基于 XGBoost 算法的 Webshell 检测方法研究

崔艳鹏[1,2] 史科杏[1] 胡建伟[1,2]
(西安电子科技大学网络与信息安全学院 西安 710071)[1]
(西安电子科技大学网络行为研究中心 西安 710071)[2]

摘 要 为解决加密型 Webshell 与非加密型 Webshell 的代码特征不统一、难以提取的问题,提出一种基于 XGBoost 算法的 Webshell 检测方法。首先,对 Webshell 进行功能分析,发现绝大部分 Webshell 都具有代码执行、文件操作、数据库操作和压缩与混淆编码等特点,这些特征全面地描述了 Webshell 的行为。因此,对于非加密型的 Webshell,将其主要特征划分为相关函数出现的次数。对于加密型的 Webshell,根据代码的静态特性,将文件重合指数、信息熵、最长字符串长度、压缩比 4 个参数作为其特征。最后,将两种特征统一起来作为 Webshell 特征,改善了 Webshell 特征覆盖不全的问题。实验结果表明,所提方法能有效地对两种 Webshell 进行检测;与传统的单一类型 Webshell 检测方法相比,该方法提高了 Webshell 检测的效率与准确率。

关键词 Webshell 检测,XGBoost 算法,机器学习,Web 安全
中图法分类号 TP393 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.06.001

Research of Webshell Detection Method Based on XGBoost Algorithm

CUI Yan-peng[1,2] SHI Ke-xing[1] HU Jian-wei[1,2]
(School of Network and Information Security, Xidian University, Xi'an 710071, China)[1]
(Network Behavior Research Center, Xidian University, Xi'an 710071, China)[2]

Abstract To solve problem of uniform code characteristics and difficulty to extract of the encrypted Webshell and non-encrypted Webshell, this paper proposed a Webshell detection method based on XGBoost algorithm. First of all, this paper analyzed features of Webshell, and found that most of the Webshell have code execution, file operations, database operations, compression, obfuscation coding and so on, which describe the behaviors of Webshell comprehensively. Therefore, for non-encrypted Webshell, its main feature is divided into the number of occurrences of correlation functions. For encrypted Webshell, according to the statistical characteristics of the code, file coincidence index, information entropy, the length of the longest string, compression ratio are taken as four parameters as its features. Finally, these two type of features are gregarded together as a Webshell features, improving the problem of lack of Webshell feature coverage. The experimental results show that the proposed method can achieve high performance, compared with the traditional single-type Webshell detection, it improves the efficiency and accuracy of Webshell detection.

Keywords Webshell detection, XGBoost algorithm, Machine learning, Web security

随着计算机网络技术的飞速发展,以 B/S 架构为基础的 Web 应用程序越来越普及,使得 Web 研发人员的入门层次越来越低。这种不同研发人员之间技术水平的差异,导致很多 Web 程序在设计之初就对安全问题考虑不周[1]。

根据国家互联网应急中心(CNCERT/CC)公布的 2016 年互联网网络安全报告[2]对网站后门的综述,全年共监测到境内有 82072 个(去重后)网站被植入后门,其中政府网站有 2361 个。在实际攻击中,网站沦陷后的第一步就是上传后门。因此,为了保障服务器的安全性,对服务器上的后门进行检测极为重要。

Webshell 作为网站后门的一种,通过浏览器访问,就可以达到控制的目的[3-4]。Webshell 通过 80 端口进行信息交互,很容易穿透防火墙[5]。一旦 Webshell 被执行,很多危险的行为,如文件操作、数据库操作、执行任意命令等,都可以被触发。因为 Webshell 以网页形式呈现,加上自身具有灵活多变的特性,所以很难对其进行准确检测。本文详细分析了 Webshell 的定义和分类,描述了 Webshell 的特征,并在此基础上提出了一种基于 XGBoost 算法的 Webshell 检测方法。

1 Webshell 的定义和分类

对于管理者而言,Webshell 可以协助远程操作服务器;对于攻击者而言,Webshell 可以远程控制沦陷主机。Webshell 通常是由 PHP,ASP 或 JSP 等脚本语言编写而来,具有与站点动态交互的特性[7],其一旦存在,文件操作、数据库操作、命令执行等行为都会被触发。如果对 Web 应用的目录权限不加控制,系统命令被执行也不例外[8]。

根据程序的功能与大小,Webshell 可分为 3 类:一句话木马、小马和大马。

崔艳鹏(1978-),女,博士,副教授,主要研究方向为网络安全;史科杏(1988-),男,硕士,主要研究方向为网络安全、数据库技术,E-mail:skxde@163.com(通信作者);胡建伟(1973-),男,博士,副教授,主要研究方向为网络安全、信息对抗。

一句话木马泛指一句话的脚本，最典型的功能就是为代码执行提供一种环境，比如熟悉的 eval()函数。这种脚本大都不会脱离数据传递与数据执行[9]这个基本框架。代码量少、使用灵活、变形种类多的特性，使得它呈现出隐匿度高和检测困难的特点。一句话木马的原理如图 1 所示。

图 1 一句话木马的原理

小马指功能单一的 Webshell，以上传文件功能居多，通常作为上传大马的中转站，辅助攻击者进行渗透。其文件一般不大，也不存在口令保护。

大马指功能强大的 Webshell。相对于小马而言，功能丰富是它的显著特点。因其文件大、功能丰富，使用相关操作的函数比较多；有些大马为了躲避检测，使用了混淆手段或者加解密压缩等技术。前一种被称为未加密的 Webshell，后一种被称为加密的 Webshell。

2 Webshell 的特征

对于任何检测模型而言，良好的特征属性是获得更好的检测性能的关键。若选取过多的特征作为机器学习的输入，则检测模型就会变得异常复杂，导致检测的通用性大大降低；若特征属性过少，则不能很好地代表 Webshell，导致检测模型的准确率降低。因此，选取良好的特征特别关键。

因为加密型和非加密型的 Webshell(大马)对功能函数使用得较为频繁，所以从页面的静态属性与动态属性(指大马功能)进行特征提取不失为一种很好的方法。

主流的查杀工具侧重于通过特征的匹配来进行检测，对特征库的依赖性非常高，这种方法对非加密型的 Webshell 的检测效果比较好。为了躲避查杀，攻击者会采用混淆编码来对这些特征进行隐藏，加密型的 Webshell 应运而生。

常见的特征隐藏手段有：在代码中引入注释等无关的信息、外形混淆、字符串加解密、拆分、连接替换、多重编码与压缩、多文件等技术。

1)引入注释：在不影响功能的前提下，引入诸如'/*…*/'和'//…'类的注释，会对检测工作造成一定的影响，间接保护后门的长期存在。

2)外形混淆：利用脚本语言灵活多变的语法特点，通过引入大量的自定义函数名、变量名等变量，对一些容易被检测到的函数进行特征变换，增加代码的理解难度[10]。

3)字符串加解密：将敏感的代码段进行加密，只在代码具体执行的过程中进行解密，从而绕过检测。

4)字符串拆分：利用脚本语言的拼接特性，将函数名拆成各种不同的组合，使用时再进行拼接。

5)连接替换：编写 Webshell 时可能会用到一些系统控件，为了避免被检测到，给相应的控件名加入一些特殊字符，使用时通过别的函数进行过滤，从而达到躲避检测的目的。

6)多重编码与压缩：通过多重编码来改变代码的静态特征，结合压缩技术使得页面的特征发生变动，从而降低被检测到的概率。

7)多文件：将一个 Webshell 拆分成多个文件，使用时通过各种文件包含技术再将它们连接起来，这样可以有效地降低 Webshell 中一些功能函数集中出现的概率。有些检测工具根据检测到的函数进行打分，然后与阈值进行比较，使用此方法可以降低被此类工具检测到的风险。

将加密型与非加密型的 Webshell 的主要特征归纳为以下几种。

1)功能函数

将 Webshell 的所有函数依照功能进行划分，这样可以对 Webshell 的动态特性进行抽象描述。经统计发现，大部分的大马都具有代码执行、文件操作、数据库操作、压缩与编码四大特性。前 3 种是很常见的特征，压缩与编码往往是为了进行特征混淆而引进的函数。在提取特征时，只需要统计每个相关功能的一类函数出现的次数，就可以做到对 Webshell 动态属性(或页面操作)的描述。

表 1 功能函数

功能	函数详情
代码执行	assert eval python_eval shell array_map call_user_func system preg_replace passthru shell_exec exec proc_open popen curl_exec curl_multi_exec parse_ini_file show_source
文件操作	file_get_contents is_file fopen fclose fwrite wget lynx curl posix_getpwuid posix_getgrgid fileowner filegroup
数据库操作	mysql_connect mysql_query mysql_num_fields mysql_close mysql_fetch_array mysql_fetch_assoc mysql_num_rows mysql_result mysql_affected_rows mysql_select_db mssql_connect mssql_query mssql_num_fields mssql_field_name mssql_fetch_array mysql_close
压缩与编码	gzdeflat gzcompress gzuncompress gzdecode str_rot13 gzencode base64_decode base64_encode

2)文件重合指数

文件重合指数[11](Index of Coincidence，IC)最早被用于密码分析和文本自然语言分析。Webshell 经过混淆编码等技术后，已不再具有正常语言的明文特性。根据基本测试和计算，在 26 个字母构成的一段有意义的文字中，任取两个元素，它们刚好相同的概率为 0.065，因此若一段明文是用同一个字母做密钥加密的，则这个概率是不变的。相关计算如下：如果英文的 26 个字母在有意义文本中出现的概率分别为 $p_i(i=0,1,2,\cdots,25)$，那么出现两个元素相同的概率为：

$$I_c \approx \sum_{i=0}^{25} p_i^2 = 0.065 \tag{1}$$

如果是用不同的字母，则这个概率是会发生变化的。经计算可知，在一个随机的字母串(而不是一段有意义的文字)中抽取两个相同的字母，其概率为 0.038。计算公式如下：

$$I_c \approx \sum_{i=0}^{25} \left(\frac{1}{26}\right)^2 = 0.038 \tag{2}$$

Webshell 经混淆编码后，在某种意义上可以当作密文，随着其内容的随机性变大，重合指数也会发生相应的变化。因此，可以把重合指数作为 Webshell 的一个特征。

3)信息熵

信息熵是数学领域中的一个抽象概念，可被理解成某种特定信息出现的概率。越是有序的信息，信息熵越低；反之，信息熵越大。为了隐藏一些特征，Webshell 的代码往往很混乱，信息熵也随之变大。因此，信息熵可以作为区分正常文件与 Webshell 文件的标志。其计算公式如下：

$$\begin{aligned} entropy(A) &= -\sum_{n=1}^{255} \{p_n * \log_2(p_n)\} \\ &= -\sum_{n=1}^{255} \{\frac{x_n}{s} * \log_2(\frac{x_n}{s})\} \end{aligned} \tag{3}$$

其中，n 为 ASCII 码，x_n 为第 n 位 ASCII 码在当前文件中出现的次数，s 为当前脚本文件中的总字符数。

4）最长字符串长度

正常的网页语言一般不会出现很长的字符串；Webshell 经过混淆编码后，很容易出现超出正常语法单词的长字符串。如果一个网页中出现了很长的字符串，它被当作 Webshell 的概率就会特别大。因此，可以选取最长字符串作为 Webshell 的一个特征。

5）压缩比

压缩比，即为压缩文件大小与原始文件大小的比值。使用压缩技术的主要目的是消除字符的不均衡分布，高频字符对应短码，低频字符对应长码。经过混淆编码的 Webshell 消除了非 ASCII 字符后，会表现出更小的不均衡分布，从而导致压缩值变大。因此，文件的压缩比可以作为 Webshell 的一个特征。

最后，将功能函数、文件重合指数、信息熵、最长字符串长度和压缩比统一起来作为 Webshell 的特征。

3　相关工作

对于 Webshell，当前主要有以下几种主流检测技术，如图 2 所示。

图 2　Webshell 检测方法的分类

1）静态特征检测

该类检测方法的核心是围绕脚本文件的特征对关键词、文件权限、文件修改时间、文件拥有者、高危函数等多个维度进行信息提取，检测效果的好坏与这些特征的设定密切相关。Shelldetector 依靠收集的众多 Webshell 特征库来检测 Webshell，宣称识别率特别高。D 盾通过对各种具有危险功能的代码执行函数识别的方法来检测 Webshell，如 preg_replace，fputs 等特殊函数。文献[12]基于页面属性和页面操作对 Webshell 进行特征提取，然后使用 SVM 分类器来进行判定。文献[3]提取 Webshell 的文档属性、基本属性、高级属性 3 类特征，采用 C4.5 决策树方法对 Webshell 进行检测。文献[13]把 Webshell 的文件特征、编码特征、语言特征、标签特征、静态文件特征等作为特征数据，采用 SVM 方法对 Webshell 进行检测。文献[14]对 Webshell 进行检测时，致使 Webshell 的特征达到 30 维，虽然效果不错，但特征过于复杂。

静态特征检测方法对一些已有的 Webshell 有效，但对 0day 的 Webshell 往往无效。由于过于依赖特征库的特点，该方法的误报率很高。对于加密或者采用混淆编码的 Webshell 而言，该方法自身也存在特征根本无法提取的问题。因此，静态特征检测往往需要手工检测作为辅助，才能完成 Webshell 的检测工作。

2）流量分析检测

该类方法的核心是通过建立网关对流量进行可视化处理，然后对 Webshell 访问过程中产生的 payload 网络流量进行监测。经过一定的 payload 积累和相关规则的定制，通过与其他的检测过程相结合建立起一套基于流量分析的检测引擎，然后将其嵌入到现有的网关型设备或云端设备上，就可以实现对 Webshell 的深度分析与查杀。

这种采用网关型的技术检测方法多数是基于大数据的处理模型，然后在核心路由（或交换机）上将 HTTP 流量以旁路的形式做成镜像流量，通过 Hadoop 等云计算平台进行海量数据处理，最终实现对攻击场景的还原。另一方面，还必须建立机器学习的分析模型，通过一定的流量积累与模型建立，使得 Webshell 引擎能自动分析并识别异常流量。这种自动检测流量的方式会带来很大的不确定性。从可用性角度考虑，这种检测方式在实时检测的同时，可以快速地对入侵者进行定位，进而还原攻击场景；但是 Hadoop 云计算平台与机器学习模型的建立较为复杂，成本开销也不小。具体操作时，可以结合 IDS（入侵检测设备）进行部署。

3）日志分析检测

作为一种取证与预测的分析手段，日志分析讲述了完整的已发生、正在发生的、将来会发生的攻击事件。而采用日志分析的 Webshell 检测，就是根据确定的攻击事件来进行事件回溯的，依据攻击事件的特征来防范下次遭受到相同的攻击。该类方法一般是先对日志建模，然后找到异常日志与攻击日志，其本质是对 Webshell 访问日志进行提取与确认的过程。如果数据量过大，可用 Hadoop 来进行日志分析。

日志分析技术主要是从文本特征、统计特征和页面特征 3 个角度来检测异常文件，效果的好坏依赖于检测规则的完善与否；也有通过对请求模型进行建模的方法来检测 Webshell，其属于一种词汇分析技术[15]。文献[16]采用了前一种日志检测技术。

4）行为分析检测

行为分析技术涉及到源码文件在系统环境中的解析过程。其中，涉及到文件方面的行为有文件读写、创建与删除等；涉及到网络方面的行为有 socket 监听行为、TCP/UDP/HTTP 请求发送（DDOS 攻击）等；涉及到数据库读写的行为有数据库查找、修改、全库备份（脱库）等；涉及到系统配置的行为有 Windows 注册表的修改、启动配置文件等。

以 PHP 为例，可以对 PHP 脚本的执行过程做行为分析，根据其执行机制编写相应的扩展模块，然后采用 hook 技术，即过滤并阻止相关的异常操作行为。这种方式是通过监控操作系统底层的 API（应用程序接口）调用实现的。另一种方式是采用蜜罐技术，即将站点源码放在蜜罐中，然后对其行为特征进行分析来检测异常。这种方法在技术实现上需要一定的语言基础，当业务以集群化规模运行时会出现各种误报，但不可否认其确实具有一定的实验意义。

5）统计学检测

此技术的重点在于，依据 Webshell 脚本与正常文件的差异性来识别混淆代码。其中主要涉及到 5 个特征：信息熵（Entropy），通过使用 ASCII 码来衡量文件的不确定性；最长单词（LongestWord），正常文件里的字符串长度符合英文规范，不会出现极端情况，出现很长的字符串很大程度上意味着代码被编码加密或者混淆过；重合指数（Index of Coincidence），低的重合指数预示着文件被加密过；特征值匹配（Signature），匹配特征函数与敏感代码；压缩比（Compression），Webshell 经编码后，与正常文件相比，压缩比会变大。其检测

核心是通过统计学的方法计算出正常的 PHP 文件的统计学特征的范围，然后将其与用户上传的文件进行比较。

该方法在识别混淆编码、加密的 Webshell 方面表现良好，如果将这些代码混入正常的文件里面，很可能产生误判。

NeoPI[17] 侧重于对混淆编码的文件进行识别，通过文件重合指数、信息熵、最长字符串、压缩比、特征等参数进行综合比较分析。文献[8]在对加密型 Webshell 进行检测时，通过提取文件重合指数、信息熵、最长字符串长度的方差、压缩比 4 种特征，采用朴素贝叶斯分类器对加密型 Webshell 进行检测。

与其他算法相比，本文提出的 Webshell 检测方法的各项分类指标均表现良好。

4 基于 XGBoost 的 Webshell 检测

4.1 检测框架

基于 XGBoost[18] 的 Webshell 检测模型首先对训练样本和测试样本分别进行特征提取，从而得到训练特征和测试特征，由 XGBoost 分类器完成分类操作，最终输出结果。检测框架如图 3 所示。

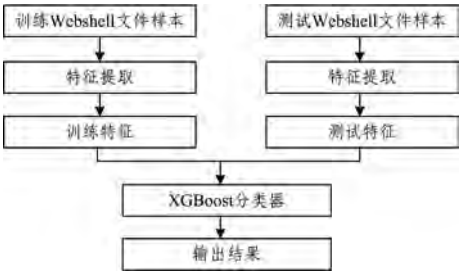

图 3 XGBoost 检测模型的框架

4.2 特征提取

特征选取的好坏直接关系到检测结果的优劣。在进行特征选取时，既要考虑页面的静态特征，也要考虑页面的动态特征。

如果提取页面的全部特征进行处理，则很容易陷入过拟合，这对检测的效率是致命的打击；特征过少，误报率会有一定程度的上升。通过提取页面的功能函数（代表 Webshell 功能的一些函数），既可以反映页面的静态特征，也可以反映页面的动态特征，尤其是后面的压缩与编码功能函数的提取，代表了相当一部分的 Webshell。最后，选取功能函数、文件重合指数、信息熵、最长字符串长度、压缩比等作为特征。

4.3 XGBoost 算法

XGBoost 算法是一个大规模、分布式的通用 Gradient Boosting(GBDT)库[19]。很多数据大赛的参赛者热衷于使用此算法，并且获得了相当不错的成绩。作为一种集成学习算法，其基学习器是一种弱学习器，开始时权重都一样，在迭代的过程中每一次都会根据上一次的预测结果进行加权，从而使得误差越来越小。

本文所用的 Boosting 算法的过程如下[20]：

0) $F_{k0}(x)=0,k=1,\cdots,K$

1) For m=1 to M

2) $p_k(x)=\exp(F_k(x))/\sum_{t=1}^{K}\exp(F_t(x)),k=1,\cdots,K$

3) For k=1 to K do:

4) $\tilde{y}_{ik}=y_{ik}-p_k(x_i),i=1,\cdots,N$

5) $\{R_{jkm}\}_{j=1}^{J}=J-\text{terminal node tree}(\{\tilde{y}_{ik},x_i\}_1^N)$

6) $r_{jkm}=\frac{K-1}{K}\frac{\sum_{x_i\in R_{jkm}}\tilde{y}_{ik}}{\sum_{x_i\in R_{jkm}}|\tilde{y}_{ik}|(1-|\tilde{y}_{ik}|)},j=1,\cdots,J$

7) $F_{km}(x)=F_{k,m-1}(x)+\sum_{j=1}^{J}r_{jkm}1(x\in R_{jkm})$

endFor

endFor

end Algorithm

0)给定一个初始值。

1)建立 M 棵决策树，即表示须迭代 M 次。

2)对函数估计值 $F(x)$ 进行 Logistic 变换。

3)对 K 个分类进行下面的操作：每一个样本点 x_i 都对应了 K 个可能的分类 y_i，所以 y_i，$F(x_i)$，$p(x_i)$ 都是一个 K 维的向量。

4)求残差减少的梯度方向。

5)根据每一个样本点 x 与其残差减少的梯度方向，得到一棵由 J 个叶子节点组成的决策树。

6)决策树建立完成后，通过公式 $r_{jkm}=\frac{K-1}{K}\frac{\sum_{x_i\in R_{jkm}}\tilde{y}_{ik}}{\sum_{x_i\in R_{jkm}}|\tilde{y}_{ik}|(1-|\tilde{y}_{ik}|)}$ 可以得到每一个叶子节点的增益；每个增益的组成其实也是一个 K 维的向量，表示在决策树预测的过程中，如果一个样本点掉入了这个叶子节点，则其对应 K 个分类值的大小。

7)将当前得到的决策树与之前的那些决策树合并起来后作为一个新模型。

5 实验

5.1 实验数据

不同语言编写的 Webshell 具有极其相似的特征，相对应的检测模型也大体一致，因此选择其中一种语言编写的网页作为实验数据即可。实验选取了从 github 上搜集的 1067 个恶意 PHP Webshell 和 305 个正常的 PHP 页面样本作为数据来源。在对 Webshell 进行具体的检测和训练前，首先将这些收集的样本分成 4 份，任选其中 3 份作为训练数据集，另外 1 份作为测试数据集。重复 4 次（机器学习常用的方法——四折交叉验证），根据平均结果来评估其检测能力。

5.2 评价标准

在对 Webshell 进行检测时，可能出现以下 4 种情况。在实验中，1 表示 Webshell，0 表示普通页面。

表 2 预测分类的结果

实际结果	预测结果	
	1	0
1	TP	FN
0	FP	TN

TP(True Positive)：预测当前页面是 Webshell，实际上是 Webshell。

FN(False Negative)：预测当前页面不是 Webshell，实际上是 Webshell。

FP(False Positive)：预测当前页面是 Webshell，实际上不是 Webshell。

TN(True Negative)：预测当前页面不是 Webshell，实际上不是 Webshell。

Webshell 的评价标准主要有 4 个：特定度(Specificity)、灵敏度(Sensitivity)、准确度(Accuracy)和 F1 值[21]。其定义分别如下：

$$Specificity=\frac{TN}{TN+FP} \tag{4}$$

$$Sensitivity=\frac{TP}{TP+FN} \tag{5}$$

$$Accuracy=\frac{TN+TP}{TN+FN+TP+FP} \tag{6}$$

$$F1=\frac{2*Precision*Recall}{Precision*Recall} \tag{7}$$

特定度是指正确预测为正常 PHP 的页面占实际样例中所有正常的 PHP 页面的比例，衡量的是检测模型对正常 PHP 网页的识别能力；灵敏度指正确预测为 Webshell 的数量占实际样例中所有 Webshell 的比例，衡量的是检测模型对 Webshell 的识别能力；准确度是指正确预测的样本数占所有预测数量的比例，衡量的是模型对任意网页的识别能力；F1 值是对 *Precision*(TP/TP+FP)与 *Recall*(TP/TP+FN)的综合评价，其值越大，代表检测性能越好。敏感度、准确度、特定度越大，代表模型的检测能力越好。

5.3　实验结果

现从特定度、灵敏度、准确度、F1 值 4 个方面对基于 XGBoost 的 Webshell 检测方法(方法 1)、基于决策树的 Webshell 检测方法(方法 2)、基于朴素贝叶斯的 Webshell 检测方法(方法 3)、基于 KNN 值的 Webshell 检测方法(方法 4)进行比较，结果如表 3 所列。

表 3　不同的检测方法的实验结果

(单位：%)

方法	特定度	灵敏度	准确度	F1 值
方法 1	98.9	98.3	98.4	99.0
方法 2	96.0	98.9	98.3	98.3
方法 3	98.6	95.6	96.2	96.4
方法 4	95.7	98.9	98.2	98.2

由表 3 可知，在不同的评价标准上，有的方法在某一方面表现非常优秀；但综合 4 个方面进行比较，基于 XGBoost 算法的 Webshell 检测方法要明显优于其他 3 种方法。

结束语　本文详细分析了 Webshell 的定义和分类，描述了 Webshell 的特征；在此基础上，提出了一种基于 XGBoost 算法的 Webshell 检测方法；从特定度、灵敏度、准确度和 F1 值 4 个方面比较了基于 XGBoost 的 Webshell 检测方法、基于决策树的 Webshell 检测方法、基于朴素贝叶斯的 Webshell 检测方法和基于 KNN 的 Webshell 检测方法。实验结果表明，本文提出的方法在 Webshell 检测方面，各项指标都较高。在威胁情报快速发展的情况下，依靠一种检测方法很难进行全面检测，这就导致在真实的环境中检测时需要借助传统的检测方法来进行辅助分析；未来可以将研究的重点放在语言的底层上来应对 Webshell 的灵活多变。

参 考 文 献

[1] 张红瑞. Webshell 原理分析与防范实践[J]. 现代企业教育，2013(20)：254-255.

[2] 2016 年中国互联网网络安全报告[R/OL]. http://www.cert.org.cn/publish/main/upload/File/2016_cncert_report.pdf.

[3] 胡建康，徐震，马多贺，等. 基于决策树的 Webshell 检测方法研究[J]. 网络新媒体技术，2012，1(6)：15-19.

[4] 袁勋，吴秀清，洪日昌，等. 基于主动学习 SVM 分类器的视频分类[J]. 中国科学技术大学学报，2009，39(5)：473-478.

[5] YAO X. Large and Medium-sized Network Intrusions Cases Research[M]. Publishing House of Electronics Industry，2010：301-310.

[6] QUINLAN J R. C4.5：programs for machine learning[M]. San Francisco：Morgan Kaufmann，1993.

[7] HOU Y T，CHANG Y M，CHEN T H. Malicious web content detection by machine learning[J]. Expert Systems with Applications，2010，37(1)：55-60.

[8] 胡必伟. 基于贝叶斯理论的 Webshell 检测方法研究[J]. 科技广场，2016(6)：66-70.

[9] 安晓瑞. ASP 网站中 asp 一句话木马的安全性问题及防范措施的研究[J]. 首都师范大学学报(自然科学版)，2014，35(1)：39-43.

[10] OSUNA E，FREUND R，GIROSI F. An improved training algorithm for support vector machines[C]// Proceedings of IEEE Workshop on Neural Networks for Signal Processing. Amelia Island，USA：IEEE Press，1997：276-285.

[11] 谢清霞，于灏，于海妹，等. 重合指数的研究[EB/OL]. http://www.docin.com/P-147014653.html.

[12] 孟正，梅瑞，张涛，等. Linux 下基于 SVM 分类器的 WebShell 检测方法研究[J]. 信息网络安全，2014(5)：5-9.

[13] 叶飞，龚俭，杨望. 基于支持向量机的 Webshell 黑盒检测[J]. 南京航空航天大学学报，2015，47(6)：924-930.

[14] 贾文超，戚兰兰，施凡，等. 采用随机森林改进算法的 Webshell 检测方法[J/OL]. [2017-03-31]. http://www.arocmag.com/article/02-2018-04-056.html.

[15] DENG L Y，DONG L L，CHEN Y H，et al. Lexical analysis for the WebShell attacks[C]// The International Symposium on Computer，Consumer and Control，IEEE Computer Society. 2016：579-582.

[16] 石刘洋，方勇. 基于 Web 日志的 Webshell 检测方法研究[J]. 信息安全研究，2016，2(1)：66-73.

[17] NeoPI：Detection of web shells using statistical methods[EB/OL]. https://github.com/Neohapsis/NeoPI.

[18] A Gentle Introduction to XGBoost for Applied MachineLearning[EB/OL]. https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.

[19] XGBoost：A Scalable Tree Boosting System[EB/OL]. https://arxiv.org/abs/1603.02754.

[20] 机器学习中的算法(1)-决策树模型组合之随机森林与 GBDT[EB/OL]. http://www.cnblogs.com/LeftNotEasy/archive/2011/03/07/random-forest-and-gbdt.html.

[21] 李航. 统计学习方法[M]. 北京：清华大学出版社，2012.