

## 基于多特征融合的 Webshell 恶意流量检测方法

李源, 王运鹏, 李涛, 马宝强

(四川大学网络空间安全学院, 四川 成都 610065)

**摘要:** Webshell 是针对 Web 应用系统进行持久化控制的最常用恶意后门程序, 对 Web 服务器安全运行造成巨大威胁。对于 Webshell 检测的方法大多通过对整个请求包数据进行训练, 该方法对网页型 Webshell 识别效果较差, 且模型训练效率较低。针对上述问题, 提出了一种基于多特征融合的 Webshell 恶意流量检测方法, 该方法以 Webshell 的数据包元信息、数据包载荷内容以及流量访问行为 3 个维度信息为特征, 结合领域知识, 从 3 个不同维度对数据流中的请求和响应包进行特征提取; 并对提取特征进行信息融合, 形成可以在不同攻击类型进行检测的判别模型。实验结果表明, 与以往研究方法相比, 所提方法在正常、恶意流量的二分类上精确率得到较大提升, 可达 99.25%; 训练效率和检测效率也得到了显著提升, 训练时间和检测时间分别下降 95.73% 和 86.14%。

**关键词:** 多特征; 特征融合; Webshell 检测; 集成学习

**中图分类号:** TP393

**文献标识码:** A

**DOI:** 10.11959/j.issn.2096-109x.2021103

## Webshell malicious traffic detection method based on multi-feature fusion

LI Yuan, WANG Yunpeng, LI Tao, MA Baoqiang

School of Cyber Science and Engineering, Sichuan University, Chengdu 610065, China

**Abstract:** Webshell is the most common malicious backdoor program for persistent control of Web application systems, which poses a huge threat to the safe operation of Web servers. For most Webshell detection method based on

收稿日期: 2021-05-18; 修回日期: 2021-10-18

通信作者: 王运鹏, [wyp@scu.edu.cn](mailto:wyp@scu.edu.cn)

**基金项目:** 国家重点研发计划 (2020YFB1805400); 国家自然科学基金 (U1736212, U19A2068, 62002248, 62032002); 中国博士后科学基金 (2019TQ0217, 2020M673277); 中央高校基本科研业务经费 (YJ201933); 四川省重点研发 (20ZDYF3145)

**Foundation Items:** The National Key R&D Program of China (2020YFB1805400), The National Natural Science Foundation of China (U1736212, U19A2068, 62002248, 62032002), The China Postdoctoral Science Foundation (2019TQ0217, 2020M673277), The Fundamental Research Funds for the Central Universities (YJ201933), The Provincial Key Research and Development Program of Sichuan (20ZDYF3145)

**论文引用格式:** 李源, 王运鹏, 李涛, 等. 基于多特征融合的 Webshell 恶意流量检测方法[J]. 网络与信息安全学报, 2021, 7(6): 143-154.

LI Y, WANG Y P, LI T, et al. Webshell malicious traffic detection method based on multi-feature fusion[J]. Chinese Journal of Network and Information Security, 2021, 7(6): 143-154.

the request packet data for training, the method for web-based Webshell recognition effect is poorer, and the model of training efficiency is low. In response to the above problems, a Webshell malicious traffic detection method based on multi-feature fusion was proposed. The method was characterized by the three dimensions of Webshell packet meta information, packet payload content and traffic access behavior. Combining domain knowledge, feature extraction of request and response packets in the data stream. Transformed into feature extraction information for information fusion, forming a discriminant model that could detect different types of attacks. Compared with the previous research method, the accuracy rate of the method here in the two classification of normal and malicious traffic has been improved to 99.25%. The training efficiency and detection efficiency have also been significantly improved, and the training time and detection time have been reduced by 95.73% and 86.14%.

**Keywords:** multi-feature, feature fusion, Webshell detection, ensemble learning

## 1 引言

随着 Web 应用的爆炸式增长, Web 应用系统遭受攻击的事件越来越多, 带来的损失和影响也越来越大。据中国产业互联网发展联盟公布的《2019 中国主机安全服务报告》<sup>[1]</sup>显示, 2019 年中国境内企业用户服务器感染病毒木马事件超过百万起。其中 Webshell 恶意程序感染事件占 73.27%。从攻击者的角度看, 植入 Webshell 是攻击者进行持续渗透的基础和前提。因此, 为了保障 Web 服务器的安全, Webshell 检测显得尤为重要。Apache 是目前应用最普遍的 Web 应用系统<sup>[2]</sup>, 在 Web 应用方面具有最为广泛的代表性, 因此本文以 Apache 服务器上的 Webshell 检测为研究和实验对象。

针对 Webshell 的检测主要有 3 个研究方向: 基于文本的静态检测方法、基于 Web 日志的事后检测以及基于通信流量的检测<sup>[3]</sup>。在基于文本的静态 Webshell 检测研究中, 研究者主要从代码层面上对恶意程序进行检测, 常用的检测方法是检查程序中是否包含 Webshell 执行时所需的系统函数<sup>[4]</sup>, 此方法检测效率高, 但难以识别经过混淆加密后的 Webshell。为了对抗混淆的 Webshell 恶意代码, Ai 等<sup>[5]</sup>提出使用 PHP VLD 插件获得 PHP 程序编译时产生的字节码, 并经 N-gram 分词后生成的执行序列作为区分正常文件和 Webshell 文件的检测特征, 此模型对基于 PHP 语言的 Webshell 有较好的检测效果, 但对于其他语言编写的 Webshell, 不能提取其编译时的字节码, 因此无法检测。崔艳鹏等<sup>[6]</sup>提出使用信息熵、最长单词、重合指数、压缩比等统计特征作为区分正常和恶意程序的特征, 该方法能够在一定程度上

识别经过加密混淆后的 Webshell 文件, 但通常需要配合其他方法一同检测以防止产生误报。Li 等<sup>[7]</sup>提出使用神经网络模型 GRU 代替人工提取特征, 但该方法在对文件进行分词时并未考虑代码经过编后(如 base64 编码)仍可能出现词向量维度过大导致训练难度增加、效率降低的问题; 基于 Web 日志的事后检测是在攻击发生后通过对 Web 应用运行日志进行分析、溯源, 以检测 Webshell 通信行为<sup>[8]</sup>, 但由于 Web 日志本身属于格式化处理后的文件, 信息丢失较多, 因而检测效果有限; 在基于通信流量的检测研究上, 研究者常采用对实时流量进行捕获分析来挖掘通信过程中可能存在的 Webshell 恶意流量。2018 年, Yang 等<sup>[9]</sup>采用基于 4-gram 分词的 SVM 算法对数据包中的 URL 请求路径和请求体进行训练以识别数据包中的恶意流量; Zhang 等<sup>[10]</sup>选择使用 CNN+LSTM 神经网络组合模型对数据包整个请求包载荷的 ASCII 序列进行训练生成分类模型, 以实现 Webshell 恶意流量的检测, 而由于在训练过程中加入过多与流量识别无关的信息, 因此检测效率不高。此外, 这两种基于通信流量的检测方法均只能完成对正常或恶意流量的二分类检测, 未能对 Webshell 恶意流量属于何种攻击类型做出判断, 不利于有针对性的 Web 防御和加固。

总的来说, 以上 3 种检测方法, 难以高效、精确地对 Webshell 进行检测和分类, 并不能有效阻止通信过程中 Webshell 行为所造成的危害。因此本文从 Web 通信流量出发, 以多源视角提取不同维度的静态特征, 并通过特征融合和算法集成的方式来挖掘网络流量中可能存在的 Webshell 恶意通信行为。主要贡献如下。

1) 提出了一种基于多特征融合的检测方法, 融合了不同维度的数据流特征, 弥补了单一特征识别能力不足的问题, 能够在不同类型的 Webshell 的识别上都有较高的检测能力, 进一步细化不同种类的 Webshell 攻击。

2) 提出基于集成学习的 Webshell 分类模型, 通过结合不同学习器各自的优势生成分类模型, 降低模型在 Webshell 多分类情况下出现误报的概率。

## 2 多特征融合检测方法

### 2.1 检测框架

Webshell 恶意流量检测模型的总体框架如图 1 所示, 包括数据预处理、特征提取和异常检测 3 个阶段。

#### (1) 数据预处理

对采集到的原始流量进行解析后依次过滤非正常响应、媒体资源文件的数据包, 并对通信过程中产生的分段数据包按五元组<源 IP, 源端口, 目的 IP, 目的端口, 协议>将相同的源和目的 IP 地址合并为一次的完整通信会话, 并生成文本向量用于特征提取。

#### (2) 特征提取

对原始流量进行预处理得到的文本向量, 从数据包元信息、数据包载荷内容以及流量访问行为 3 个维度提取特征, 并将融合后的特征向量作为特征数据集。

#### (3) 异常检测

将融合后的特征数据集输入至包含 4 个初级分类器和 1 个次级分类器的 Stacking 集成模型并

保存模型参数, 通过模型对未知流量进行识别分类, 以实现具体攻击类型的分类。

### 2.2 数据预处理

数据预处理模块主要由数据包解析、数据包分段重组、数据包解码以及数据包过滤 4 部分组成。数据包解析主要完成对捕获的二进制流量按网络协议进行逐层解析, 生成可读的网络封包的详细信息以供特征提取阶段使用; 数据包分段重组主要完成对来自同一个数据包产生的分段数据包的重组; 数据包解码主要完成对编码后的数据进行数据还原; 数据包过滤主要采用文件头标识来处理数据流中的多媒体资源文件, 以避免其中的不可见字符对模型训练造成干扰。

#### 2.2.1 数据包解析

本文主要使用基于 Python 编程语言的数据包解析工具 Scapy 库<sup>[11]</sup>对采集的网络流量进行处理。Scapy 库是一个功能强大的数据包处理工具, 利用该工具可以轻松实现发送、嗅探、解析和伪造网络数据包。本文使用 Scapy 解析并提取数据包字段(时间戳、请求类型、确认号、序列号、源 IP 地址、目的 IP 地址、有效载荷、数据包大小等)。对取值为离散值的字段需要对其进行编码, 如请求类型字段为: GET、POST 等, 则对应编码为 GET=1, POST=2。

#### 2.2.2 数据包分段重组

当发送端发送的数据包长度大于通信双方协商的每一个报文段所能承载的最大数据长度 MSS (maximum segment size) 时, 需要对数据包进行分段传输, 产生多个具有完整首部的 TCP 报

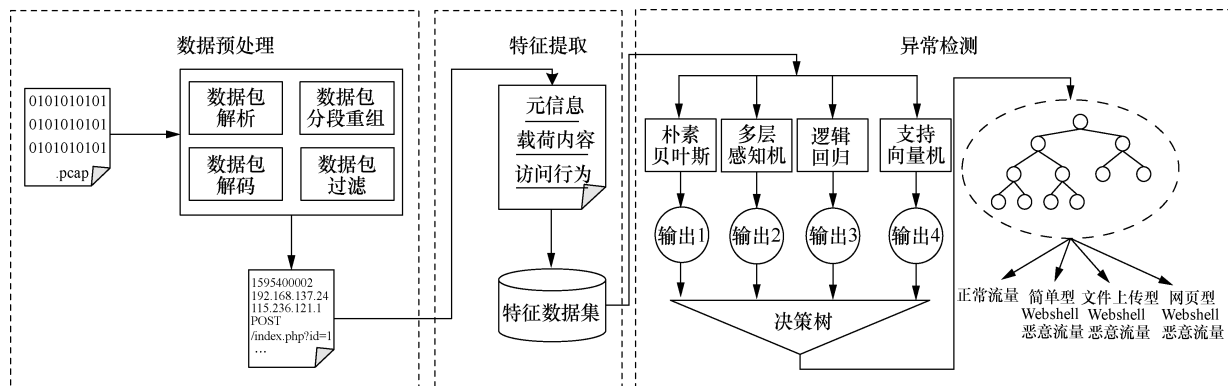


图1 Webshell 恶意流量检测模型的总体框架

Figure 1 Webshell malicious traffic detection model framework

文段。为了在接收端将来自同一个数据包产生的多个报文段进行重组还原,本文首先将捕获到待重组报文段按照序列号由小到大排列得到 $[seqk_1, seqk_2, \dots, seqk_n]$ 。对于序列号为 $seq_i$ 的报文段,其 TCP 长度为 $TCP - payload_i$ ,则下一个分片序列号为 $seq_i + TCP - payload_i$ 。以此类推,可将所有来自相同数据包产生的报文段按顺序拼接完成。

### 2.2.3 数据包解码

当发送端发送请求包中的请求地址字段包含特殊字符时,根据 RFC 3986 协议<sup>[12]</sup>规定,需要在数据传输前对其进行编码,具体操作为使用百分号和对应的 ASCII 值进行替换。因此,对于经过编码后的请求地址字段内容,如:“/index?id=shop%23name”,需要进行解码得到原始请求字段内容为“index?id=shop#name”。

### 2.2.4 数据包过滤

为了缩短模型在预处理阶段所花的时间,同时避免数据流中的不可见字符影响模型训练效果,需要对采集到的网络流量依据数据类型进行过滤,其中一部分是页面加载时的静态资源文件(如 html、js、css)等,另一部分主要是图片、音视频等非文本类型的数据。针对这两部分数据,在数据包过滤时主要根据数据包中的请求资源定位符来过滤掉非 PHP 文件。但已有研究表明,存在将 Webshell 藏在图片的末尾形成“图片马”以绕过目标服务器防护设备检查的攻击手段,因此对于 PNG、JPEG 等格式的图片文件,主要通过文件头尾标识符定位来处理。文件头标识符是以二进制形式存于文件开头一段区域的数据,该数据对应每一种文件类型。常见的多媒体资源文件同样拥有独一无二的文件头信息,如 PNG 格式图片的文件头尾标识符分别为 0x89 50 4E 47 0D 0A 1A 0A 和 0x49 45 4E 44 AE 42 60 82。因此,通过识别数据流中的文件标识符,进而定位到多媒体资源文件的首尾并对其中部分进行过滤,即可提取出插入文件末尾的 Webshell,处理示意如图 2 所示。

## 2.3 特征提取

为了对不同类型的 Webshell 恶意流量进行准确识别,需要提取有关流量的各方面特征,并将

其序列化表示输入分类器中。按照业界对 Webshell 恶意程序的功能以及攻击影响进行划分,可分为 3 种类型。

| 0000028b | 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f |                         |
|----------|-------------------------------------------------|-------------------------|
| 00000000 | 89 50 4e 47 0d 0a 1a 0a                         | 00 00 00 0d 49 48 44 52 |
| 00000010 | 00 00 03 20 00 00 02 58                         | 04 03 00 00 00 68 58 9f |
| 00000020 | 43 00 00 00 1e 50 4c 54                         | 45 00 00 00 ff ff ff 66 |
| 00000030 | 66 66 5a 5a 5a 2c 2c 2c                         | 0a 0a 0a d3 d3 d3 60 60 |
| 00000040 | 60 15 15 15 9f 9f 9f a9                         | 39 39 b5 00 00 08 d4 49 |
| 00000050 | 44 41 54 78 da ec d4 b1                         | 6d 14 00 10 05 d1 0b 96 |
| 00000060 | d2 be a8 6e c3 15 d4 fc                         | f2 11 18 e3 73 e8 e8 c6 |
| 00000070 | d2 bc 16 46 9a 07 50 7e                         | 3c 44 32 bf 1e 22 e9 cf |
| 00000080 | 87 40 36 06 21 99 33 08                         | 4a 63 10 92 8d 41 48 e6 |
| 00000090 | 0c 82 d2 16 84 64 63 10                         | 92 39 83 a0 34 06 21 d9 |
| 000000a0 | 18 84 64 c0 20 28 8d 41                         | 48 36 06 21 99 33 08 4a |
| 000000b0 | 64 f8 76 0b 88 26 2c 0b                         | 88 26 2c 09 88 27 2c 07 |
| 000000c0 | 88 28 2c 07 88 28 2c 05                         | 88 29 2c 03 88 2a 2c 03 |
| 000000d0 | 2c 36 29 2c 0c a4 b0 2c                         | 90 c2 b2 40 0a 0b 03 29 |
| 000000e0 | 2c 0b a4 b0 2c 90 c2 c2                         | 40 0a cb 02 29 2c 0b a4 |
| 000000f0 | b0 30 90 c2 b2 40 0a cb                         | 02 29 2c 0c a4 b0 2c 90 |
| 00000100 | c2 b2 5e 65 08 b0 7e 8b                         | 80 d5 6a c0 6a 35 60 b5 |
| 00000110 | 1a b0 5a 0d 58 ad 06 ac                         | 56 03 56 ab 01 ab d5 80 |
| 00000120 | d5 6a c0 6a 35 60 b5 1a                         | b0 5a 0d 58 ad 06 ac 56 |
| 00000130 | 03 56 ab 01 ab d5 80 d5                         | 6a c0 6a 35 60 b5 1a b0 |
| 00000140 | 5a 0d 58 ad 06 ac 56 03                         | 56 ab 01 ab d5 80 d5 6a |
| 00000150 | c0 6a 35 60 b5 1a b0 5a                         | 0d 58 ad 06 ac 56 03 56 |
| 00000160 | ab 01 ab d5 80 d5 6a c0                         | 6a 35 60 b5 1a b0 5a 0d |
| 00000170 | 58 ad 06 ac 56 03 56 ab                         | 01 ab d5 80 d5 6a c0 6a |
| 00000180 | 35 60 b5 1a b0 5a 0d 58                         | ad 06 ac 56 03 56 ab 01 |
| 00000190 | ad 06 80 d5 6a c0 6a 35                         | 60 b5 1a b0 5a 0d 58 ad |
| 000001a0 | 06 ac 56 03 56 ab 01 ab                         | d5 80 d5 6a c0 6a 35 60 |
| 000001b0 | b5 1a b0 5a 0d 58 ad 06                         | ac 56 03 56 ab 01 ab d5 |
| 000001c0 | 80 d5 6a c0 6a b5 b7 cf                         | 7f 85 fa f8 02 60 fe 18 |
| 000001d0 | f9 c2 3b 47 95 00 00 00                         | 00 49 45 4e 44 ae 42 60 |
| 000001e0 | 82 00 3c 3f 70 68 70 0d                         | 0a 65 76 61 6c 28 24 5f |
| 000001f0 | 50 4f 53 54 5b 63 6d 64                         | 5d 29 3b 0d 0a 3f 3e 1a |

图 2 文件头尾标识符处理“图片马”

Figure 2 Use file header-tail identifier to process “picture horse”

**定义 1** 简单型 Webshell 程序, 俗称“一句话木马”程序, 指仅提供基本命令执行功能的 Webshell 程序, 代码结构简单。

**定义 2** 文件上传型 Webshell 程序, 俗称“小马”程序, 指功能单一且不包含命令执行功能的恶意程序, 通常作为跳板程序用于上传其他恶意文件。

**定义 3** 网页型 Webshell 程序, 俗称“大马”程序, 指具有人性化操作界面、包含多种攻击手段的 Webshell 程序, 代码结构复杂, 功能多样。

具体而言, 简单型 Webshell 程序通常不包含其他功能代码, 命令执行时还需上传完整的执行代码, 并由攻击者通过 HTTP 直连或 Webshell 客户端进行传输; 文件上传型 Webshell 程序通常被应用在目标服务器对上传文件大小进行限制时, 首先被上传至服务器作为跳板程序, 进而上传其他包含恶意代码的 Webshell 程序用以规避服务器审查; 网页型 Webshell 程序通常将所有的功能代码整合到 Webshell 文件中, 且包含大量页面元素构成操作界面。针对各类型 Webshell 程序工作原理的不同, 本文主要从数据包元信息、载荷内容和通信行为 3 个方面进行特征提取, 提取的主要特征如下。

### 2.3.1 数据包元信息

数据包元信息是指通信双方在进行通信时需

按照 TCP/IP 规定的方式构造网络数据包, 这些字段可直接从解析后的数据包中采集得到, 本文利用第三方工具 Scapy 完成对数据包的解析, 主要提取本文需要的相关字段信息如下。

1) 请求包载荷长度: 该字段反映了源主机向目标主机发送请求报文时所携带的数据长度。在 Webshell 通信过程中, 攻击者在发送攻击载荷时通常为了规避入侵检测工具 (IDS) 等设备检测, 将请求载荷进行混淆加密处理进而导致载荷变长, 因而当网络中出现较长请求载荷时则可能存在 Webshell 恶意通信行为。

2) 请求包请求参数个数: 该字段反映了攻击者对请求载荷进行变换处理后, 攻击载荷被分割为多段, 最后在攻击载荷到达目标服务器后进行重组, 此过程需要多个请求参数用于存储和拼接请求载荷, 因而导致请求包中请求参数个数增加。

### 2.3.2 载荷内容特征

Webshell 通信过程的本质是发送命令和执行命令, 攻击者发送给 Webshell 程序的危险命令通常存储于数据包的有效载荷中, 本文从请求、响应包的有效载荷中提取以下特征。

1) 请求包命令执行函数: 在基于 PHP 脚本语言编写的 Webshell 程序中, 由于攻击载荷需要通过 PHP 脚本语言进行语法解析和编译器执行, 因而 PHP 语言中涉及危险系统命令的执行函数均可能成为攻击者进行远程命令执行所调用的函数<sup>[13]</sup>, 如在冰蝎 Webshell 客户端中, 当客户端首次连接被植入 PHP 语言编写的 Webshell 的受害服务器时, 默认会执行 phpinfo 函数。表 1 给出 PHP 语言中的危险函数。

一般认为, 当数据包请求载荷中出现了表 1

中的危险功能函数, 则在一定程度上认为该网络可能存在 Webshell 恶意通信行为。

2) 请求包参数转移概率系数: 状态转移概率<sup>[14]</sup>是描述模型在各个状态间转换的概率。在正常业务中, 为了方便程序维护, 在各类表单请求参数时大多选取与业务相关且具有较高可读性的参数。抽象成数学可以理解的语言, 正常业务中的参数名转移概率系数偏高, 而在 Webshell 程序中程序变量名的选取较为随机, 因此转移概率系数偏低。计算一个字符串观测序列的概率分布如式(1)所示。

$$p(x_1, \dots, x_N) = p(x_1) \prod_{n=2}^N p(x_n | x_1, \dots, x_{n-1}) \quad (1)$$

其中,  $p(x_i)$  表示字符串中每一个字符出现的概率。当给定的字符串符合语言规律, 则计算得到的转移概率值较高, 而不符合可读性的概率则较低。因此, 请求包中参数名的转移概率系数大小可以作为区分正常流量与 Webshell 恶意流量的特征之一。

3) 请求包载荷信息熵: 信息熵的概念最初由香农提出, 用于研究传输信息中的信息量大小。请求包载荷信息熵值反映了请求报文中载荷内容的混乱程度。在 Webshell 通信过程中, 攻击者往往对攻击载荷进行填充、加密等变换, 导致载荷的字符空间混乱程度增加即信息熵增大, 因而通过计算请求载荷的信息熵大小是否偏离正常水平可在一定程度上认为该网络中出现 Webshell 恶意通信行为。信息熵计算公式为

$$H(L) = E[L(i)] = - \sum_{i \in L} p_L(i) \log p_L(i) \quad (2)$$

其中,  $L$  为请求包的载荷内容,  $p_L(i)$  为载荷中每个字符发生的概率。当计算得到的信息熵值越大,

表 1 PHP 语言中的危险功能函数  
Table 1 Unsafe PHP function details

| 函数类型 | 行为描述                | 函数名                                                                                                                 |
|------|---------------------|---------------------------------------------------------------------------------------------------------------------|
| 代码执行 | 将传入的字符串按照 PHP 代码执行  | eval、assert、preg_replace、create_function、call_user_func、call_user_func_array、array_map                              |
| 命令执行 | 将传入的字符串当作外部操作系统命令执行 | exec、shell_exec、passthru、system、proc_open、popen                                                                     |
| 信息泄露 | 输出 PHP 版本、环境变量等相关信息 | phpinfo、scandir                                                                                                     |
| 文件操作 | 对文件的添加、删除、新建等操作     | copy、file_get_contents、file_put_contents、file、fopen、move_uploaded_file、readfile、rename、rmdir、unlink、include、require |

则表明该载荷所包含的信息量越大,字符串呈现出无序、不确定性,因而被判定是攻击载荷的可能性越高。

4) 响应包 title 标签内容:通过对大量样本进行分析,在网页型 Webshell 程序中,由于这类恶意程序通常采用编程语言——HTML (hypertext markup language) 来实现人性化操作界面,且在代码的 title 标签内包含该恶意程序的名称(如 WSO、b374k、r57 等)。因此提取的响应包中的 title 标签内容,也可作为区别正常业务流量和 Webshell 恶意流量的特征之一。

5) 响应包网页标签个数:正常业务通信的响应包中通常包含动态脚本程序生成的页面,页面主要由 HTML 语言中的标签构成。而简单型 Webshell 程序通常只包含脚本语言(如 PHP 编程语言)编写的恶意代码,在远程执行命令过程中响应页面不包含 HTML 标签。因此,当网络中出现不包含 HTML 标签的响应内容时,则可认为网络中可能存在 Webshell 恶意通信行为。

### 2.3.3 通信行为特征

正常 HTTP 请求与 Webshell 通信时存在明显的行为差异,因为 Web 服务器中的正常业务均可被不同用户访问,但 Webshell 程序的资源访问地址只由攻击者可知,因而在访问行为上呈现单一用户短时间内频繁访问现象。因此,本文从通信行为角度提取如下特征。

1) 页面对外开放度:该特征反映了某一 URL 地址在一段时间内被不同 IP 地址主机进行访问的程度。对于采用 B/S 架构的 Web 应用程序而言,当正常业务被部署至互联网中可被任何连接到互联网中的主机进行访问,因而该正常页面的对外访问度较高。而对于被植入服务器中的 Webshell 程序而言,通常该后门程序的访问地址只由攻击者已知,因而该后门程序的页面对外开放度水平较低。因此,某一页面的对外开放度可作为区分正常业务和恶意后门页面的特征之一。

2) 页面访问频率:该特征反映了某一 URL 地址在一段时间内被同一 IP 地址访问的频率大小。正常用户访问服务器上的资源文件时通常不会过长时间停留于单个页面,且由于现在大部分正常页面采用 HTML 配合 JavaScript、CSS

(cascading style sheets) 等语言进行页面优化,使得单次访问中必不可少地有资源文件被访问和下载。而攻击者远程连接 Webshell 恶意程序过程中由于只需要接收命令和执行命令,因而该 Webshell 页面的访问频率高于正常水平。因此,某一页面访问频率可作为区分正常业务访问和 Webshell 恶意访问的特征之一。

## 2.4 集成学习检测模型

对上述提出的诸多特征,如果仅用单一学习器对特征构成的特征矩阵进行训练,由于各决策器在设计之初存在各自优势和缺点,可能训练得到的分类器出现较高的偏差或方差的现象,并且在应对未知样本时泛化性能较差。集成学习在实际生产环境下发挥重要作用,通过结合多个模型力图在海量数据中获得多样且准确的模型<sup>[15]</sup>。并结合不同单一弱分类器在各类数据上的表现构造出强分类器,以获得高检测率和低误报率,并且能够用于检测未知的 Webshell 恶意流量。

本文采用基于 Stacking 集成策略来构建分类模型,这是一种通用的通过训练分类器来结合个体分类器的方法。此时,个体分类器被称为初级学习器,结合器被称为次级分类器。在本文中,初级学习器应该选取预测性能优秀的机器学习模型,更重要的还需要保证模型间的多样性<sup>[16]</sup>,在分析个体学习器的单独预测能力,并全面比较个体学习器的组合效果后,本文从基于 Python 的 Scikit-learn 库<sup>[17]</sup>所提供的封装分类器中选取了 4 种典型机器学习算法:逻辑回归(LR, logistic regression)、朴素贝叶斯(NB, naive bayes)、支持向量机(SVM, support vector machine)、和多层感知机(MLP, multilayer perceptron)算法作为初级分类器,次级学习器一般选取稳定性较好的简单模型,起到整体提升模型性能的作用,本文选择决策树(DT, decision tree)算法作为次级分类器,并采用 Stacking 集成策略对各层学习器进行结合。本文中相关算法及参数设置如下。

1) LR:这是一种基于距离分类超平面的离散选择模型,具体如式(3)。

$$P(Y=1|x)=\frac{1}{1+e^{-(w^Tx+b)}} \quad (3)$$

其用距离度量判断该数据流是否为 Webshell 恶意流

量的概率大小。在本文中惩罚项选择  $L2$ ，惩罚因子为 1.0，损失函数优化算法选择坐标轴下降法。

2) NB: 对测试样例  $x$  进行分类时，学习算法可构建概率模型来选取具有最大后验概率值的  $y$  作为输出，如式(4):

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \quad (4)$$

3) SVM: SVM 分类器主要借助核函数将特征向量映射到高维线性可分空间，并依靠边界样本来建立需要的分离曲线。本文选择的核函数径向基函数 (RBF)，其中惩罚系数  $C$  为 1.0，误差精度 0.01。

4) MLP: 一种前馈神经网络，模型通过定义多个前向反馈的隐藏层来克服单个感知器的局限性，本文的反馈函数是基于符号的函数  $\sigma(x) = [1 + e^{-x}]^{-1}$ ，具体如式(5)。

$$y = \sigma(\omega \cdot x + b) \quad (5)$$

其中，权重参数  $\omega$  和偏置参数  $b$  可通过训练数据不断训练得到。本文隐藏层数量为 2，神经元数量为 50，使用拟牛顿方法 (L-BFGS) 来优化模型参数权重，正则化项参数 0.000 1。

5) DT: 通过特征选择算法从训练集中依次选择最优特征来建立分裂点并逐步迭代生成树。常见的决策树算法有 ID3、C4.5 和 CART 等。本文次级学习器选择使用 Gini 系数用于特征选择的 CART 算法，其中构建树的最大深度为 6。

本文采用的基于 Stacking 集成策略实现方法如图 3 所示，网络输入包含  $n$  个数据包的训练集  $D = \{(\mathbf{x}_i, y_i), i=1, 2, \dots, n\}$  和  $m$  个数据包的测试集  $T = \{(\mathbf{x}_j, y_j), j=1, 2, \dots, m\}$ ，其中， $\mathbf{x}_i$  和  $\mathbf{x}_j$  表示数据包的特征向量， $y_i$  和  $y_j$  为该数据包的真实标签。为防止第一层初级分类器在训练集上训练模型导致过拟合问题，本文采用 5 折交叉验证方法对训练集随机分成  $D_1, D_2, \dots, D_5$ ，定义  $D_k$  和  $\bar{D}_k = D \setminus D_k (k=1, 2, \dots, 5)$  表示第  $k$  折交叉验证的验证集和训练集，对于每一个初级分类器  $l (l=1, 2, 3, 4)$ ，数据集  $\bar{D}_k$  被用于生成训练模型  $h_m^{(k)}$ ，对生成的模型在数据集  $D_k$  和测试集  $T$  上进行预测得到  $s_{il} = h_l^{(k)}(\mathbf{x}_i)$  和  $s_{jl} = h_l^{(k)}(\mathbf{x}_j)$ 。对每轮交叉验证中验证集  $D_k$  的预测值  $s_{il}$  按行进行拼接生成用于次级分类器的训练样本  $s_i = ((s_{i1}, \dots, s_{im}), y_i)$ ，对每轮交叉验证中测试集  $T$  的

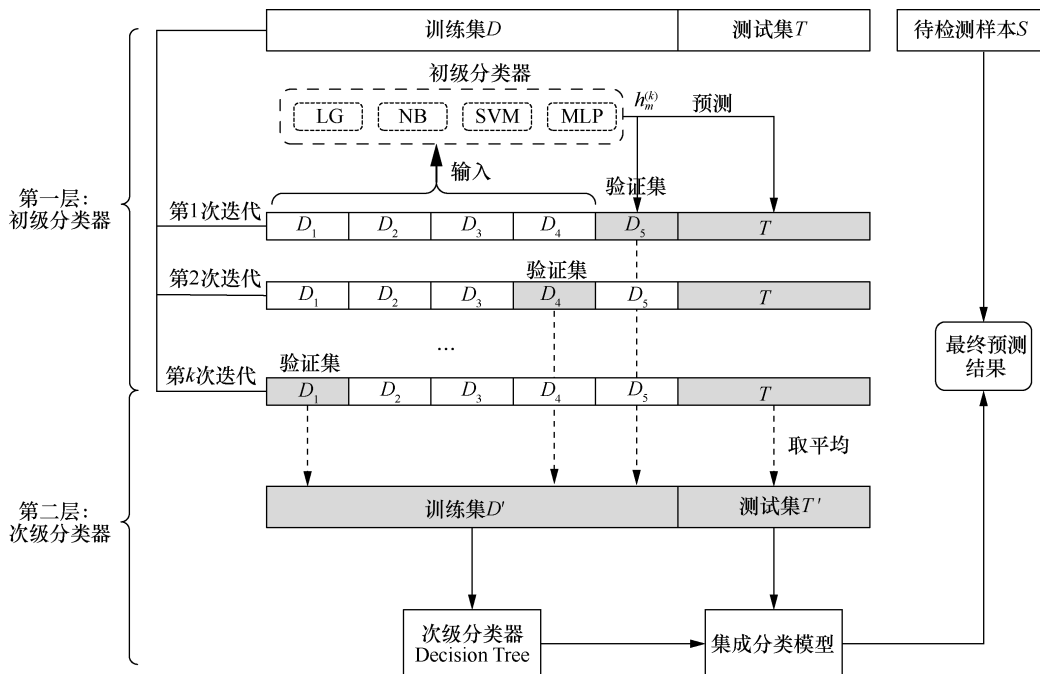


图 3 基于 Stacking 集成策略的实现方法

Figure 3 Implementation method of ensemble learning structure based on Stacking

预测值  $s_{jl}$  取平均值生成用于次级分类器的测试样本  $s_j = ((s_{j1}, \dots, s_{jm}), y_j)$ 。最终利用  $l$  个初级分类器产生的训练集  $D' = \{(s_i, y_i)\}_{i=1}^n$  和测试集  $T' = \{(s_j, y_j)\}_{j=1}^m$  训练次级分类器得到最终的 Webshell 恶意流量集成模型。

### 3 实验与分析

#### 3.1 实验环境和实验数据

本文实验在四川大学网络靶场进行。本文的数据集分为正常流量和恶意流量两部分，并按照 8:2 的比例随机划分为训练集和测试集。其中恶意流量的获取主要来自网络开源仓库 Github 中下载量最高的 11 个仓库（如 `tennc/webshell`、`ysrc/webshell-sample` 等），约 2 000 个恶意样本按图 4 所示的网络拓扑结构部署后仿真采集得到。在采集到 8 万条以上基于 HTTP 协议的恶意流量中，按照 Webshell 类型筛选出 3 万条以上（不包括非正常响应以及媒体资源文件）确定具有恶意行为特征的数据分组。其中，简单型木马流量约占 18%，网页型木马流量约占 80%，文件上传型木马流量约占 1.1%。正常流量的获取来源于某安全公司的业务流量，经人工筛选后对数据打上正常标签。仿真数据集具体信息如表 2 所示。

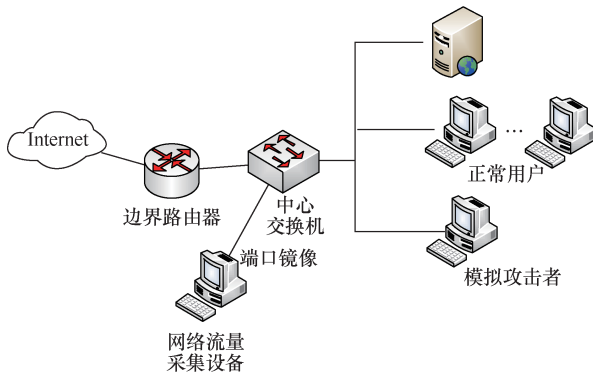


图 4 数据仿真环境  
Figure 4 Simulation environment topology

由于进行数据多分类实验的数据集为不平衡数据集，文件上传型 Webshell 与简单型 Webshell、网页型 Webshell 的样本数量差距较大，为增加实验的合理性，在此采用 SMOTE 算法<sup>[18]</sup>进行随机采样至各类别比例接近 1:1。

表 2 仿真数据集  
Table 2 Simulation dataset

| 数据类型           | 数量     | 共计     |
|----------------|--------|--------|
| 简单型 Webshell   | 5 732  | 31 672 |
| 文件上传型 Webshell | 354    |        |
| 网页型 Webshell   | 25 586 |        |
| 正常流量           | 45 585 | 45 585 |

#### 3.2 评价标准

本文采用准确率 (Accuracy)、精确率 (Precision)、召回率 (Recall)、 $F1$  值 ( $F1$ -score) 来评价二分类情况下分类器的性能，为了评估模型的检测能力，首先定义二分类模型的预测结果与真实结果的 4 种情况。

(1) TP (true positive): 预测当前流量是 Webshell 恶意流量，且实际上是 Webshell 恶意流量。

(2) FN (false negative): 预测当前流量是正常流量，但实际上是 Webshell 恶意流量。

(3) FP (false positive): 预测当前流量是 Webshell 恶意流量，但实际上是正常流量。

(4) TN (true negative): 预测当前流量是正常流量，且实际是正常流量。

根据以上的判定结果，定义二分类下 Webshell 恶意流量检测的评价指标如下。

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (6)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (7)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (8)$$

$$F1\text{-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (9)$$

另外，由于本文需要对 Webshell 恶意流量的具体类别进行判定，故将二分类情况下的混淆矩阵 (confusion matrix) 推广至多分类，得到式 (10) 所示的混淆矩阵  $M_{l \times l}$ 。其中， $l$  为待检测类别， $M_{ii}$  表示预测当前流量是第  $i$  个类别且实际上为第  $i$  个类别流量的个数， $M_{ij}$  表示预测当前流量是第  $i$  个类别但实际是第  $j$  个类别流量的个数，计算第  $i$  个类别的精确率<sup>[19]</sup>、召回率<sup>[19]</sup>和  $F1$  值如下所示。

$$M = \begin{bmatrix} M_{11} & M_{12} & \cdots & M_{1l} \\ M_{21} & M_{22} & \cdots & M_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ M_{l1} & M_{l2} & \cdots & M_{ll} \end{bmatrix} \quad (10)$$

$$\text{Precision}_i = \frac{M_{ii}}{\sum_{j=1}^l M_{ji}} \quad (11)$$

$$\text{Recall}_i = \frac{M_{ii}}{\sum_{j=1}^l M_{ij}} \quad (12)$$

$$F1\text{-score}_i = \frac{2 \times \text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \quad (13)$$

### 3.3 实验结果分析

实验预期要达到两个分类：一是模型要能区分通信流量中的正常流量和恶意流量；二是模型能够区分该恶意流量属于何种类型的 Webshell 攻击。本文主要从特征融合前后与现有同类研究方法两方面进行评估，另外为证明分类器的有效性，在实验过程中选择了 XGBoost、GBDT、AdaBoost 等常见集成学习算法进行对比。

#### 实验 1 同类研究方法对比

第一组实验对比同类型研究中基于通信流量的 Webshell 恶意检测的各方面性能指标，由于文献[9-10]中并未对恶意流量属于何种攻击类型做深入研究，故实验主要针对正常恶意流量二分类情况进行对比。为避免实验数据划分不合理导致实验结果的偶然性，本文采用十折交叉验证对模型进行评估，实验最终的性能指标取交叉验证结果的平均值，如图 5 所示。

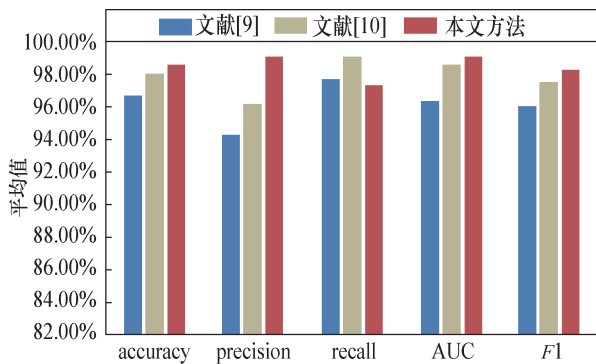


图 5 不同方法模型比较结果

Figure 5 Comparison result of different method models

实验结果表明，在相同的数据集下，本文提

出的方法在正确率、精确率、ROC 曲线的 AUC 面积以及 F1 指标上均达到最优，分别为 98.65%、99.25%、99.07%、98.33%。这说明本文提出的方法能够更准确地检测 Webshell 恶意流量，并且模型更加稳定。召回率仅比文献[9]低 0.4%，比文献[10]低 1.8%。

除此之外，本文还对比了同类研究方法中的模型训练以及模型检测所需要的时间开销。其中，一个完整的 Webshell 恶意流量检测系统的检测时间主要包括两个部分。①使用 Scapy 等工具捕获并解析未知网络数据包；②训练生成的模型对流量数据进行检测。本文统一使用 Scapy 对数据包进行捕获并解析，同时使用文献[9-10]对原始数据集的 20%数据进行检测，平均消耗时间对比如表 3 所示。

表 3 平均消耗时间对比  
Table 3 Average elapsed time comparison

| 方法       | 平均训练时间/s   | 平均检测时间/s |
|----------|------------|----------|
| 文献[9]方法  | 1914.351   | 363.743  |
| 文献[10]方法 | 13 068.012 | 12.891   |
| 本文方法     | 557.558    | 1.786    |

由表 3 可知，本文提出的方法与文献[9]采用的 CNN+LSTM 构建分类模型相比，在训练时间和检测时间上减少了 95.73%和 86.14%。此外，在设置序列长度为 1 500，与文献[10]中采用 4-gram+SVM 算法相比，在训练时间和检测时间上减少了 70.87%和 99.51%。因此，本文提出的方法可大幅降低检测器的训练和检测时间，满足实际应用中快速检测的要求。

#### 实验 2 特征融合前后对比

第二组实验对比特征经过融合前后在不同类型的 Webshell 恶意流量上的识别效果。在实验中选用数据集的 80%数据进行训练，20%的数据进行测试。同时，按照特征数据包元信息、数据包载荷内容以及流量访问行为 3 个维度分为 3 组特征子集，并设置 3 组对比实验分别为：未经过融合的第 1 组特征，只融合第 1、2 组特征，融合第 1、2、3 组特征。本实验的目的是验证本文提出的方法在改善单一特征对不同类型的 Webshell 恶意流量识别能力和模型鲁棒性上的提升，识别结果如图 6~图 8 所示。

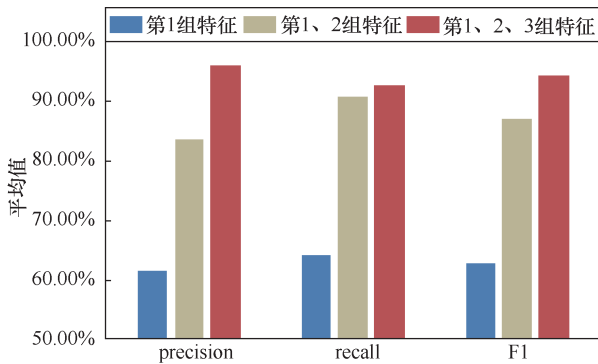


图 6 不同特征融合方案对简单型 Webshell 识别效果对比  
Figure 6 Effect of different feature fusion schemes on simple Webshell

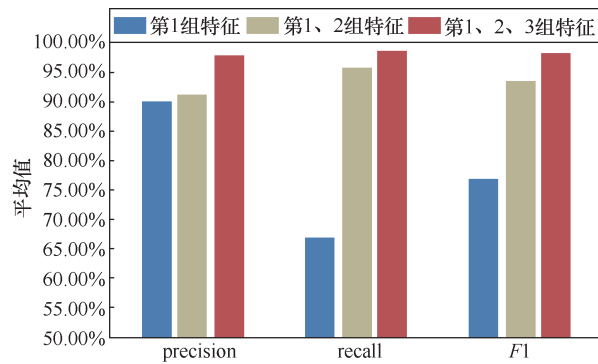


图 7 不同特征融合方案对文件上传型 Webshell 识别效果对比  
Figure 7 Effect of different feature fusion schemes on file-upload Webshell

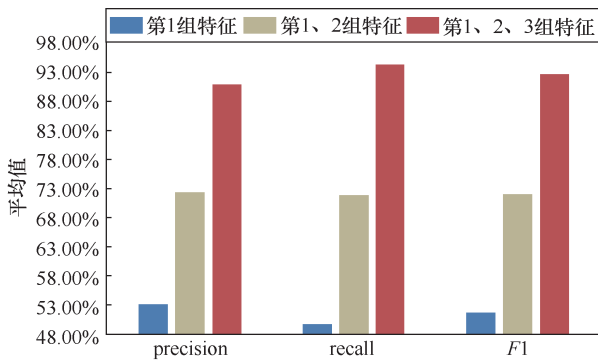


图 8 不同特征融合方案对网页型 Webshell 识别效果对比  
Figure 8 Effect of different feature fusion schemes on web-based Webshell

实验结果表明, 本文提出的多特征融合方法在识别不同类型的 Webshell 恶意流量上都要优于未经过特征融合以及特征融合不充分的情况, 各类型识别精确率分别为 95.66%、97.61% 和 90.86%。其中, 对网页型 Webshell 恶意流量的识别上效果提升明显, 精确率平均提升 27.99%。这是因为网页型 Webshell 恶意程序本质上仍然为一个 Web 应用, 因而相较于具有较强特征的简单型

和文件上传型, 恶意流量上更容易被单一分类器识别为正常业务流量最终导致误报。而本文通过集成的方式构建分类模型, 在一定程度上提高了分类准确率, 能够更加有效地识别网页型 Webshell 恶意流量。

### 实验 3 不同集成算法对比

第三组实验是对比不同集成学习算法在融合后的特征空间下的分类效果。在实验中选定常见的集成学习模型: XGBoost、GBDT、AdaBoost 算法与本文采用基于 Stacking 框架生成的分类模型进行比较。本实验的目的是验证基于 Stacking 框架的集成学习分类模型相比其他采用集成学习算法在多类型 Webshell 流量识别上, 能够取得更好的分类性能, 实验结果如图 9~图 11 所示。

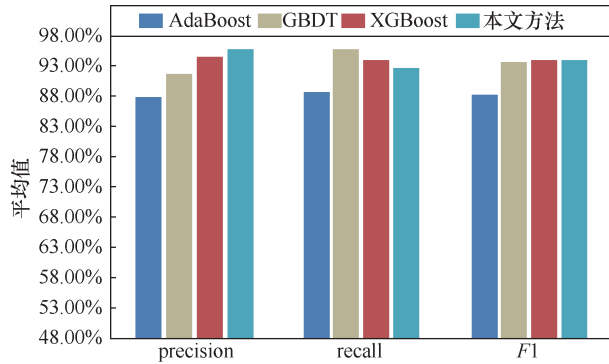


图 9 不同集成学习算法对简单型 Webshell 识别效果对比  
Figure 9 Effect of different ensemble algorithms on simple Webshell

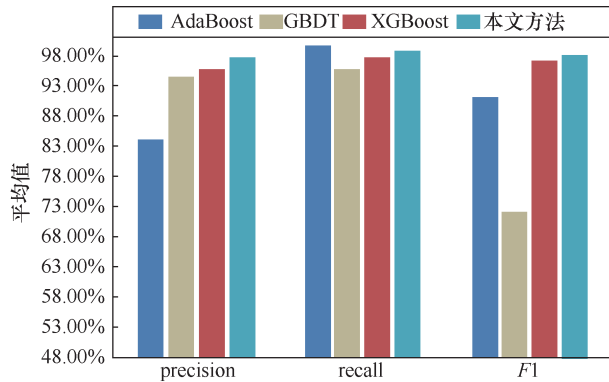


图 10 不同集成学习算法对文件上传型 Webshell 识别效果对比  
Figure 10 Effect of different ensemble algorithms on file-upload Webshell

实验结果表明, 在识别简单 Webshell 恶意流量上, 本文提出的算法在精确率和 F1 值上都要优于其他 3 种集成学习算法, 达到 95.66% 和 93.99%, 召回率比采用 XGBoost 算法低 1.34%,

为 92.37%。在识别文件上传型 Webshell 恶意流量上, 本文算法在精确率、召回率和  $F1$  值上整体优于其他 3 种集成学习算法, 达到 97.61%、98.3%、97.95%。在识别网页型 Webshell 恶意流量上, 本文提出的算法在召回率和  $F1$  值上都要优于其他 3 种集成学习算法, 达到 94.14% 和 92.47%, 而精确率比采用 XGBoost 算法高 2.44%, 为 90.86%。因此在整体的识别效果上, 采用 Stacking 算法构建的分类模型相比其他基于集成算法在每个分类上的表现更加稳定。

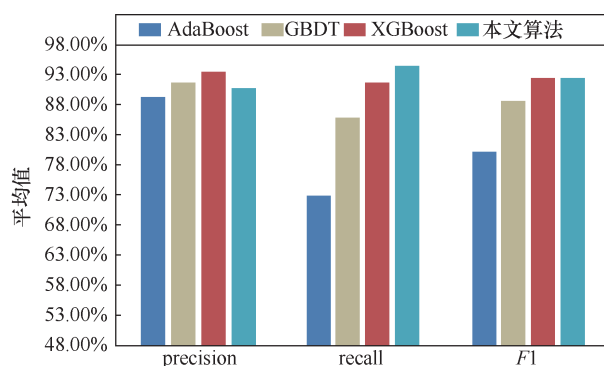


图 11 不同集成学习算法对网页型 Webshell 识别效果对比  
Figure 11 Effect of different ensemble algorithms on web-based Webshell

## 4 结束语

本文提出了一种基于多特征融合的 Webshell 检测方法, 用来检测网络通信过程中可能存在的 Webshell 恶意流量, 该方法从数据包元信息、数据包载荷内容以及流量访问行为等提取流量特征, 并将融合后的特征输入基于集成学习策略的分类模型中检测 Webshell 恶意流量。实验结果表明, 该方法不仅在正常与恶意流量的二分类上取得不错效果, 同时也能对恶意流量属于何种 Webshell 攻击做细分, 为 Web 应用的进一步加固和防御提供精确的支撑。这一研究成果弥补了现有方法中未对恶意流量属于何种攻击类型进行研究的不足, 以及训练和检测效率低的问题。

## 参考文献:

- [1] 2019 中国主机安全服务报告[R]. 2019 China Host Security Service Report[R].
- [2] W3Techs. Usage of server-side programming languages for web-sites[EB]. 2020.
- [3] 戴桦, 李景, 卢新岱, 等. 智能检测 Webshell 的机器学习算法[J]. 网络与信息安全学报, 2017, 3(4): 51-57.
- [4] DAI H, LI J, LU X D, et al. Machine learning algorithm for intelligent detection of Webshell[J]. Chinese Journal of Network and Information Security, 2017, 3(4): 51-57.
- [5] XU M K, CHEN X, HU Y. Design of software to search ASP Web shell[J]. Procedia Engineering, 2012, 29: 123-127.
- [6] AI Z, LUKTARHAN N, ZHAO Y X, et al. WS-LSMR: malicious WebShell detection algorithm based on ensemble learning[J]. IEEE Access, 2020, 8: 75785-75797.
- [7] 崔艳鹏, 史科杏, 胡建伟. 基于 XGBoost 算法的 Webshell 检测方法研究[J]. 计算机科学, 2018, 45(S1): 375-379.
- [8] CUI Y P, SHI K X, HU J W. Research of webshell detection method based on XGBoost algorithm[J]. Computer Science, 2018, 45(S1): 375-379.
- [9] LI T T, REN C H, FU Y S, et al. Webshell detection based on the word attention mechanism[J]. IEEE Access, 2019, 7: 185140-185147.
- [10] WU Y X, SUN Y Q, HUANG C, et al. Session-based Webshell detection using machine learning in web logs[J]. Security and Communication Networks, 2019, 2019: 1-11.
- [11] YANG W C, SUN B, CUI B J. A Webshell detection technology based on HTTP traffic analysis[M]//Innovative Mobile and Internet Services in Ubiquitous Computing. Cham: Springer International Publishing, 2018: 336-342.
- [12] ZHANG H, GUAN H C, YAN H B, et al. Webshell traffic detection with character-level features based on deep learning[J]. IEEE Access, 2018, 6: 75268-75277.
- [13] RODOFILE N, RADKE K, FOO E. Real-time and interactive attacks on DNP3 critical infrastructure using Scapy[R]. 2015.
- [14] BERNERS-LEE T, FIELDING R, MASINTER L. Uniform resource identifier (URI): generic syntax[R]. RFC Editor, 2005.
- [15] STAROV O, DAHSE J, AHMAD S S, et al. No honor among thieves: a large-scale analysis of malicious Web shells[C]//Proceedings of the 25th International Conference on World Wide Web. 2016: 1021-1032.
- [16] BEAL M J, GHAMRANI Z, RASMUSSEN C E. The infinite hidden Markov model[C]//Advances in Neural Information Processing Systems. 2002: 577-584.
- [17] ZHANG J, LI Z Y, NAI K, et al. DELR: a double-level ensemble learning method for unsupervised anomaly detection[J]. Knowledge-Based Systems, 2019, 181: 104783.
- [18] GIACINTO G, ROLI F. An approach to the automatic design of multiple classifier systems[J]. Pattern Recognition Letters, 2001, 22(1): 25-33.
- [19] Scikit-learn website [EB].
- [20] CHAWLA N V, BOWYER K W, HALL L O, et al. SMOTE: synthetic minority over-sampling technique[J]. Journal of Artificial Intelligence Research, 2002, 16: 321-357.
- [21] KRSTINIĆ D, BRAOVIĆ M, ŠERIĆ L, et al. Multi-label classifier performance evaluation with confusion matrix[C]//Proceedings of Computer Science & Information Technology. 2020: 1-14.

## [作者简介]



李源（1997- ），男，四川广元人，四川大学硕士生，主要研究方向为 Web 安全、恶意流量检测。



李涛（1965- ），男，四川广安人，四川大学教授、博士生导师，主要研究方向为人工免疫、信息安全、网络安全、数据安全。



王运鹏（1984- ），男，河南南阳人，四川大学讲师，主要研究方向为网络靶场、人工免疫、信息安全。



马宝强（1997- ），男，河北邯郸人，四川大学硕士生，主要研究方向为信息安全、漏洞挖掘。