四川大學

课程实践报告

课　　程　网络空间安全数据挖掘技术(314067030)

评　　分 _____

姓　　名 _____ 学号 _____

姓　　名 _____ 学号 _____

目录

基于 LSTM 的字符级恶意 Web 请求检测系统

网络空间安全 专业

[摘要] 如今，在网络空间安全领域，Web 参数注入攻击十分常见，其对 Web 应用程序的安全性构成了极大的威胁。在这种攻击中，攻击者可以通过向 HTTP 请求的参数中注入一些恶意代码来利用 HTTP 请求对服务器实施攻击。针对 Web 参数注入攻击，目前的大多数 Web 入侵检测系统（WIDS）由于缺乏重新学习的能力，并且很少关注字符间的内在关系，因此无法发现未知的新攻击，且具有较高的误报率。本文提出了一种基于 LSTM 的字符级恶意 Web 请求检测系统。我们在卷积层之前添加了字符级嵌入层，这使得我们的模型能够学习请求参数的字符之间的内在关系。此外，我们修改了 LSTM 的过滤器，使其可以提取请求参数的细粒度特征。同时，我们实现了一个基于 flask 架构的查询页面，提供了友好的用户界面。实验结果表明，我们的系统具有较低的误报率和良好的性能。

关键词：网络空间安全 恶意 Web 请求检测 LSTM 字符级

1 选题背景

如今，大量的网络通信是通过 HTTP/HTTPS 等协议进行的，客户端可以发起 Web 请求并将其发送至 Web 服务器。 Web 请求通常会带有用户输入的一些参数。图 1 显示了 HTTP 协议的 GET 请求中使用的参数。然而，这种普通的协议或方法很可能会带来安全风险，许多攻击者可以通过发送 Web 请求将恶意代码传递给 Web 服务器。

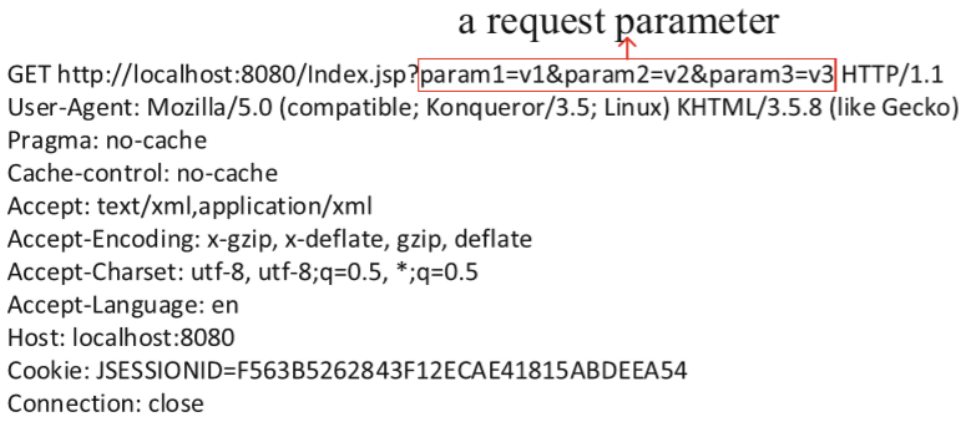

图 1 HTTP 协议的 GET 请求

针对 Web 参数注入攻击的现有对策通常称为 Web 入侵检测系统（WIDS），主要包括两种实现方式：基于签名的检测和基于异常的检测。在基于签名的检测中，服务器需要维护一个恶意符号库以区分注入攻击与正常的 Web 请求。收到请求后，服务器会在其中搜索请求参数，查看它们是否包含恶意符号，并将包含恶意符号的 Web 请求视为攻击。基于签名的检测通常在包含不同符号的已知攻击中表现良好。但是，若攻击者对注入的代码进行微小的更改，那么恶意符号将会被轻易地隐藏，因此这种检测方式无法找到未知的攻击。对于基于异常的检测，服务器需要训练一个数学模型，该模型可以表征正常的 Web 请求并过滤大多数异常的 Web 请求。与基于签名的检测相比，这种类型的 Web 入侵检测系统能够发现一些未知的攻击。不幸的是，基于异常检测的系统 FPR（反正例率）较高。

除了高 FPR 之外，基于异常的检测模型还具有重新学习的能力。实际上，早期的基于异常的检测系统会一次性收集所有训练数据，然后生成一个常数模型。早期的基于异常的检测系统将具有异常数据结构的样本视为攻击样本。因此，它

可以检测到一些未知的攻击。但是，当攻击的数据结构与正常请求的数据结构相似时，由于这种模型无法及时进行更新，它将无法发现此类攻击。

2 团队分工

3 检测模型

由于 LSTM 在文本分类问题中的表现非常出色，因此我们建立了一个 LSTM 模型来解决恶意 Web 请求检测问题。我们使用字符嵌入层，并使模型自行学习以根据其任务数字化字符。然后，使用 LSTM 模型提取请求参数的特征，并通过输出层给出检测结果。

3.1 字符表示

Web 请求参数是一个字符序列，它包含三种类型的字符，：字母、数字和特殊字符，如图 2，总共有 95 个字符。我们通过索引这些字符将字符序列转换为索引列表。但是，字符索引不能反映特定上下文中字符之间的关系。例如，普通文本中的字符“ =”与字符“＆”没有关联。如图 3 所示，在 Web 请求参数的上下文中，它们起着分隔符的作用。这种相似性不能用索引号表示。

Lowercase letters	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Uppercase letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Digital	0	1	2	3	4	5	6	7	8	9																
Special characters	;	,		:	/	=	%	&	@	!	*	(	)	{	}	[	]	>	<	?	‘	\	.	_	|	+
	-	`	“	#	$	~	^																			

图 2 字符列表

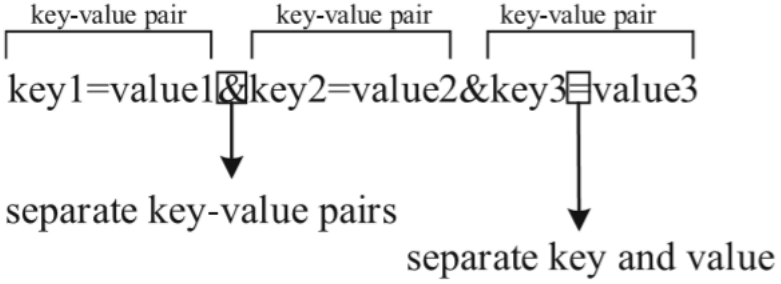

图 3 请求参数中的“=”和“&”

关于此问题，我们在模型中使用字符级嵌入来了解 Web 请求参数中字符之间的隐藏相似性。字符级嵌入的思想来自 NLP 中使用的词嵌入。

3.2 模型结构

图 4 所示的模型架构是我们在演示系统中使用的深度神经网络。它主要由四个部分组成：嵌入层，卷积层，池化层和输出层。

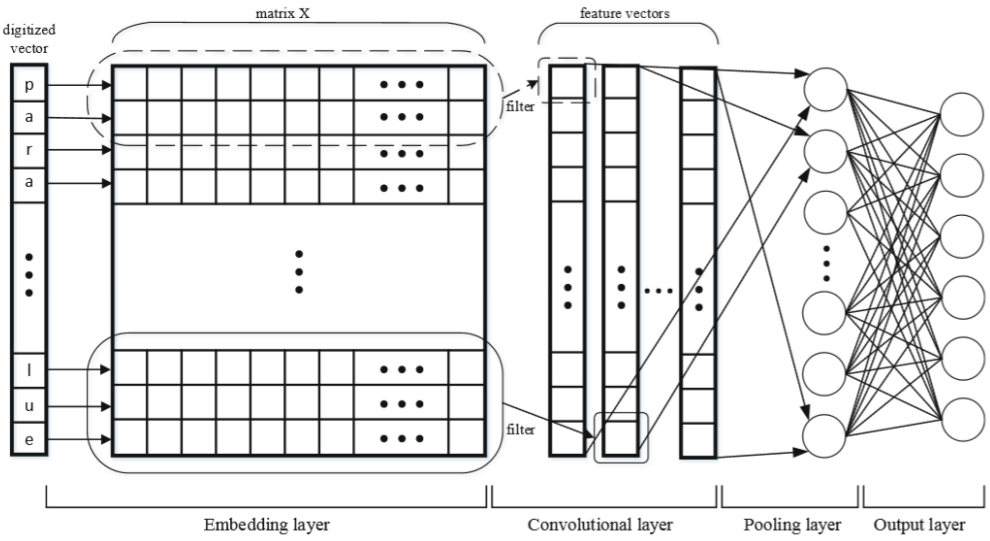

图 4 模型结构

4 技术实现

4.1 数据集描述

项目使用 HTTP 数据集 CSIC 2010。 CSIC 2010 包含数千个 Web 请求，这些请求是通过产生对电子商务 Web 应用程序的流量而自动生成的。 数据集包含 36,000 个标记为正常的 HTTP 请求和大约 25,000 个标记为异常的请求。 攻击类型包括 SQL 注入，缓冲区溢出，信息收集，XSS 等。 根据神经网络模型对数据集进行预处理。 HTTP 请求中包含的某些功能（这些功能不能帮助区分正常请求和异常请求）已删除，例如协议，userAgent，pragma，cacheControl，accept，acceptEncoding，acceptCharset，acceptLanguage，连接等功能。

4.2 环境配置

python == 3.7.4

tensorflow==1.3.0

keras==2.3.1

numpy==1.17.1

4.3 项目目录

```
├─── DataPreprocessing.py #数据预处理文件
├─── README.md
├─── SourceCode.py #模型训练及测试
├─── anomalousTraffic.txt #异常流量数据
├─── anomalous_parsed.txt #异常流量解析后数据
├─── normalTraffic.txt #正常流量数据
└─── normal_parsed.txt #正常流量解析后数据
```

4.4 数据预处理

分别读取 CSIC 2010 数据集的异常流量数据包和正常流量数据包，解析 GET 或 POST 请求所在行：首先将获取 GET 或 POST 方法，如果是 GET 方法直接追加 url，如果是 POST 或 PUT 方法获取数据包长度后追加字符串。

数据处理前后对比：

```
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+W
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=B92A8B48B9008CD29F622A994E0F650D
Connection: close


POST http://localhost:8080/tienda1/publico/anadir.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,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=AE29AEEBDE479D5E1A18B4108C8E3CE0
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 146
```

图 5 原始数据

图 6 处理后数据

实现代码：

```
line = lines[i].strip()
        if line.startswith("GET"):
            res.append("GET" + line.split(" ")[1])
        elif line.startswith("POST") or line.startswith("PUT"):
            url = line.split(' ')[0] + line.split(' ')[1]

            j = 1
            while True:
                if lines[i + j].startswith("Content-Length"):
                    break
                j += 1
            j += 1
            data = lines[i + j + 1].strip()
            url += '?' + data
            res.append(url)
```

之后对 url 进行解码并转为小写后分别存入 normal_parsed.txt、anomalous_parsed.txt 文件。

首先读入预处理后的数据，将数据按字符为单位分隔开并转换为对应的字符索引值。

```
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(x)

sequences = tokenizer.texts_to_sequences(x)
char_index = tokenizer.word_index
```

字符索引列表如下图所示：

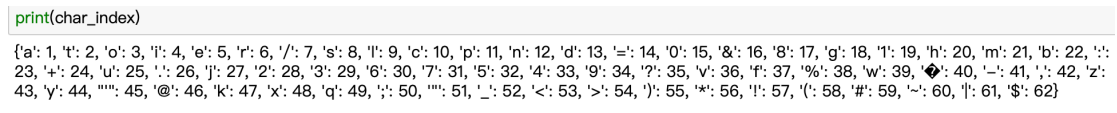

图 7 字符索引列表

4.5 模型实现

定义模型输入向量维度为最大值 1000，输出向量维度为 32，依次添加嵌入层、LSTM 层、Dropout 层和全连接层。设置优化函数为 adam，损失函数为交叉熵，模型结构如图 8 所示。

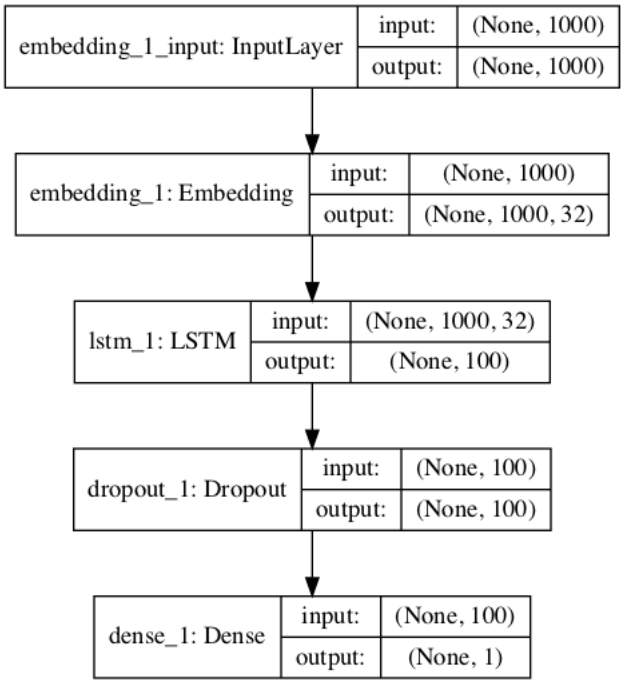

图 8 模型结构

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 32)          2016
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 55,317
Trainable params: 55,317
Non-trainable params: 0
```

图 9 模型各层参数状况

实现代码：

```
1.  embedding_dim = 32
2.  max_chars = 63
3.
4.  def build_model():
5.    model = models.Sequential()
```

```
    model.add(layers.Embedding(max_chars, embedding_dim, input_length=maxlen))
    model.add(layers.LSTM(100))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam',  loss='binary_crossentropy', metrics=['accuracy'])
    return model

model = build_model()
```

4.6 模型训练

按 1：4 划分训练集和测试集，设置批处理大小为 32，将模型训练十轮，准确率达到 97.7%

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=21)
model.fit(x_train, y_train, epochs=10, batch_size=32)
test_acc, test_loss = model.evaluate(x_test, y_test)
```

```
11936/12213 [============================>.] - ETA: 1s
11968/12213 [============================>.] - ETA: 0s
12000/12213 [============================>.] - ETA: 0s
12032/12213 [============================>.] - ETA: 0s
12064/12213 [============================>.] - ETA: 0s
12096/12213 [============================>.] - ETA: 0s
12128/12213 [============================>.] - ETA: 0s
12160/12213 [============================>.] - ETA: 0s
12192/12213 [============================>.] - ETA: 0s
12213/12213 [==============================] - 49s 4ms/step
0.06931999455083551 0.977073609828949
```

图 10 训练结果

4.7 可视化界面

本项目实现了一个基于 flask 架构的查询页面，如图 11 所示，使得该系统具备更强的实用性。

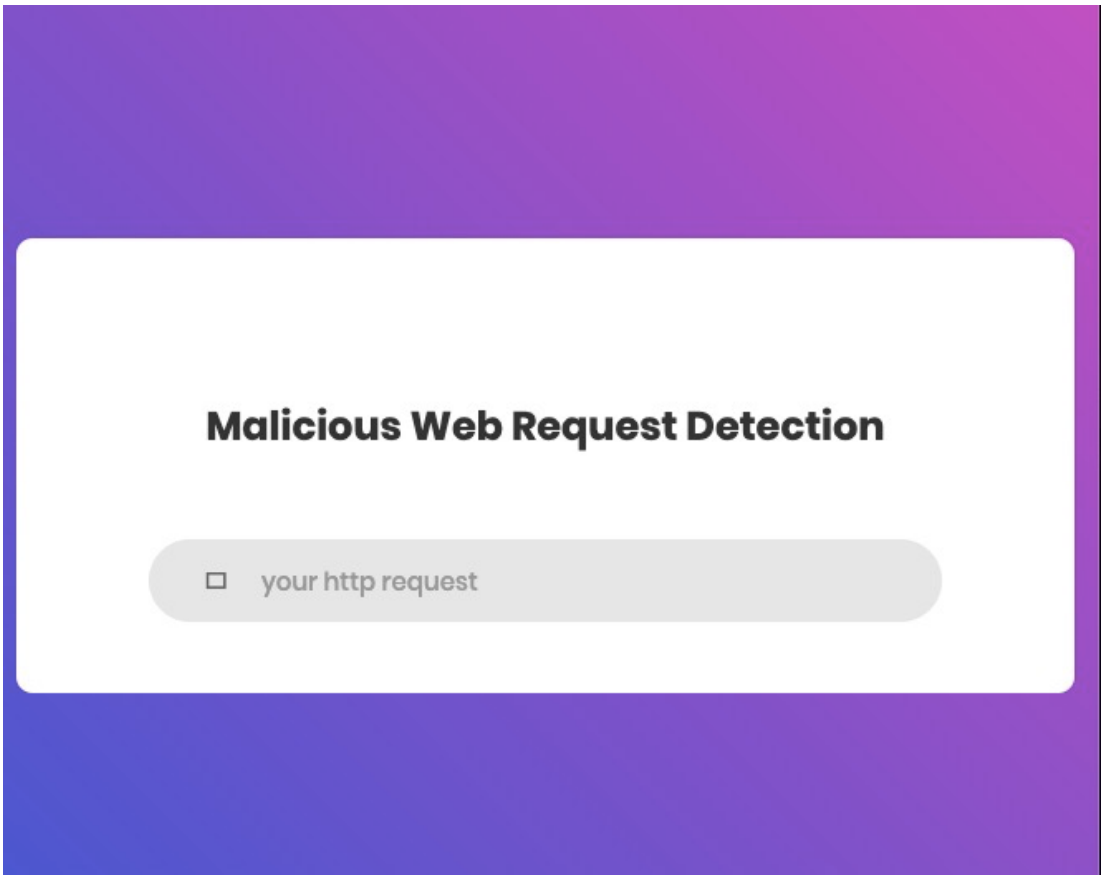

图 11 可视化界面

5 测试及评估

我们使用图 4 所示的模型实现了一个用于恶意 Web 请求检测的分类器。在训练过程中，我们打乱所有训练样本并将其分为几批，然后使用每个批次的样本来优化每个图层的嵌入表和权重。使用一批样本来优化这些参数的过程是其中一个步骤。因此，我们可以逐步优化所有批次的这些参数。图 12 显示了我们的模型逐步优化的过程，并展示了 softmax 交叉熵和 L2 正则化的总和。

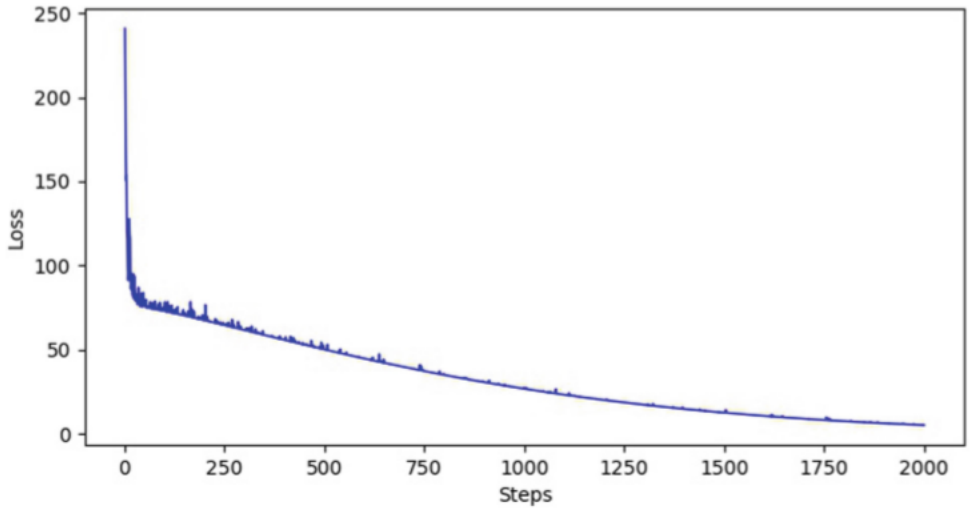

图 12 训练过程

6 未来工作

未来，我们将实现一个基于 LSTM 的字符级恶意 Web 请求检测系统，系统架构如图 13 所示。

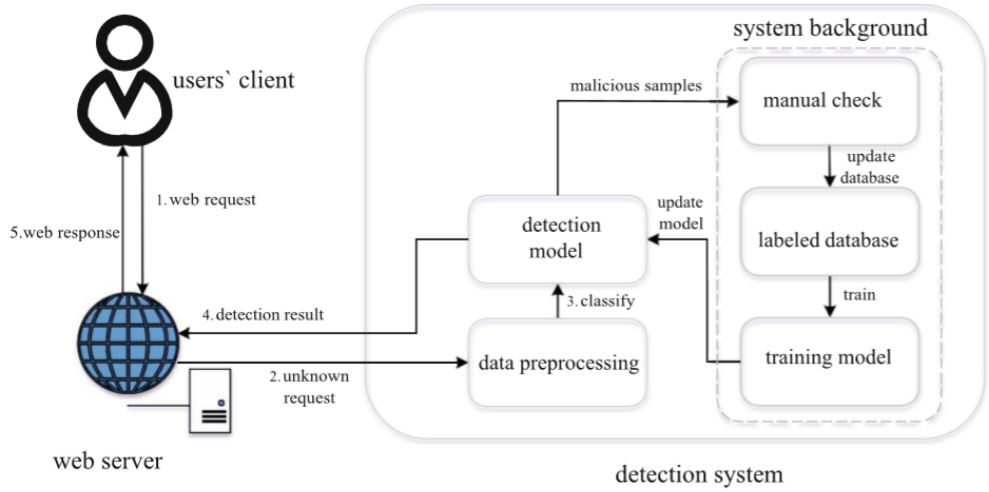

图 13 系统架构

系统工作流程如下：

（1）客户端将带有某些参数的 Web 请求发送到 Web 服务器。

（2）Web 服务器将请求传递到数据预处理模块。数据预处理模块从请求中提取请求参数，并通过索引这些字符将其转换为索引列表。

（3）数据预处理模块将此索引列表反馈给检测模型。

（4）经过对该向量的复杂计算后，检测模型将检测结果返回到 Web 服务器。

（5）Web 服务器根据检测结果对客户端进行响应。如果这是一个正常请求，则 Web 服务器将返回所需的 Web 响应。否则，Web 服务器将返回错误。

在系统背景下，分类模型会保存一些样本及其检测结果。从检测模型不太可靠（概率相对较低）的那些样本中随机选择这些保存的样本。为了确保标记的数据具有代表性并平衡每种数据的数量，我们将在将样本添加到标记的数据库之前手动检查样本。当系统发现标签数据库已更改时，它将使用更改后的数据库在旧模型的基础上训练新的检测模型。然后，系统使用新模型更新其检测模型。这样，模型可以获得重新学习的能力并适应网络攻击的变化。

7 总结

本项目中，我们提出了一种基于 LSTM 的字符级恶意 Web 请求检测模型。与传统的检测模型相比，我们的模型更加关注请求参数中字符间的内在关系，我们在卷积层之前添加了字符级嵌入层，并修改了 LSTM 过滤器以提取请求参数中的局部特征。实验结果表明，我们的模型具有较低的误报率和良好的性能。

参考文献

[1] Pan Y, Sun F, Teng Z, et al. Detecting web attacks with end-to-end deep learning[J]. Journal of Internet Services and Applications, 2019, 10(1): 1-22.

[2] Dong Y, Zhang Y, Ma H, et al. An adaptive system for detecting malicious queries in web attacks[J]. Science China Information Sciences, 2018, 61(3): 032114.

[3] Wang J, Zhou Z, Chen J. Evaluating CNN and LSTM for Web Attack Detection[C]//Proceedings of the 2018 10th International Conference on Machine Learning and Computing. ACM, 2018: 283-287.

[4] Zhang M, Xu B, Bai S, et al. A deep learning method to detect web attacks using a specially designed CNN[C]//International Conference on Neural Information Processing. Springer, Cham, 2017: 828-836.

[5] Dong Y, Zhang Y, Ma H, et al. An adaptive system for detecting malicious queries in web attacks[J]. Science China Information Sciences, 2018, 61(3): 032114.

[6] Kejriwal N G. Method for detecting malicious javascript: U.S. Patent Application 12/849,721[P]. 2011-2-3.