2018 年全国大学生信息安全竞赛

作品报告

作品名称：基于深度学习与集成学习的可配置 WebShell 检测系统

电子邮箱：leo_infosec@foxmail.com

提交日期：2018.06.05

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。

2. 作品报告采用 A4 纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5 倍行距。

3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)

4. 作品报告模板里已经列的内容仅供参考，作者可以在此基础上增加内容或对文档结构进行微调。

5. 为保证网评的公平、公正，作品报告中应避免出现作者所在学校、院系和指导教师等泄露身份的信息。

目　　录

摘要

随着信息技术的迅速发展，大量互联网新兴产业也开始出现并迅速崛起。例如熟知的 Web 应用系统，其广泛应用于社交、银行、购物等重要业务，给人们的生活和工作带来了极大的便利，影响了现实空间中多个方面（经济、文化、科技等），在当今的网络资产中占有很高的比例。然而，相关的安全威胁也不容小觑，系统的广泛受攻击面以及攻击技术的多样化，降低了 Web 应用被攻击入侵的门槛，严重影响了社会稳定甚至国家安全。

已有 WebShell 检测系统有着准确率低、部署复杂、实时性差及不可配置等常见缺陷，针对此现状，本系统采用了多个模块相互结合的方法，包括基于传统规则检测技术的静态检测模块、基于动态检测与深度学习的动态检测模块以及基于静态检测与集成学习的统计检测模块，应用了目前流行的 AI 算法，极大的提高了检测效率，此外，还允许用户对其进行自定义配置，方便了用户的自定义使用。

简言之，本系统采用多模块思路、应用 AI 算法、基于综合策略，实现了一个可配置、高效的 WebShell 检测系统。与已有检测系统相比，具有准确率高、实时性强、可自定义配置、部署简单等特点。

第一章 作品概述

1.1 背景分析

近年来，网络安全事故频发，网络安全的面临的威胁与挑战也越来越大。网络安全已经危害到民生安全、社会安全、经济安全、基础设施安全，乃至政治安全。

据 CNCERT 监测，仅在 2016 年，就有约 4 万个 IP 地址对我国境内的 8.2 万余个网站植入后门，其中约 3.3 万个 IP 地址（占全部 IP 地址总数的 84.9%）是来自于境外，他们通过植入网站后门的方式对其进行远程控制。进一步看，来自美国的 IP 地址最多（4618 个），其次是中国香港（2115 个）和俄罗斯（1255 个）。从控制我国境内网站总数来看，来自香港的 IP 地址控制数量最多，有 1.3 万余个，其次是来自美国和乌克兰的 IP 地址，分别控制了 9734 个和 8756 个网站[1]。

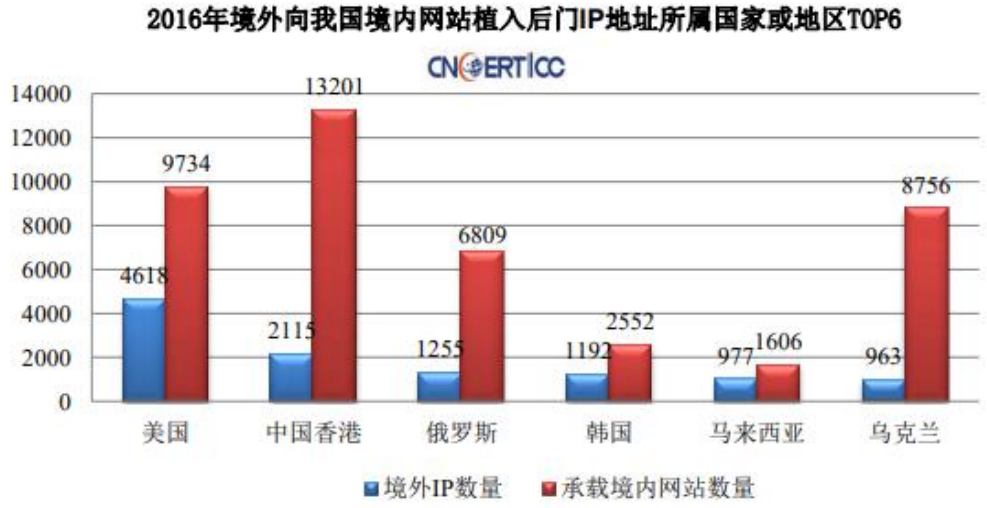

图 1.1 2016 年境外向境内植入后门国家或地区 TOP6

对于 WebShell 攻击，2017 年 1 月到 10 月间，360 网站卫士共为 187.5 万个网站拦截各类网站漏洞攻击 26.4 亿次，较 2016 年的 17.1 亿次，增长 54.4%，平均每天拦截漏洞攻击 868.9 万次。3 月、4 月是 2017 年攻击量最大的两个月，三月达到最高 6.3 亿次。网站每月遭遇漏洞攻击情况如下图所示。

图 1.2 2016-2017 网站每月遭遇漏洞攻击次数

下表给出了 360 网站卫士拦截漏洞攻击次数最多的 10 个漏洞类型。这十个类型共遭到攻击 22.0 亿次，占到漏洞攻击拦截总量的 83.4%[2]。

表 1.1 漏洞攻击 TOP10

TOP10	漏洞攻击类型	拦截次数（万）
1	SQL 注入	157682.9
2	Webshell	21615.1
3	通用漏洞	9402.9
4	nginx 攻击	7849.0
5	XSS	7491.3
6	扫描器	6385.4
7	信息泄露	3139.1
8	代码注入	2520.9
9	本地文件包含	2260.0
10	文件备份探测	2075.0

由表可见 WebShell 攻击类型排第二位且远超第三位，因此，针对 WebShell 攻击的防护迫在眉睫。

关于 AI 在安全方面的应用情况，在第三届世界互联网大会“世界互联网领先科技成果发布活动”现场，微软、IBM、谷歌三大国际科技巨头展示了基于机器学习的人工智能技术，描绘了人工智能美好的未来。目前，网络攻击事件层出不穷、手段多样、目的复杂，较为短缺的网络安全人才难以应对变化过快的网络安全形势，而由于机器学习在数据分析领域的出色表现，人工智能被认为在网络安

全方面将会“大有作为”。有研究机构统计发现，2016 年“网络安全”与“人工智能”两词共同出现在文章中的频率快速上升，表明越来越多的讨论将二者联系在一起。可见，在不久的将来，以网络安全相关的大数据为基础，利用机器学习等人工智能技术，能够在未知威胁发现、网络行为分析、网络安全预警等方面取得突破性进展。

1.2 作品简介

作品首先介绍了 WebShell 的概念及其简单分类，研究了 WebShell 检测和逃逸的思路与技术，说明了在当前 Webshell 检测领域中已有的技术成果，列举了多个应用机器学习及深度学习技术的具体检测系统，进而引出了对各个不同检测方案的辩证看待以及对未来的展望；随后介绍了如 LSTM、随机森林等与 Webshell 检测相关的具体技术；其次便是整个作品的实现设计与实现，包括总体的系统方案、数据的预处理、三个检测模块以及面对用户的 Web UI；而后便涉及到了作品实际的测试与评价，简单介绍了开发环境、实验数据及方案，并通过相关评价指标对最后的测试结果进行评价和分析；最后是对整个作品的创新性说明及总结，作品有其亮点也有其不足之处，虚心学习并不断完善才是正道。

静态检测模块、动态检测模块以及统计检测模块是本系统采用的三个检测模块，其中静态检测模块是基于传统规则检测技术的，动态检测模块则利用了深度学习方法，而统计检测模块结合了静态检测和集成学习。除了多个模块结合，为了提高检测效率，本系统还应用了当前流行的 AI 算法，用户的自定义配置也为用户的使用提供了较大的便利。

相比较于传统检测系统准确率低、检测效率低下等问题，本作品实现了一个基于多模块策略的高效 WebShell 检测系统。

1.3 作品特色

传统的检测思想会存在一定的不足，如：

1.误报率高

这是因为传统的检测大多采用规则针对日志进行操作。对于基于日志的检测方法存在弊端，比如，由于业务功能复杂且繁多，部分功能可能很少被用到，而

仅仅使用检测规则，也会造成更多的误报；

2.检测效率低，漏报率高

大量的日志记录处理也会对服务器性能造成一定负担，而且由于日志量巨大，检测过程消耗时间长、检测速度较慢也是无法避免的。同时，现有检测技术往往针对普通脚本后门（以控制服务器为目的、通常包含较为明显的静态特征或行为模式），对窃密型后门并不能进行有效检测。窃密型 WebShell 后门，往往具有以下特征：模拟正常的数据库操作、不具有较为明显静态特殊属性；被访问的次数比较少，无法形成较为明显的访问特征，因此，通过日志分析的手段也很难发现其存在。

与传统技术比较，本系统具有如下显著特色：

1.准确率高

针对一些业务系统更新频繁，Web 脚本文件相关属性经常变化的特点，本系统使用多种检测技术，从多个维度多种检测方式，运用深度学习与集成学习相结合的方式对 WebShell 进行检测，达到了较高的准确率。

2.实时性强

本系统采用训练后的集成学习、深度学习和规则模型对 WebShell 进行检测，检测时间因文件数目、文件大小及主机配置而异，测试发现对于一般 PC 及 10M 样本总大小，运行时间可以达到秒级。

3.独立性强

本系统对于 WebShell 的检测不需要上下文信息，只需提供待检测的 php 文件即可运行，实现了较高的独立性。

4.可配置

本系统不同于已有 WebShell 检测系统，可以对三个检测模块进行自定义配置，并且可引入自己定制的规则，具备一定的可配置性。

5.部署简单，运行稳定

本系统部署简便，没有特殊的硬件环境限制。采用 B/S 架构提供服务，方便易用。此外，系统采用技术手段处理基于动态行为的检测，并且经过实际运行测试，能够满足系统稳定性需求。

1.4 应用市场

1.公司企业设备

公司企业一直以来都是网络攻击的对象，其庞大的内网资源更是黑产梦寐已求的目标。由此来看，企业频繁遭受 Web 攻击（WebShell 攻击）的现象也不足为怪，故企业公司更应加强此类防护，防止被入侵而造成重大经济损失。

2.网络安全公司产品

将本系统集成于网络安全软件中，可以有效的提高其对 WebShell 的防护能力；同时，由于本产品采用集成学习及深度学习等技术，网络安全公司可以提供大量样本供系统训练学习，所以可以进一步提升产品识别准确性。

3.党政机关和生产建设关键部门

没有网络安全就没有国家安全，党政机关和生产建设关键部门历来是敌对势力的重点攻击目标，Web 攻击而导致的机密文件泄露，会给党和国家造成重大损失，因此，党政机关和生产建设关键部门更应该重点加强此类防护，以保证自身信息系统安全。

第二章 研究现状与相关技术

2.1 WebShell 简介

WebShell 就是以 asp、php、jsp 或者 cgi 等网页文件形式存在的一种命令执行环境，通俗来讲，就是一种网页版后门。WebShell 后门文件往往会与 Web 目录下的正常网页文件混在一起，没有经验的网站管理员很难发现其存在。恶意用户在入侵了一个网站后，通常会上传 WebShell 文件至服务端，然后就可以使用浏览器来访问 WebShell，得到一个命令执行环境，从而控制目标网站服务器，也为后续的提权操作做准备[3]。

通常来说，WebShell 可以简单分为三类：小马、大马和一句话木马。

小马：体积小，支持功能少，一般只具备特定功能，如文件上传功能，常服务于上传大马。

大马：体积大；支持的功能全面，包括文件管理、命令执行、数据库操作等；可通过代码加密等手段使其具有一定隐蔽性。

一句话木马：代码简短，通常只有一句代码，结合菜刀等 WebShell 管理工具使用；基于 B/S 架构，仅用于向服务端提交控制数据；使用灵活，既可单独使用也可嵌入其它正常文件。

2.2 WebShell 检测与逃逸

2.2.1 检测思路

笼统来讲，WebShell 检测思路有两种：静态检测与动态检测。WebShell 要实现其功能，其代码中必然有特定关键字和恶意函数等特征，从文件代码入手进行的基于特征的检测即是静态检测。在 WebShell 运行后，检测 WebShell 的字节码及其通信的即是动态检测。

基于静态特征的检测，不需要执行此文件，仅从文件代码层面入手。WebShell

要实现某些特定功能，必然离不开某些特定关键字及敏感函数，静态特征检测是最基础，同时也是最常见的一种检测技术，有些高级的检测技术，会涉及语法语义分析。静态检测优点对已知的 WebShell 查找准确率高，且实施方便，通常一个脚本便可部署好。静态检测缺点也比较突出，就是需要收集某些特征码、特征值、关键字及危险函数，进而建立一个恶意字符串特征库，在检测时将文件内容与之匹配，导致其只能检测已知的 WebShell，存在误报和漏报的问题（误报：特征库中的某些敏感函数，程序的正常功能也会用到。比如 PHP 里面的 eval、system 等，ASP 里面的 FileSystemObject、include 等。漏报：本检测思想的关键是建立的恶意字符串特征库，特征库中只会收录已知 WebShell 的特征，若新出现一种 WebShell，其特则字符串并未收录到库中，便不会检测到，黑名单思想的弊端便是如此）。

动态检测则利用 WebShell 执行时的动态特征，将 WebShell 上传到服务器，总得需要执行以实现功能，因此其在执行过程中就会表现出来一系列的动态特征。例如对于动态脚本语言 PHP，其在 zend 虚拟机执行过程为：读入脚本程序字符串，经由词法分析器将其转换为单词符号，接下来语法分析器从中发现语法结构后生成抽象语法树，再经静态编译器生成 opcode，最后经解释器模拟机器指令来执行每一条 opcode。Opcode 是计算机指令中的一部分，运行 php 脚本，分析 opocode 是一种通用的动态检测方法。另外，WebShell 通信基于 HTTP 协议，执行会有特定的 HTTP 请求和响应，基于此，同样可以采取黑名单思想，将其有的 HTTP 请求和响应做成特征库，部署到 IDS 中检测所有 HTTP 请求即可。但是因为是黑名单思想，此方法也会有漏报的问题，因此保持特征库的更新便显得尤为重要。除了 HTTP 层面，若 WebShell 执行系统命令，会在服务器产生进程，如在 Linux 下 nobody 用户启动了 bash，Windows 下 IIS User 启动 cmd，诸如此类，皆是动态特征。

2.2.2 检测模式

除了静态检测和动态检测两大检测思路，还可以从其他角度分为三种检测模式：基于流量模式、基于 agent 模式和基于日志分析模式。

基于流量模式即对应于动态检测思路，基于 agent 模式即对应于静态检测思

路（检测思路中已有涉及，不再赘述）。

基于日志分析模式，可以这样理解：使用 WebShell 一般不会在服务器的系统日志中留下记录，但是 WebShell 页面的访问及其数据提交记录会被留在网站的 Web 日志中。日志分析检测技术即通过大量的日志文件来建立请求模型，从而检测出异常文件。因此，基于日志分析模式的 WebShell 检测又可称为 HTTP 异常请求模型检测。

除此之外，数学中的统计学方法也广泛应用于 Webshell 后门检测中，如 NeoPi 是国外流行的一个基于统计学的 Webshell 后门检测工具，它使用五种统计学方法在脚本文件中搜索潜在的被混淆或被编码的恶意代码。

2.2.3 逃逸技术

静态检测、动态检测、统计分析以及日志检测是 WebShell 测技术中几种常见思路，而其逃逸技术也纷繁众多，常见的如字符串加密、字符串拆分构造及动态函数调用。通常，逃逸技术可主要分为两种：隐蔽位置传递载荷（如 UA 字段实现载荷传递）和构造法绕过检测（如动态函数调用和字符串拼接）[4]。若对其进行细化，可以大致分为以下几种方法[5]：

字符串加密方法：将文件中的敏感字符串（如敏感函数名）进行加密处理，通常做法是将其可逆的映射到另一个字符串以逃避检测，在对其访问时再进行解密处理。这种方式广泛应用于免杀 Webshell 中，如 PHP 语言内置的 base64、rot13 加密函数。若要进一步提高逃逸的成功率，还可以自定义加解密函数，因为各编程语言中的内置加解密函数往往也在敏感字符串的查杀范围内。

字符串拆分构造方法:利用 ASCII 码中两个不同的字符运算可得到新字符的特性，再结合字符串拼接即可完成代码混淆。如通过对“#”、“|”等特殊符号的异或计算，可以得到敏感字符串“POST”。基于此种思想，可以构造出相应的 webshell 文件。但是值得注意的是，此种逃逸方式仅仅停留在表面，后门文件运行时终究是要到底层的，其编译到底层的 opcode 都会表现出同样的结构形式，所以基于语义层面的检测技术能够有效应对这种逃逸方式。

流量加密方法：流量加密是一种能有效对抗流量检测的逃逸技术，与字符串加密技术不同的是，加解密的对象是与后门文件通信时产生的流量，而非文件内

容中具有敏感特征的字符串和函数名。

文件包含方法：一个 WebShell 文件可以拆分证多个文本或图片，通过文件包含的方式再进行整合，这样一来，后门文件中的恶意特征被分散到多个不同的位置，从而影响检测模型的评估，降低被发现的概率。

其它逃逸方法：逃逸技术纷繁众多，一一列举是不现实的，除了上述几种常用的逃逸方法外，还包括在隐蔽位置隐藏攻击载荷、加载其他脚本组件以及无文件攻击等方法。而且，各方法也并非单独使用的，多种逃逸方法相互结合，可进一步提高逃逸的成功率。

2.3 研究现状

传统的检测思想会存在一定的不足，比如：由于业务系统更新频繁，Web 脚本文件相关的属性会经常发生变化，所以偏重于文件属性检测的方法往往会产生更多的误报。基于动态行为检测的方法往往技术难度较大，难以实现，而且会对系统性能造成较大影响，甚至危及系统稳定性。而对于基于日志的检测方法，大量的日志记录处理也会对服务器性能造成一定负担，而且由于日志量巨大，检测过程消耗时间长、检测速度较慢也是无法避免的。

随着机器学习与深度学习的发展，其也越来越广泛的应用到 Webshell 检测领域。深度学习方面，如胥小波和聂小明二人提出一种基于多层感知器（MLP）神经网络的 WebShell 检测方法[6]，该方法通过 TF-IDF 计算词频矩阵，并筛选特征以得到训练样本集的特征矩阵，最后通过多层神经网络训练得到检测模型。可以很好的应对复杂灵活的未知及变种 WebShell。

虽然机器学习在某些方面不及深度学习，但是其也频繁出现于 webshell 检测中，比如：（1）潘杰在其硕士论文中提出了改进的单类支持向量机检测模型[7]，通过改进单类支持向量机的决策函数来降低误报率，该方法本质上是一种基于机器学习的 WebShell 检测技术。（2）戴桦等人提出一种智能检测 WebShell 的机器学习算法[8]，学习已知 WebShell 的页面特征，可完成对未知页面的预测。（3）叶飞等人基于对 WebShell 对应的 HTML 页面特征的分析，提出了一种基于支持向量机（SVM）分类算法的黑盒检测方法[9]，该方法是一种有监督的机器学习系统，可以在未知脚本源代码的情况下对 WebShell 进行检测。（4）机器学

习方面的应用还是比较多的，除上述之外，贾文超等人还提出一种基于随机森林改进算法的 Webshell 检测方法[10]，此方法改进了随机森林特征选取算法，并且对三种类型的 Webshell 进行深入特征分析，构建好多维特征，可以比较全面的覆盖静态属性和动态行为，解决了 Webshell 检测特征覆盖不全的问题。

其它具体的检测技术也有许多，如孔德广等人提出的基于 Simhash 算法的方法[11]、周颖和胡勇的基于关联分析的方法[12]、易楠等人的基于语义分析的检测方法[13]、Truong Dinh Tul 等人提出的基于最优阀值思想的检测方法[14]以及王文清等人构建出的基于组合策略的检测框架[15]等。

综上，每一种思想下的具体检测技术都不可能是完美无缺的，都会有各自的适用场景，没必要对每一种方法吹毛求疵。

对于 WebShell 检测技术及其逃逸技术，二者本身就是相互博弈的过程，可以预见的是，随着 WebShell 逃逸技术变得越来越复杂，其传递的数据流也就越复杂，从而与正常程序的复杂度产生较大差异，也就为检测技术提供新思路；同时，在被动检测的基础上，利用蜜罐技术、社会工程学、指纹跟踪等技术来可以实现主动的 WebShell 溯源；最后，机器学习与深度学习是一个大趋势，其必将更多的应用到检测领域。

2.4 相关技术

2.4.1 RNN

神经网络是一种应用类似于大脑神经突触联接的结构进行信息处理的数学模型，是 20 世纪 80 年代以来人工智能领域兴起的研究热点。它从信息处理的角度对人脑神经元网络进行抽象，建立某种简单模型，按不同的连接方式组成不同的网络。网络内部由大量节点（或称神经元）相互联接构成，每个节点代表一种特定的输出函数，称为激励函数（activation function）。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出则依网络的连接方式、权重值和激励函数的不同而不同。

在传统的神经网络中，假设所有的输入（包括输出）之间是相互独立的。若要预测一个序列中的下一个词，最好能知道哪些词在它前面。RNN 之所以是循环

的，是因为它针对系列中的每一个元素都执行相同的操作，每一个操作都依赖于之前的计算结果。换言之，可以认为 RNN 记忆了已经计算过的信息[16]。

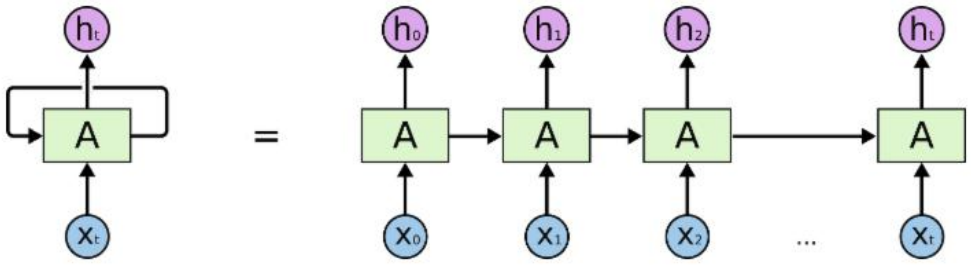

图 2.1 RNN 的结构

2.4.2 LSTM

LSTM 神经网络于 1997 年首先被 Sepp Hochreiter 和 Jurgen Schmidhuber 提出，与传统神经网络相比优势在于其会持续思考，能够记住长周期有效信息，可以明确的将前面事件传递给后面的事件，以让信息持续传递下去，而不需要很多的上下文语境就能分析出结果。

LSTM 神经网络是一种特殊的 RNN（递归神经网络），可以解决长期依赖问题。所有 RNN 都具有一种重复神经网络模块的链式形式，在标准的 RNN 中，这个重复的模块仅有一个非常简单的结构，例如一个 tanh 层[17]。

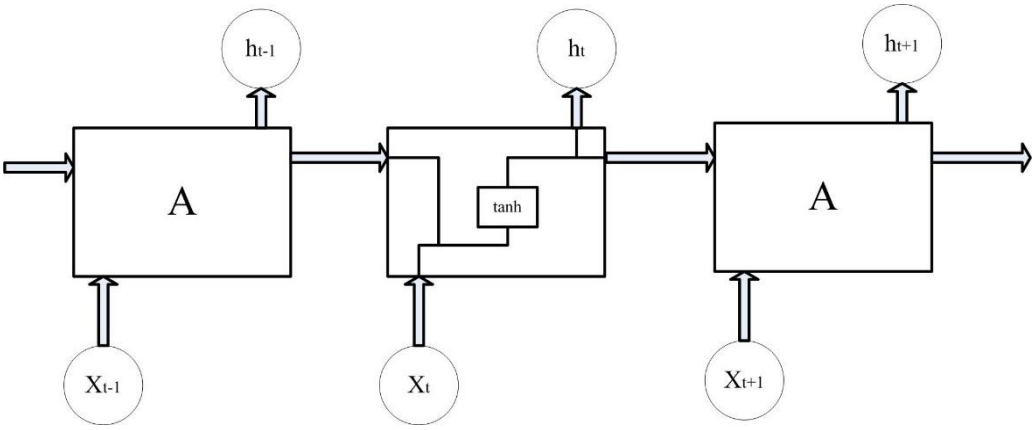

图 2.2 标准 RNN 中的重复模块包含单一的层

LSTM 同样是这样的结构，但是重复的模块拥有一个不同的结构。不同于单

一神经网络层，这里是有四个 tanh 层，以一种非常特殊的方式进行交互。

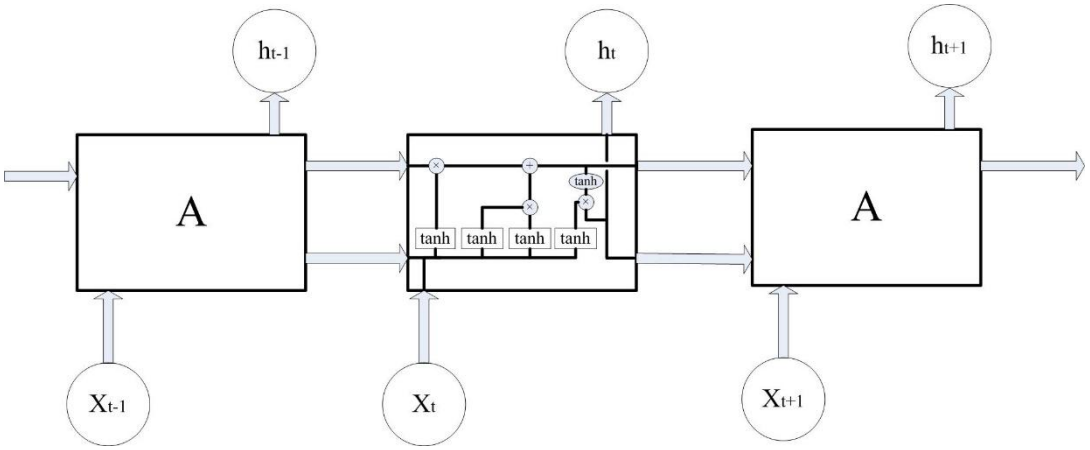

图 2.3 LSTM 中的重复模块

其中四个交互的层最关键的是细胞状态，即模型最上方的水平线：

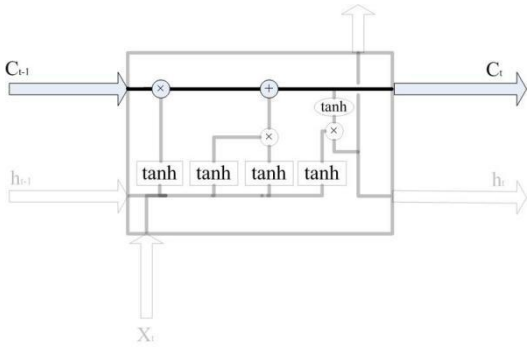

图 2.4 细胞状态（黑线标记）

细胞状态主要作用就是保持信息的传递性，只有少量交互。同时 LSTM 有三个门来控制细胞状态：

忘记门层（确定丢弃信息）：

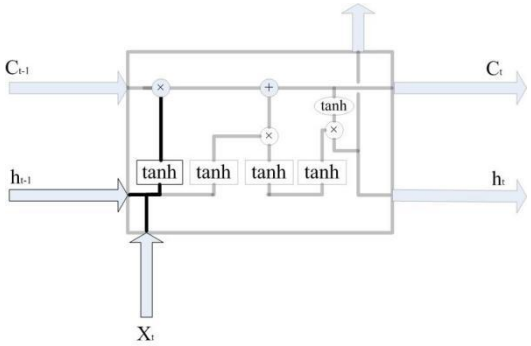

图 2.5 忘记门层（黑线标记）

输入门层（确定更新信息）：

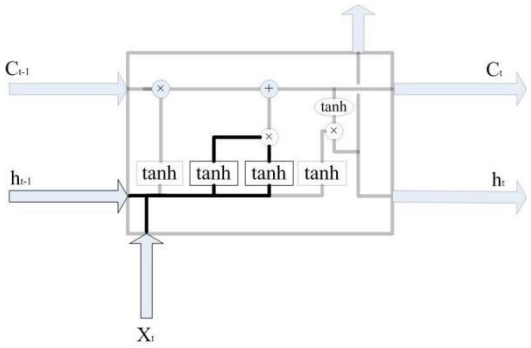

图 2.6 输入门层（黑线标记）

更新门层（更新细胞状态）：

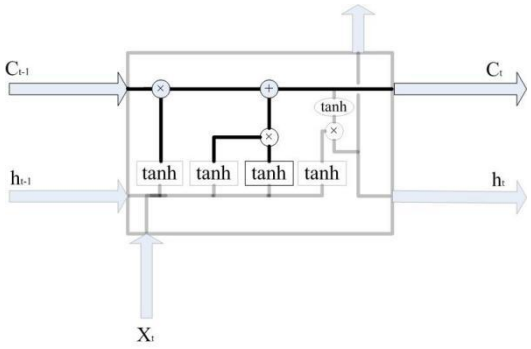

图 2.7 更新门层（黑线标记）

2.4.3 集成学习

集成学习将多个学习器进行结合，通常可获得比单一学习器更加显著的泛化性能，对“弱学习器”尤为明显（弱学习器常指泛化性能略优于随机猜测的学习器）。集成学习的结果通过投票法产生，即“少数服从多数”。个体学习不能太坏，并且要有“多样性”，即集成个体应“好而不同”。

假设基分类器的错误率相互独立，则由 Hoeffding 不等式可知，随着集成中个体分类器数目 T 的增大，集成的错误率将指数级下降，最终趋向于零。但是这里有一个关键假设：基学习器的误差相互独立，而现实中个体学习器是为解决同一个问题训练出来的，所以不可能相互独立。因此如何产生并结合“好而不同”的个体学习器是集成学习研究的核心[18]。

集成学习大致分为两大类：

个体学习器间存在强依赖关系，必须串行生成的序列化方法。代表：Boosting

个体学习器间不存在强依赖关系，可同时生成的并行化方法。代表：Bagging

1、Boosting

Boosting 的主要思想：先从初始训练集训练出一个基学习器，再根据学习器的表现对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续受到更多关注，然后基于调整后的样本分布来训练下一个基学习器；如此重复进行直至基学习器数目达到事先指定的值 T，最终将这 T 个基学习器进行加权结合。

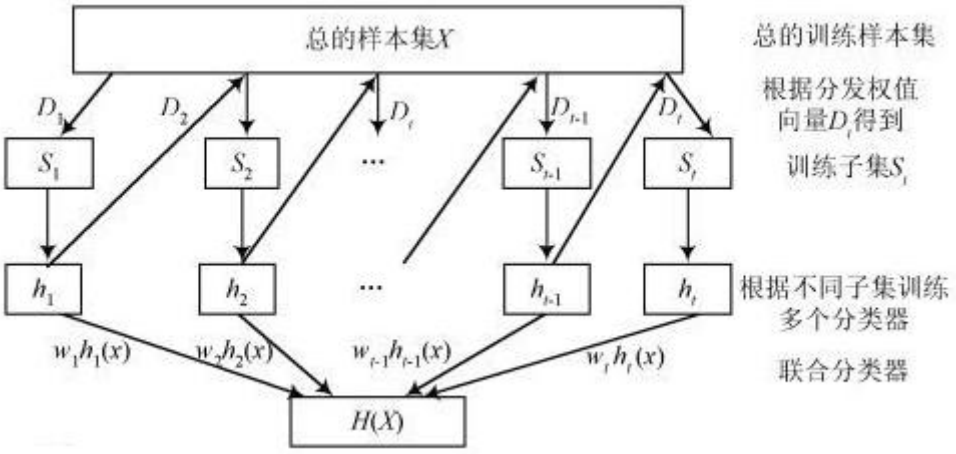

图 2.8 boosting

Boosting 族算法著名代表是 AdaBoost，AdaBoost 算法有多种推导方法，比较容易理解地是基于“加法模型”（additive model），即基学习器的线性组合来最小化指数损失函数：

$$H(x) = \sum_{t=1}^{T} \alpha_t \mathrm{h}_t(x)$$

$$l_{exp}(H|D) = E_{x-D}\left[e^{-f(x)H(x)}\right]$$

若指数损失函数最小化，则分类错误率也将最小化；这说明指数损失函数是分类任务原本 0/1 算是函数的一致的替代函数，其有更好的数学性质即连续可微。

Boosting 算法要求基学习器能对特定的数据分布进行学习，这可通过“重赋权法”（re-weighting）实施。对无法接受带权样本的基学习算法，则可通过“重采样法”（re-sampling）来处理。若采用“重采样法”，则可获得“重启动”机会以避免训练过程过早停止。可根据当前分布重新对训练样本进行采样，再基于新的采样结果重新训练基学习器。

从偏差-方差分解的角度看，Boosting 主要关注降低偏差（即降低错误率，偏

差反映学习算法的拟合能力），因此 Boosting 能基于泛化性能相当弱的学习器构建出很强的集成。

2、Bagging

欲得到泛化性能强的集成，集成中的个体学习器应尽可能相互独立；无法做到独立，也可设法使基学习器尽可能具有较大的差异：一种是对训练样本进行采样，产生不同子集，每个子集训练一个基学习器，但如果采样的子集不足以进行有效学习，就无法确保产生比较好的基学习器，因此可以考虑使用相互有交叠的采样子集。

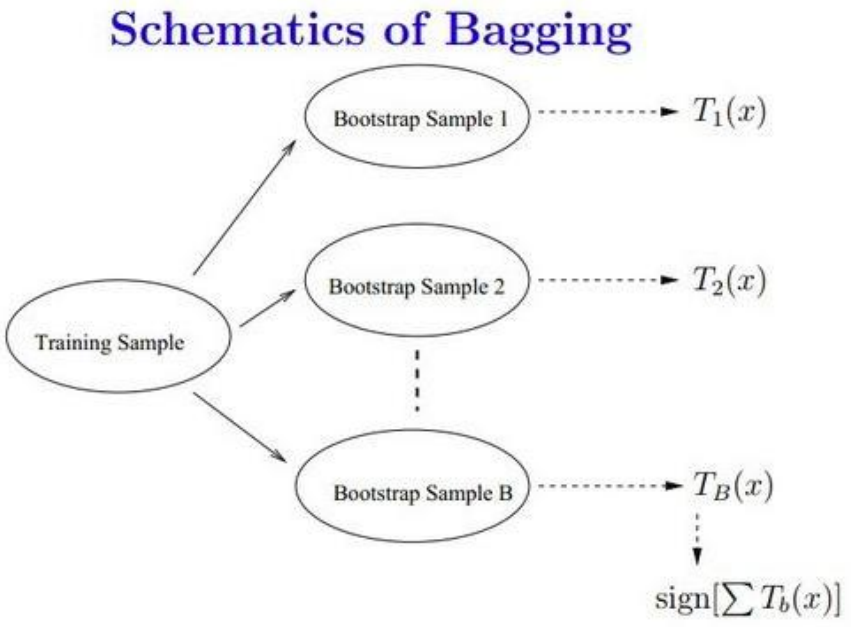

图 2.9 bagging

Bagging 是并行式集成学习方法最著名的代表，Boostrap 的意思是其基于自助采样（bootstrap sampling）。即给定包含 m 个样本的数据集，先随机取出一个样本放入采样集中，再把该样本放回初始数据集，使得下次采样时该样本仍有可能被选中，这样经过 m 次有放回随机采样操作，即可得到含 m 个样本的集合（有的样本重复出现，有的样本从未出现，大体约有 63.2%的样本出现在采样集中）。照此，可以采样出 T 个含 m 个训练样本的采样集，然后基于每个采样集训练一个基学习器，再将这些基学习器进行结合。

从偏差-方差分解的角度看，Bagging 主要关注降低方差（方差度量同样大小的数据集的变动所导致的学习性能的变化）。当基学习器不稳定（large varance）

时，Bagging 带来的性能提升尤为明显。因此它在不剪枝决策树、神经网络等易受样本扰动的学习器上效用更为明显。

2.4.4 随机森林

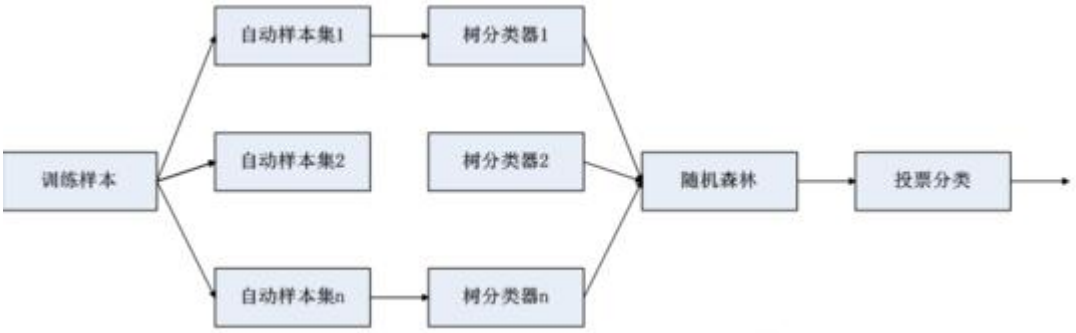

图 2.10 随机森林

随机森林（RF, RandomForest）包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而定。通过自助法（boot-strap）重采样技术，不断生成训练样本和测试样本，由训练样本生成多个分类树组成的随机森林，测试数据的分类结果按分类树投票多少形成的分数而定。随机森林是 Bagging 的一个扩展变体，其进一步在决策树的训练过程中引入了随机属性选择[19]。

建立每棵决策树的过程中，有 2 点：采样与完全分裂。首先是两个随机采样的过程，RF 要对输入数据进行一下行（样本）、列（特征）采样，对于行采样（样本）采用有放回的方式，也就是在采样得到的样本中可以有重复。从 M 个特征中（列采样）出 m 特征。之后就是用完全分裂的方式建立出决策树。RF 的随机性体现在每棵树的训练样本是随机的，树中每个节点的分类属性也是随机选择的，有了这两个随机的保证，RF 就不会产生过拟合现象。

优点：

1.很多的数据集上表现良好；

2.能处理高维度数据，并且不用做特征选择；

3.训练完后，能够给出那些 feature 比较重要；

4.训练速度快，容易并行化计算。

缺点：

1. 在噪音较大的分类或回归问题上会出现过拟合现象；

2. 对于不同级别属性的数据，级别划分较多的属性会对随机森林有较大影响，则 RF 在这种数据上产出的数值是不可信的。

第三章 作品设计与实现

3.1 系统方案

本系统采用了三个模块相互结合的方式进行检测，分别是：基于传统规则检测技术的静态检测模块、基于动态检测与深度学习的动态检测模块以及基于静态检测与集成学习的统计检测模块，多个模块相互结合，并且应用了目前流行的AI 算法，极大的提高了检测效率，除此之外，还允许用户对其进行自定义配置，方便了用户的自定义使用。系统整体架构如图 3.1：

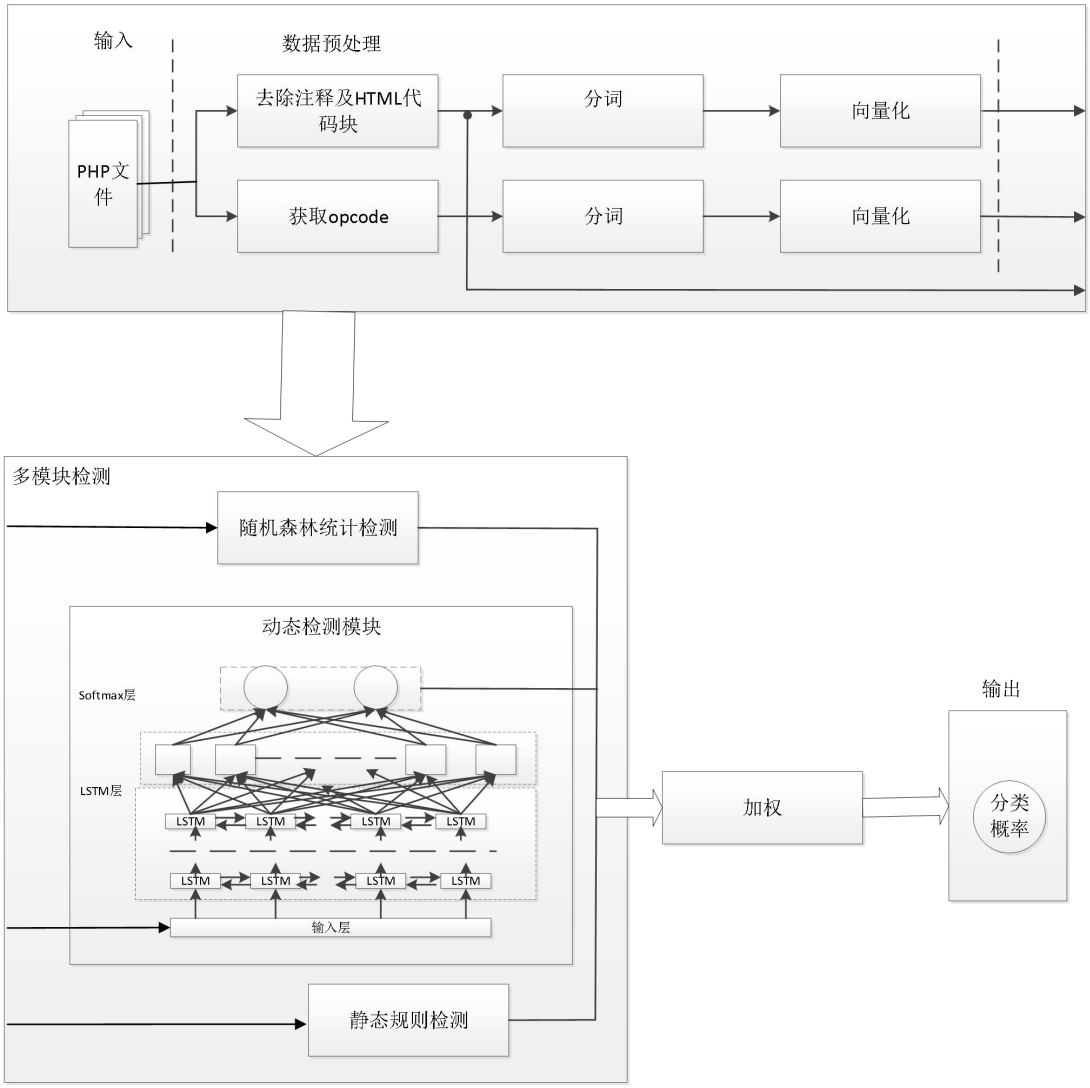

图 3.1 系统整体架构

1. 用户上传待检测 php 脚本，配置好三个检测模块的权重（可选），上传

自定义规则（可选）。

2. 用户上传的 php 脚本经过三个不同模块的检测，得到三组列表，每组列表中存放着单个 php 脚本为 WebShell 的概率。

3. 如果用户自定义了三个模块的权重，则按照用户自定义的权重来计算总概率，否则按照默认权重处理。

4. 输出文件名、对应的三个模块的概率以及总概率。

3.2 数据预处理

数据预处理模块包含三部分：静态检测、动态检测和统计检测模块数据处理。

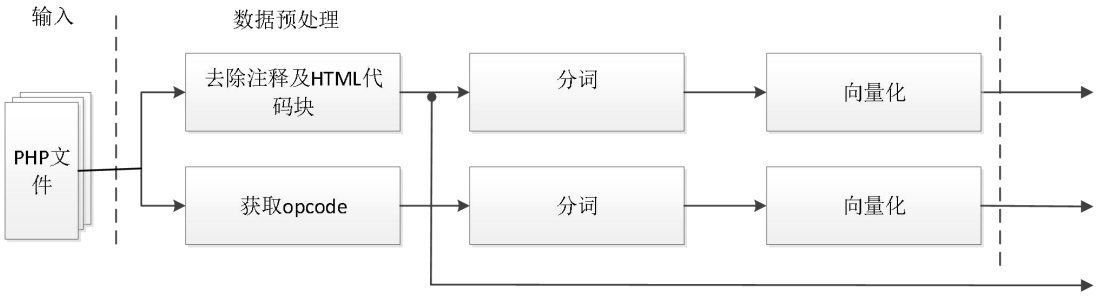

图 3.2 数据预处理

统计检测模块数据处理：去除样本中的注释和 html 代码块从而降低样本中的噪声，并进行编码处理保证样本编码相同便于下阶段统一处理，最后基于词袋 2-gram 和 TF-IDF 进行特征提取，从而得到特征向量。

动态检测模块数据处理：对于每个样本，运行命令“php_bin+" -dvld.active=1 -dvld.execute=0 "+file_path”以获取对应 php 样本的 opcode，再使用 2-gram 分词，进而得到特征向量。

静态检测模块数据处理：去除样本中的注释和 html 代码块从而降低样本中的噪声，从而进行规则匹配即可。

2-Gram 是词袋模型的一个细分类别，N-Gram 基于这样一种假设，第 n 个单词之和它前面的 n-1 个词有关联，每 n 个单词作为一个处理单元。通过设置 CountVectorizer 函数的 ngram_range 参数和 token_pattern 即可实现 N-Gram，其中 ngram_range 表明 N-Gram 的 n 取值范围，如果是 2-Gram 设置成(2,2)。

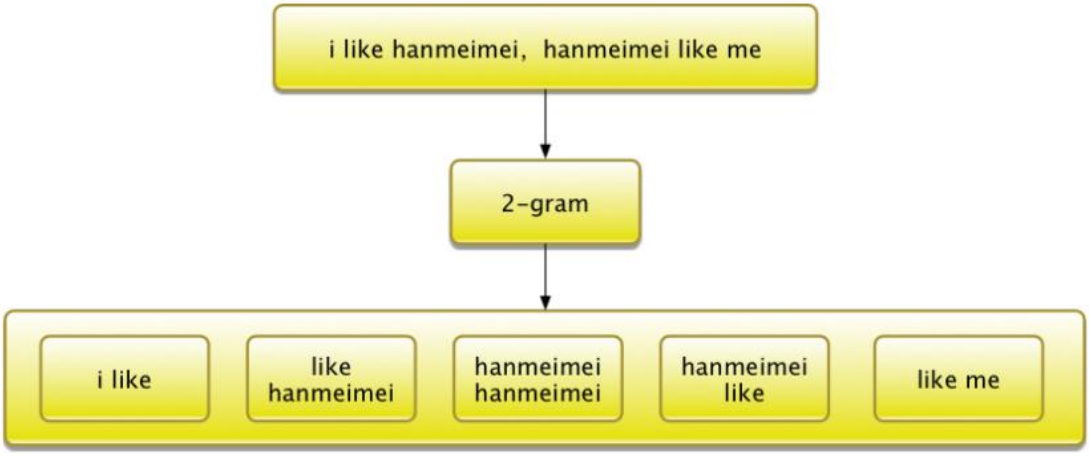

图 3.3 2-Gram

opcode 是计算机指令中的一部分，用于指定要执行的操作， 指令的格式和规范由处理器的指令规范指定。除了指令本身以外通常还有指令所需要的操作数，可能有的指令不需要显式的操作数。这些操作数可能是寄存器中的值，堆栈中的值，某块内存的值或者 IO 端口中的值等等。通常 opcode 还有另一种称谓：字节码(byte codes)。 例如 Java 虚拟机(JVM)，.NET 的通用中间语言(CIL: Common Intermeditate Language)等等。PHP 中的 opcode 则属于前面介绍中的后者，PHP 是构建在 Zend 虚拟机(Zend VM)之上的[20]。

TF-IDF(Term Frequency-Inverse Document Frequency, 词频-逆文件频率).是一种用于资讯检索与资讯探勘的常用加权技术。TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。总结就是，一个词语在一篇文章中出现次数越多，同时在所有文档中出现次数越少，越能够代表该文章。

IDF 公式如下：

$$\mathrm{IDF} = \log \frac{\text{语料库的文档总数}}{\text{包含词条 w 的文档数} + 1}$$

分母之所以要加 1，是为了避免分母为 0。

TF-IDF 的公式：

$$\mathrm{TF} - \mathrm{IDF} = \mathrm{TF} * \mathrm{IDF}$$

3.3 静态检测模块

静态特征是对 WebShell 的静态特征检测，采用模块化实现。每一种检测方法都对应于一个函数，所有函数（即检测方法）位于同一文件，其作为一个整体模块导入至主文件中，可对某一特定目录进行扫描检测，包括其子文件夹下的所有文件。

对于每一个待检测文件，遍历自定义扫描模块中所有的方法，依次对此文件进行扫描检测。除此之外，还提供一用户自定义接口，用户可将自己遇到的 WebShell 特征字符串以正则表达式的格式提交给程序，程序接收用户提交的规则并保存，再次检测时，便可以应用此新规则进行检测。

因为是模块化实现，故可扩展，若学习得新的检测思想，可自行添加至扫描模块，作为一新的扫描检测方法，后期可不断完善，因此保持检测方法的不断更新是很有必要的。目前程序已有近十种检测方法，可查杀目前常见的大部分 WebShell。对于新出现的 WebShell，可以通过手动添加以弥补其不足。

接下来列出的是系统采用的几种典型的扫描检测方法：

1-敏感关键字检测方法：

对应于 scan_modules 模块中 keywords 方法。一些 WebShell 中会含有敏感关键字，如“serv-u、phpspy、WebShell、cmd.exe “等，通过 python 读取待检测文件内容，遍历所有敏感关键字，确定其内是否包含，以检测是否为 shell 文件。关键字检测方法仅为一个大体思想，除了检测特定关键字，还会有一些已知 shell 文件的特征，如” r57shell “；也会有针对 eval、assert 等敏感函数的正则表达式匹配，如[\'"]e[\'"]\.[\'"]v[\'"]\.[\'"]a[\'"]\.[\'"]l[\'"]，程序都会对其进行遍历，检测文件内容中是否匹配有这些特征，从而做出判断。

除此之外，还会有一个组合特征查找，若关键字“cmd.exe“和”program files “同时出现在文件内容中，基本上就可以判断此文件为一个包含敏感关键字的 shell 文件。

对于一些加密后的 shell 文件，其文件中可能会含有相关加密信息，如在线 php 加密网站” http://www.phpjm.net/”和 php 神盾加密。其实，这也是属于已知 shell 特征的范畴。

2-DDOS 检测：

对应于 scan_modules 模块中 ddos 方法。

提供一专门针对 ddos 的 shell 脚本检测方法，内置一常见关键字列表，如“xxddos、phpddos、fsockopen("udp:“，为防止误报，建立一白名单，如某些程序会存在文件” install/svinfo.php “，其内也会有 ddos 的敏感代码” fsockopen("tcp:' “，应不予考虑。

3-动态函数检测：

对应于 scan_modules 模块中 dynamic_func 方法

PHP 中，函数名称可以当成字符串来动态改变，最常见的形式为“$_GET[a]($_POST[b]) “，a、b 均由 get 方法动态接收，a 为函数名，b 为参数。因为可动态改变，故查杀关键字较难定位。本方法提供多种检验规则，并提取变量且追踪，通过一系列逻辑判断看其是否可控，可以很好的对此类后门进行检测。

4-回调后门检测：

对应于 scan_modules 模块中 array_map 方法。

在 PHP 中，利用包含回调函数参数的函数来做后门是比较隐蔽的，如很常见的 call_user_func 函数，call_user_func('assert', $_REQUEST['pass']);，assert 直接作为回调函数，然后$_REQUEST['pass']作为 assert 的参数调用。类似的还有“call_user_func_array、array_filter、array_map “。以函数 array_map 为例，程序提供一检测此函数的规则，第一步先初步检测文件内容中是否存在有 array_map 的关键字，若存在，进一步利用此规则生成的正则表达式来查找，若利用正则仍可检得，则基本可以判断，这是一个利用 array_map 函数的回调后门。

5-eval 后门检测：

对应于 scan_modules 模块中 eval_assert 方法。

eval 与 assert 等函数可用于执行具体代码，可视为最常见最简单得一句话后门函数，其中 eval 函数中参数是字符，如：eval('echo 1;'); assert 函数中参数为表达式 （或者为函数），如：assert(phpinfo())。对于一些基本一句话后门文件，直接对此类函数检验是极有效率的。最原始的方法是直接在文件内容中搜寻，若搜得，则其可能是一个 shell 文件，但是也会有例外，某些应用程序的正常文件中也可能应用此类函数以完成某些特定功能，因此简单的字符串搜索是远远不够的，为提高准确性，可通过编写正则表达式，通过正则来进一步判断，看此类函

数接受的参数是来自与程序内部还是外部变量，如$_GET、$_POST。有些情况下，其参数是经过 base64 加密的，因此编写正则时，要考虑到 base64_decode 等特定函数。

6-preg_replace 检测：

对应于 scan_modules 模块中 preg_replace 方法。

preg_replace 函数执行一个正则表达式的搜索和替换。函数原型如下：mixed preg_replace (mixed pattern, mixed replacement, mixed subject [, int limit])，特别注意的是，/e 修正符能够使 preg_replace() 将 replacement 参数当作 PHP 代码来执行，这也为此类后门提供了思路。例如，文件中存在代码：“echo preg_replace("/test/e",$_GET["h"],"jutst test"); “，若浏览器中提交?h=phpinfo()，phpinfo()将会被执行，因为使用/e 修饰符，preg_replace 会将 replacement 参数当作 PHP 代码执行，由此可见，修饰符 e 的危险性。

与其它检测方法同样的思想，系统首先通过字符串查找，在文件内容中搜寻是否含有 preg_replace 函数，再进一步使用正则，判断其是否含有 e 修饰符以及其它 shell 特征，如接收外部输入的$_GET、$_POST。

7-打包 shell 检测：

此方法对应于 scan_modules 模块中 packshell 方法。有些 WebShell 有特定的功能，如打包网站所有文件，从而得到网站源码以便审计。打包文件的 WebShell 通常会用到特定的函数，如 PHP 中的用于压缩文件的函数 gzencode、gzdeflate 和 gzcompress，除了特定的压缩函数，还会有一些敏感关键词会出现在 shell 文件中，如“打包、unix2DosTime “。利用正则化或直接在文件内容中搜索，可以完成检测任务。

8-文件包含 shell：

此方法对应于 scan_modules 模块中 include_file 方法。文件包含即意味着执行该文件，常用的文件包含函数如 include、require、include_once、require_once，若一脚本文件，通过此类函数包含引入了一个 shell 文件，则仍会达到执行此 shell 的文件，形象来说，此文件即为“帮凶“。正常文件经常会使用到文件包含功能，但值得注意的是，正常脚本文件包含的文件一般是程序内部的其它正常文件，即来源为内部，若外部输入可做为文件包含的来源，如 get 方式或 post 方式接收到

的用户输入，举例说明，1.php?file=shell.php，通过 1.php 来动态包含 shell 脚本文件。检测思路也比较见到那，搜索文件内容，看是否含有文件包含功能的相关函数，再进一步追溯其包含的文件，看是否用户可控，若可控，即外部输入作为的待包含的文件，则一定要多加留意。

9-用户自定义扫描 shell：

此方法对应于 scan_modules 模块中 customize 方法。基于静态检测的程序总会存在漏报的问题，若用户遇到新的 WebShell，可以自行将其特特征字符串提交给程序，程序接收并保存至一文件，作为用户自定义特征库，检测时，读取此库文件，与文件内容作对比，若符合规则，也可以完成检测，这其实是一种敏感字符串匹配思路。通过收集用户自定义规则，可以完善程序检测方法，一定程度上解决漏报问题。

3.4 统计检测模块

模型构建：

本系统首先使用了多个单一机器学习算法（SVM 、朴素贝叶斯、生成树）对样本进行训练和预测，经过不断尝试比较，最终系统采用随机森林集成学习方法（RF）对样本进行训练预测，并取得了较好的效果。

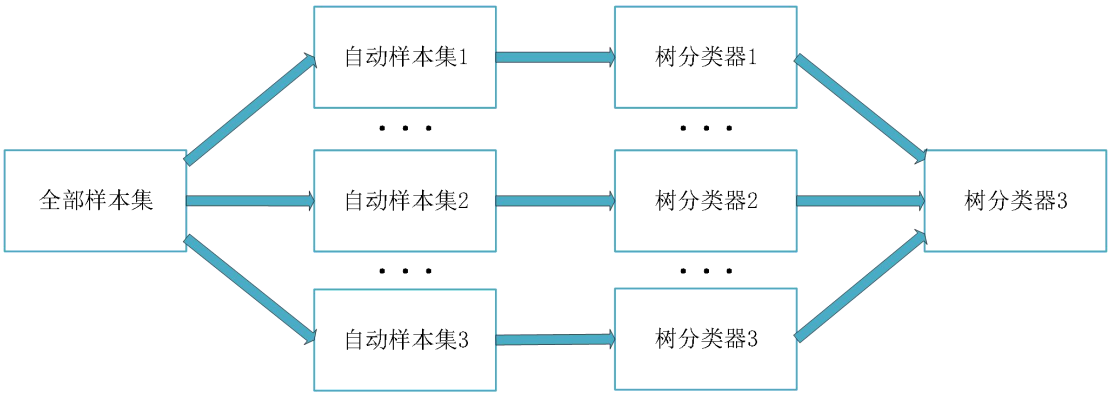

图 3.4 随机森林方法

参数分析：

A. max_features：

随机森林允许单个决策树使用特征的最大数量。 Python 为最大特征数提供了多个可选项。下面是其中的几个：

Auto/None：简单地选取所有特征，每颗树都可以利用他们。这种情况下，每颗树都没有任何的限制。

sqrt ：此选项是每颗子树可以利用总特征数的平方根个。 例如，如果变量（特征）的总数是 100，所以每颗子树只能取其中的 10 个。“log2”是另一种相似类型的选项。

0.2：此选项允许每个随机森林的子树可以利用变量（特征）数的 20％。如果想考察的特征 x％的作用， 本系统可以使用“0.X”的格式。

增加 max_features 一般能提高模型的性能，因为在每个节点上，本系统有更多的选择可以考虑， 然而，这未必完全是对的，因为它降低了单个树的多样性，而这正是随机森林独特的优点。 但是，可以肯定，你通过增加 max_features 会降低算法的速度。 因此，你需要适当的平衡和选择最佳 max_features。本系统中本系统经过多次测试选取 max_features 为 auto。

B. n_estimators：

在利用最大投票数或平均值来预测之前，你想要建立子树的数量。 较多的子树可以让模型有更好的性能，但同时让你的代码变慢。本系统选择了 n_estimators=50，达到了速度与准确率的均衡。

C. min_sample_leaf：

叶是决策树的末端节点。 较小的叶子使模型更容易捕捉训练数据中的噪声。在调优过程中，尝试多种叶子大小种类，以找到最优的参数，本系统多次尝试后，将最小叶子节点数目设置为 50。

模型保存：

在系统实现时，采用 sklearn 中的 joblib 对 RF 模型进行保存，并构建 check_WebShell 函数，对上传的 php 样本进行验证，并输出该 php 文件为 WebShell 的概率。

3.5 动态检测模块

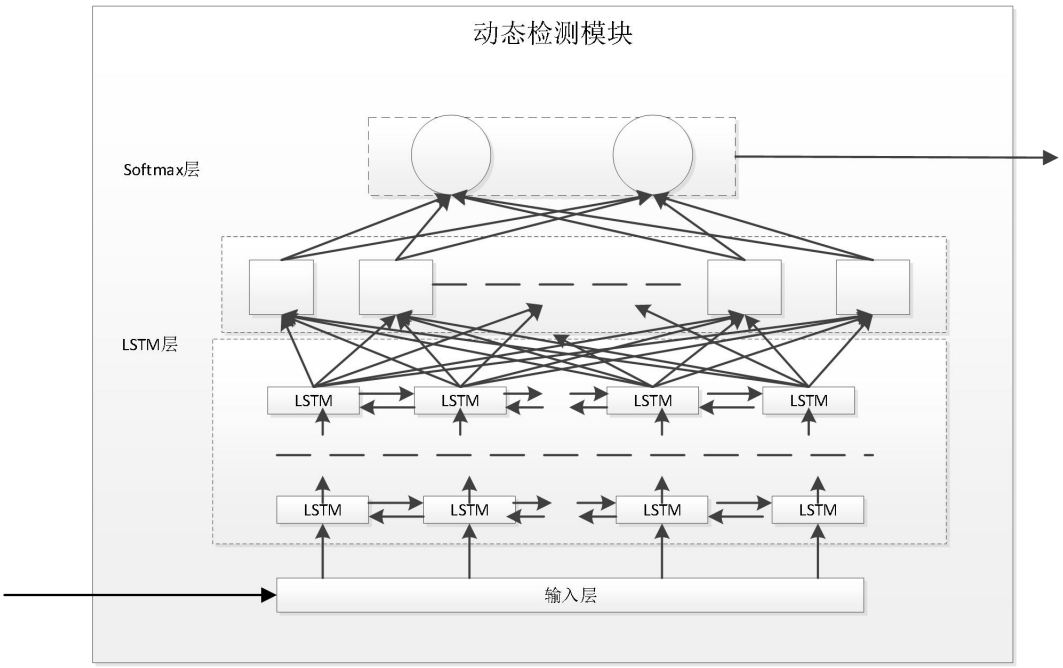

图 3.5 动态检测

模型构建：

使用 Kera 构建神经网络：

第一层：使用输入层对数据进行处理;

第二层：使用 LSTM 层进行序列收敛;

第三层：使用全连接（softmax）层输出最终结果;

构造 LSTM 网络，输入层长度为 100，输入的数据维度为词袋的词汇表的长度，输出的数据维度定义为 128，LSTM 层随机筛选 70%的数据传递给下一层；全连接层激活函数使用 softmax 函数，优化使用 adam 算法，学习速率为 0.001。

模型保存与 WebShell 检测：

使用 Keras 自带模块将 LSTM 网络保存为 H5 文件。

对于每个样本，运行命令”php_bin+" -dvld.active=1 -dvld.execute=0 "+file_path “来获取对应 php 样本的 opcode，再使用 N-gram 分词得到向量，最后将向量输入 LSTM 模型中，得到返回结果。

3.6 Web UI

Web UI 基于 Tornado 开发，为用户提供系统可视化接口，实现获取用户输入，将三个模块进行整合，引入总概率计算和自定义规则，并最终返回相应输出的功能。其中输入页面如图 3.5。

3.6 输入页面

在图 3.6 中，用户上传待检测 php 脚本，使用系统默认权重，或者根据需求自定义配置三个检测模块的权重（可选），上传自定义规则（可选）。用户上传的 php 脚本经过三个不同模块的检测，得到三组列表，每组列表中存放着单个 php 脚本为 WebShell 的概率。经过检查，系统输出文件名、对应的三个模块的概率以及总概率，如图 3.7。

OUTPUT

Weights

1st	2nd	3th
0.1	0.4	0.5

Results

#	Filename	1st	2nd	3th	Total
1	webshell.php	0.4	0.7	0.4	0.5

图 3.7 输出页面

第四章 作品测试与评价

4.1 系统开发环境

表 4.1 开发与测试环境

实验环境	配置
处理器	Intel Core i5
内存	4GB
操作系统	Ubuntu16.04LTS（64 位）
软件环境	VMWare(11),GoogleChrome(62.0.3202.94),Tornado(5.0.2),Python(2.7),Keras(2.1.5),Scikit-learn(0.19.1)

4.2 实验数据与方案

测试白样本使用主流 CMS 源代码（phpcms、wordpress、phpmyadmin、smarty），黑样本采用 github 中的开源 WebShell 项目，并使用去重工具对样本进行去重处理。去重后总计白样本数量 6051，黑样本数量 2131，进一步随机划分出测试数据（黑白样本各 100 个），并规定黑样本为正例（P），白样本为负例（N）。

4.3 测试流程

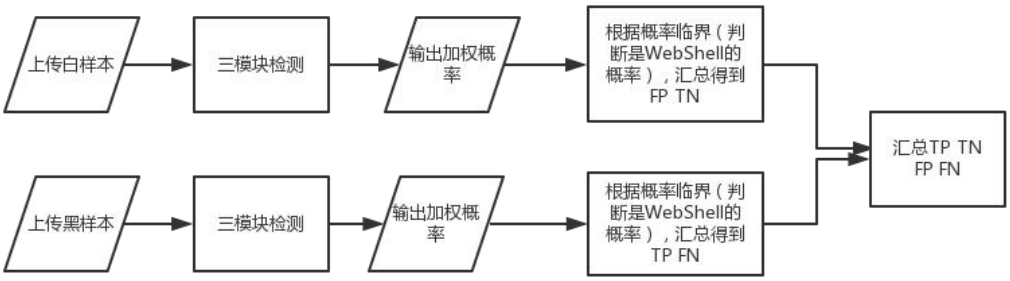

图 4.1 测试流程

测试流程如图 4.1，主要步骤如下：

1.访问测试系统，批量上传 PHP 脚本（上传白样本）；

2.自定义模块权重与规则；

3.查看测试结果，根据概率临界统计得到 FP、TN；

4.访问测试系统，批量上传 PHP 脚本（上传黑样本）；

5.自定义模块权重与规则；

6.查看测试结果，根据概率临界统计得到 TP、FN；

7.将上述结果汇总，得到 TP、TN、FP、FN。

4.4 评价指标

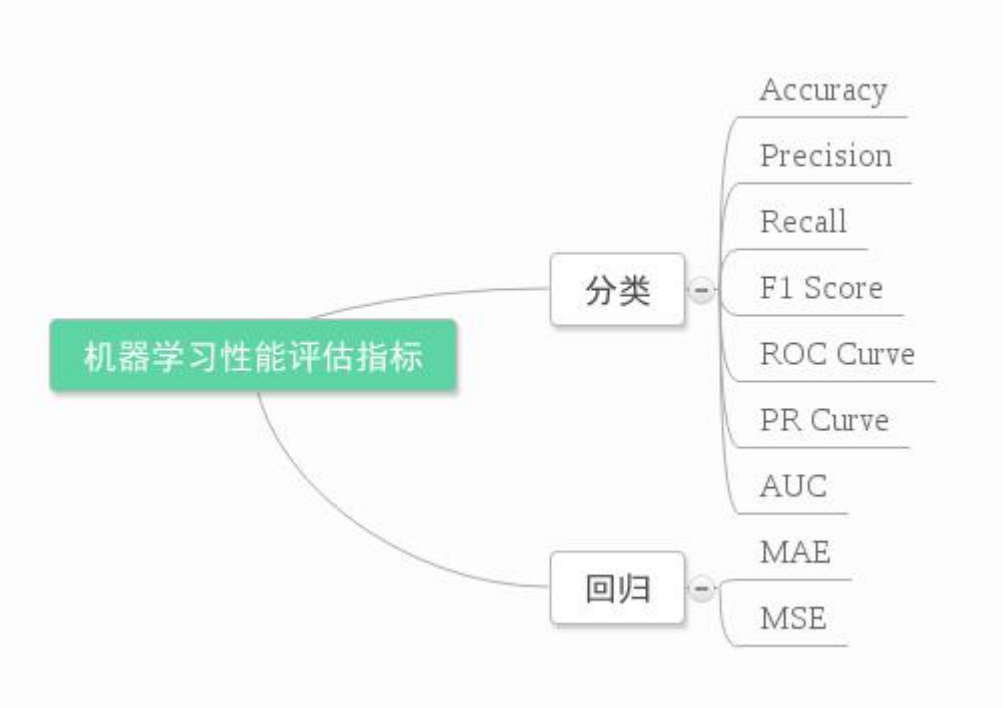

图 4.2 评估指标

1、准确率(accuracy)

准确率(accuracy): 对于给定的测试数据集，分类器正确分类的样本数与总样本数之比，也就是损失函数是 0-1 损失函数时测试数据集上的准确率。

对于二分类问题，可将样例根据其真实类别和分类器预测类别划分为：

真正例（True Positive，TP）：真实类别为正例，预测类别为正例。

假正例（False Positive，FP）：真实类别为负例，预测类别为正例。

假负例（False Negative，FN）：真实类别为正例，预测类别为负例。

真负例（True Negative，TN）：真实类别为负例，预测类别为负例。

然后可以构建混淆矩阵（Confusion Matrix）如下表所示。

表 4.2 混淆矩阵

真实类别	预测类别	
	正例	负例
正例	TP	FN
负例	FP	TN

2、精准率(precision)

精准率(precision)的公式：

$$P = \frac{TP}{TP + FP}$$

即所有”正确被检索的 item(TP)”占所有”实际被检索到的(TP+FP)”的比例。

3、召回率(recall)

召回率(recall)的公式：

$$\mathrm{R} = \frac{TP}{TP + FN}$$

即所有”正确被检索的 item(TP)”占所有”应该检索到的 item(TP+FN)”的比例。

4、F1 值

F1 值就是精确值和召回率的调和均值,也就是 F 表达式中当参数 a 为 0 时的值：

$$F = \frac{(a^2 + 1)P * R}{a^2(P + R)}$$

$$F1 = \frac{2PR}{P + R}$$

5、工作特征（ROC）

工作特征（ROC）也是分类器的评价指标。它将连续变量设定出多个不同的临界值，从而计算出一系列 TPR(真正率)和 FPR(假正率)，再以 TPR 为纵坐标、FPR 为横坐标绘制成曲线，曲线下面积越大，准确性越高。ROC 曲线下方的面积（Area Under the ROC Curve, AUC）提供了评价模型性能的方法，如果模型 TPR 在 FPR 尽量小的时候接近 100%，那么它的 AUC 也就越接近 1。

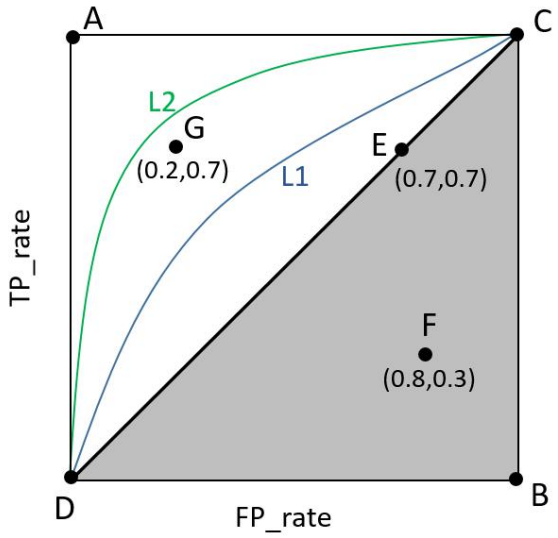

图 4.3 ROC 曲线

$$TPR = \frac{\Sigma_{TP}}{\Sigma_{TP} + \Sigma_{FN}}$$
$$FPR = \frac{\Sigma_{FP}}{\Sigma_{FP} + \Sigma_{TN}}$$

4.5 测试结果及分析

随机选取 100 个白样本与 100 个黑样本上传，进行综合检测，最终系统会给出综合评价。规定样本是 WebShell 为正例，样本是正常 php 脚本为负例。混淆矩阵如下：

表 4.3 结果混淆矩阵

	P	N
T	91	85
F	9	15

100 个白样本中判定为 WebShell 的样本数为 15 个，图 4.4 给出部分随机选择的白样本测试结果：

Results

#	Filename	1st	2nd	3th	Total
0	users.php	0.277614035088	0.243845	0.6	0.430952900277
1	upgrade-functions.php	0.723416701905	0.243837	0.6	0.698433706136
2	widgets.php	0.478213058419	0.111659	0.6	0.538093717699
3	wp-blog-header.php	0.182900091456	0.243837	0.6	0.374123739866
4	wp-activate.php	0.68	0.243838	0.6	0.672383789301
5	upload.php	0.48	0.242792	0.6	0.552279247224

图 4.4 白样本结果

100 个黑样本中认定是 WebShell 的样本数为 91 个，对于上传的黑样本测试，给出部分结果如下：

10	script.php	0.758197930857	0.243837	0.9	0.767930244351
11	themes_beatufied.php	0.34	0.243837	0.9	0.742838368499
12	ykbh.php	0.868790793517	0.243837	0.9	0.77456581611
13	mai.php	0.868790793517	0.243837	0.9	0.77456581611
14	theme_bold_footer.php	0.868790793517	0.243837	0.9	0.77456581611
15	wso-24.php	0.800511996488	0.243837	0.9	0.770469088289
16	mildnet.php	0.365733171362	0.243864	0.9	0.7443826273

图 4.5 黑样本结果

最终系统将加权总概率大于等于 0.7 时，认定该脚本为 WebShell；当加权总概率小于 0.7 时，认定该脚本为正常 php 文件。

根据前面 4.4 中的定义，准确率、精确率、召回率、F1 计算如下：

表 4.4 准确率、精确率、召回率、F1

Accuracy	Precision	Recall	F1
0.88	0.86	0.91	0.88

实验数据得到的 ROC 曲线如下：

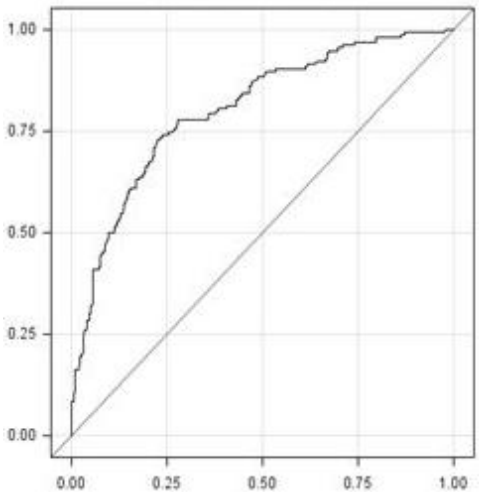

图 4.6 ROC 曲线

可见 ROC 中的 AUC 值已达到 0.8，满足现实需要。

将选取的 100 个白样本与 100 个黑样本分别置于 SVM、LSTM 模型下进行检测，并将得到的 F1 值与本系统进行比较，得到结果如下：

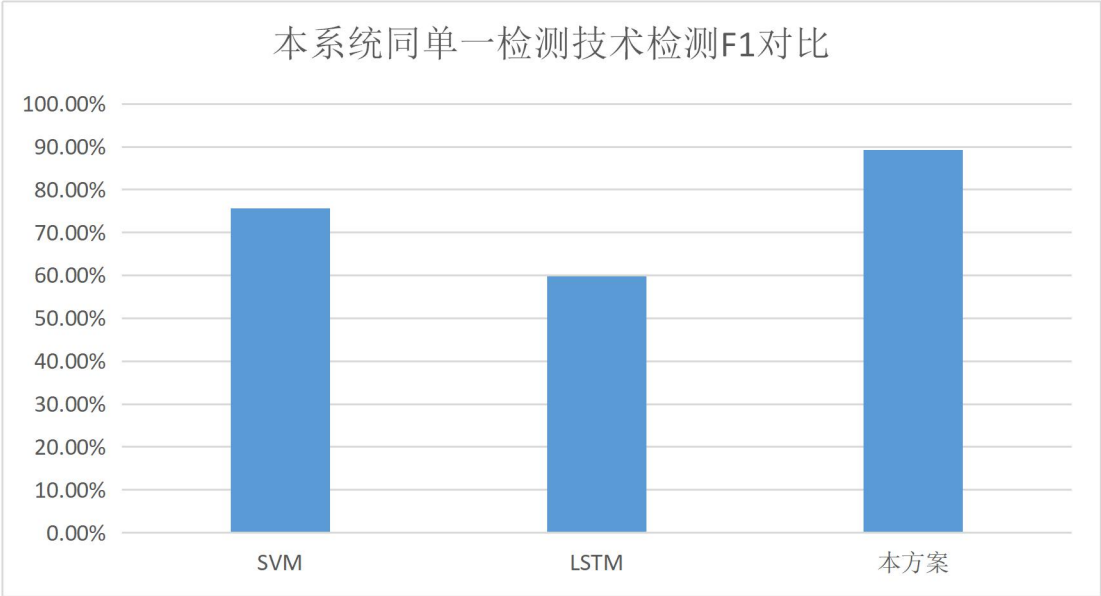

图 4.7 同单一检测技术对比

可见与单一检测策略相比，本系统的综合检测策略具有较大优势，得到了较高的 F1 值。

将选取的 100 个白样本与 100 个黑样本分别置于 360、D 盾等主流杀毒软件中进行检测，并将得到的召回率、精确率与本系统比较，得到结果如下：

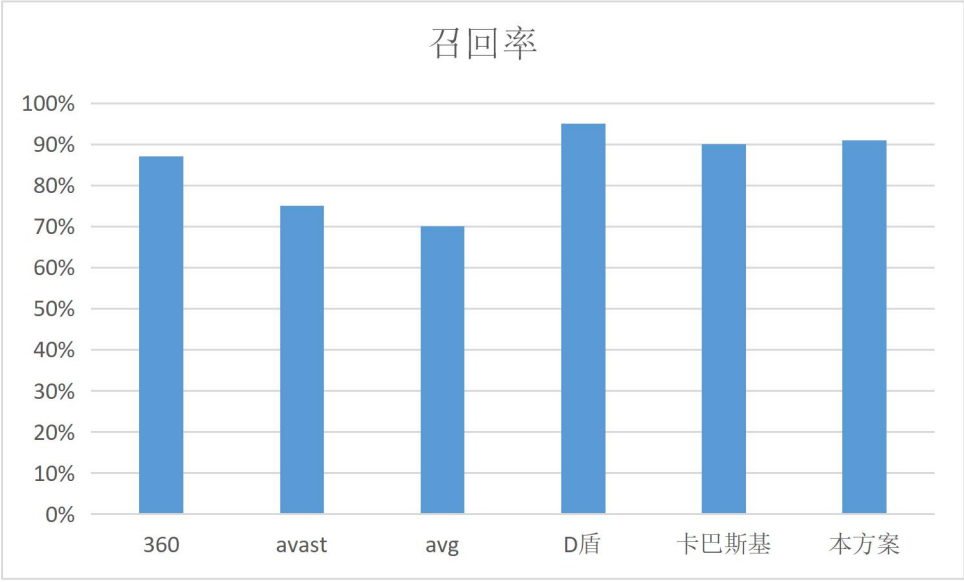

图 4.8 召回率

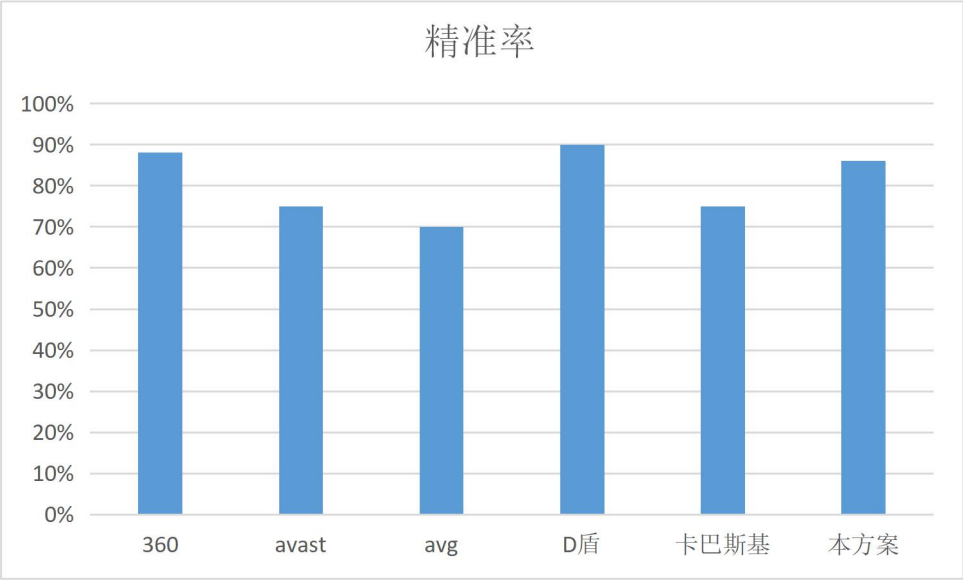

图 4.9 精准率

可见与主流杀毒软件相比，本系统召回率与精确率均达到了实际应用要求，仍有部分提升空间。

综上可知，该系统的综合策略架构较单一检测方式有明显优势，横向比较中达到了较高的召回率，但精确率方面还有一定改进空间。此外，在测试过程中上传文件和检测过程也消耗了较多时间，故检测速度方面也有很大改进空间。

第五章 创新性说明

本系统提供的 Webshell 检测功能，相较于传统检测系统，极大改善了其误报率、漏报率高以及检测效率低下的现状，拥有较高的检测效率及检测准确率，因为检测时不需要文件的上下文信息，故实现了一定的独立性，此外，还可进行用户的自定义配置，为用户的使用提供了便利性。

总体而言，本作品的创新性体现在如下几个方面：

5.1 多维综合策略检测模式

传统的检测系统往往仅采用单一的 WebShell 检测策略，而检测方法单一就不可避免的存在漏报等的问题。针对此现状，本系统基于综合检测的思想，采用了三个模块相互结合的方式来进行后门文件的检测，三个模块分别是：基于传统规则检测技术的静态检测模块、基于动态检测与深度学习的动态检测模块以及基于静态检测与集成学习的统计检测模块。

静态检测多基于静态特征的检测，仅从文件代码层面入手即可，静态检测优点对已知的 WebShell 查找准确率高，且实施方便，但其误报和漏报的缺点却也不容忽视。动态检测则利用 WebShell 执行时的动态特征，将 WebShell 上传到服务器，总得需要执行以实现功能，因此其在执行过程中就会表现出来一系列的动态特征。

多模块相互配合，综合动态检测和静态检测，可以很好的弥补因单一检测模块造成的漏报率高的不足，从而进一步提高 WebShell 的检测效率。

5.2 基于 LSTM 的高性能低误差识别

机器学习与深度学习近年来得到很大发展，其在 WebShell 检测领域中也崭露头角，本系统便应用了 LSTM 神经网络，可以很好的提高 WebShell 的检测能力。

LSTM 神经网络于 1997 年被首先提出，相较于传统神经网络，其能够记住长周期有效信息，可明确的将前面事件传递给后面的事件，从而可以让信息持续传递下去，而不需要很多的上下文语境就能分析出结果。

LSTM 可以把原本 RNN 的单元改造成一个叫做 CEC 的部件，这个部件保证了误差将以常数的形式在网络中流动 ，并在此基础上添加输入门和输出门使得模型变成非线性的，并可以调整不同时序的输出对模型后续动作的影响。

由于 LSTM 的优秀特性，应用经过训练的 LSTM 模型，对经过向量化的 PHP 代码进行分类，可以达到比传统 RNN 更加准确的结果，从而提高本系统对 WebShell 的检测效率及准确度。

5.3 基于 RF 的高精度稳定识别

在当前所有算法中，随机森林算法（RF）具有极好的准确率，在生成过程中，能够获取到内部生成误差的一种无偏估计。除此之外，相较于其它机器学习算法，其还有许多优秀的特点，它能够有效地运行在大数据集上，并且不需降维即可处理具有高维特征的输入样本，对于缺省值问题也能够获得很好的结果。

本系统应用随机森林算法，将其作为分类器的一部分，针对 php 代码而言，随机森林的诸多优点决定了其会在这种分类问题上取得很好的结果，应用了 RF 的本系统，可实现对 WebShell 的高精度稳定识别。。

5.4 在线可配置的结构化架构

本系统还可以进行用户自定义配置（如三个模块的各自的权重部分用户是可以自控的，又如静态检测模块中用户可以自己提交静态检测规则到系统），为用户的自定义使用提供了方便。

除此之外，本系统对于 WebShell 的检测只需提供待检测的 php 文件即可而无需相关上下文信息，实现了较高的独立性；此外，本系统采用 B/S 架构的形式提供服务，部署简便，并且没有严苛的硬件设备限制；最后，经过实际运行测试，发现本系统也满足运行稳定的需求。

第六章 总结

随着网络技术的迅速发展，网络安全问题也受到了越来越多的关注，何况网络安全事故频发，其中 WebShell 相关的攻击层出不穷，因此基于 Webshell 相关的安全防护显得迫在眉睫，对于 Webshell 的检测便显得尤为重要。

目前，较为短缺的网络安全人才难以应对迅速变化的网络安全形势，随着机器学习、深度学习等技术的迅速发展，其在越来越多的应用在网络安全领域，不仅是因为其可以应对安全人才短缺的现状，更是因为其在数据分析领域的出色表现。

本系统便顺应人工智能大趋势，采用了当前流行的 AI 算法（如 LSTM、RF）和，本系统的检测模块也应用了机器学习、深度学习等技术，如动态模块中的深度学习技术以及统计检测模块中的集成学习技术。除此之外，还有基于传统检测方法的静态检测模块，三个模块相互结合共同实现对后门文件的检测，同时作为一个可配置且基于多种策略的高效 WebShell 检测系统，针对已有检测系统准确率低、误报率高等常见问题做了改善，可以很好的较少误报率和漏报率，提高检测效率及准确率。

在未来的工作中，将在以下方面进行改进：

1.增加检测方式，如增加基于流量与日志的检测方式，进一步提高该系统检测的准确率。

2.不断更新并扩大训练样本集，改进学习算法，进一步提高本系统识别的精准度。

3.改进动态检测模块，如采用 server hook 检测与 opcode 检测相结合的方式，进一步发挥动态检测技术的优势。

客观来讲，由于时间与自身水平的限制，本作品的设计与测试必然存在或多或少的缺陷，在接下来的工作中，需要进行更加系统全面且深层次的设计与测试，进一步找到并修复作品的各种漏洞，不断完善系统，从而提高系统对于 WebShell 的检测能力。

参考文献

[1].国家计算机网络应急技术处理协调中心.2016 年我国互联网网络安全态势综述[R],2017

[2].360 互联网安全中心.2017 年中国网站安全形势分析报告[R],2018

[3].Helmanba.webshell-detect-methonds[R/OL]:https://blog.csdn.net/u011066706/article/details/51175971

[4].王亚丽. webshell 查杀逃逸技术研究[J]. 网络安全技术与应用, 2017

[5].龙啸,方勇,黄诚,刘亮. Webshell 研究综述. 检测与逃逸之间的博弈[J]. 网络空间安全, 2018

[6].胥小波,聂小明. 基于多层感知器神经网络的 WebShell 检测方法[J]. 通信技术, 2018

[7].潘杰. 基于机器学习的 WebShell 检测关键技术研究[D]. 中国民航大学, 2015

[8].戴桦,李景,卢新岱,孙歆. 智能检测 WebShell 的机器学习算法[N]. 网络与信息安全学报, 2017

[9].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(06)

[10].贾文超,戚兰兰,施凡,胡荣贵. 采用随机森林改进算法的 WebShell 检测方法[J]. 计算机应用研究, 2018(05)

[11].孔德广,蒋朝惠,郭春,周燕. 基于 Simhash 算法的 Webshell 检测方法[J]. 通信技术, 2018

[12].周颖,胡勇. 基于关联分析的 Webshell 检测方法研究[J]. 信息安全研究, 2018 , 4 (3) :251-255

[13].易楠,方勇,黄诚,刘亮.基于语义分析的 Webshell 检测技术研究[J].信息安全研究, 2017

[14].Truong Dinh Tu, Cheng Guang, Guo Xiaojun, Pan Wubin. Webshell Detection Techniques in Web Applications[C].2014 International Conference on

Computing,Communication and Networking Technologies (ICCCNT), 2014

[15].王文清,彭国军,陈震杭,胡岸琪. 基于组合策略的 WebShell 检测框架[J]. 计算机工程与设计, 2018

[16].Ian Goodfellow and Yoshua Bengio and AaronCourville.深度学习[M].人民邮电出版社,2017

[17].朱小虎.理解 LSTM 网络[EB/OL]:https://www.jianshu.com/p/9dc9f41f0b29

[18].周志华.机器学习[M].清华大学出版社,2016,8:171-185

[19].董师师, 黄哲学. 随机森林理论浅析[J]. 集成技术, 2013

[20]. 刘焱 . 基于机器学习的 WebShell 发现探索之旅[R/OL]:http://gitbook.cn/books/5954727545ce787060240588/index.html