenge/

Zhenyu Guo is currently working toward the PhD degree with the School of Computer Science, Beijing Institute of Technology, Beijing, China. His current research interests include the development of algorithms for deep learning and their application to cyberspace security.

Qinghua Shang is currently working with QI-ANXIN Technology Group Inc., Beijing, China. Her current research interests include natural language understanding.

Xin Li received the PhD degree in computer science from Hong Kong Baptist University, in 2009. She is currently an associate professor with the School of Computer Science, Beijing Institute of Technology, China. Her research focuses on the development of algorithms for representation learning, reasoning under uncertainty, and machine learning with applications to graph data mining, recommender systems, and robotics.

Chengyi Li is currently working toward the ME degree with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. His current research interests include deep learning and network anomaly detection.

Zijian Zhang (Senior Member, IEEE) received the PhD degree from the School of Computer Science and Technology, Beijing Institute of Technology. He is now a professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology. He was a visiting scholar with the Computer Science and Engineering Department, State University of New York at Buffalo in 2015. His research interests include design of authentication and key agreement protocol and analysis of entity behavior and preference.

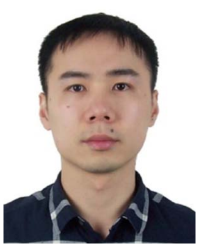
Zhuo Zhang is currently the vice president with Qi An Xin Technology Group. With more than 13 years of experience in research and development, architecture, product, and project management in the cybersecurity industry. His research interests include offensive and defensive confrontation, advanced sustainability threat (APT) attack detection, and deep learning.

Jingjing Hu received the PhD degree in computer science from the Beijing Institute of Technology, Beijing, China. She is currently an associate professor with the School of Computer Science, Beijing Institute of Technology. Her research interests include the areas of service computing, web intelligence, and information security.

Jincheng An is currently an assistant research fellow with Qi An Xin Technology Group Inc. He is an expert in cybersecurity and technology standardizing document drafting. His research interests include cyber resilience theory and evaluation, and data security detection.

Chuanming Huang employ with QI-ANXIN Technology Group Inc. His responsibilities include network security products and artificial intelligence research, with a focus on the integration of large language models and network security.

Yang Chen is currently working with QI-ANXIN Technology Group Inc., Beijing, where he primarily engages in work related to deep learning, machine learning, and Big Data analysis and processing.


Yuguang Cai is the general manager with QI-ANXIN SkyEYE series products since October 2021, leading NDR and XDR teams. His current research interests include incident response and adversary-based threats research to Qi-An-Xin and its customers.