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：2018.05.28

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

随着网络的高速发展，互联网新兴产业崛起迅速，其中 Web 应用系统广泛应用于社交、银行、购物和邮件等重要业务，对现实空间的影响涵盖了经济、文化、科技、服务等方方面面，在现今的网络资产中占有较高的比例。然而，由此引发的安全事件日益涌现，系统的受攻击面广以及攻击技术的多样化，导致 Web 应用系统易被攻击入侵，严重影响了国家的经济安全、社会稳定。

本系统针对已有 WebShell 检测系统准确率低、部署复杂、实时性差、不可配置的现状，采用了 WebShell 统计学检测与集成学习结合的统计检测模块、WebShell 动态检测与深度学习结合的动态检测模块和传统检测技术的静态检测模块，这三个模块相互结合并允许用户对多个模块进行自定义配置。本系统采用多模块、多维度并结合目前流行 AI 算法实现了对 WebShell 的可配置综合策略检测，达到了较高的检测效率。

本系统相比与已有 WebShell 检测系统具有准确率高，实时性强，可自定义配置，部署简单的特点。

第一章 作品概述

1.1 背景分析

1.1.1 Web 安全与 WebShell

基于网页及网页所连接的各种资源的安全性问题就构成了 Web 安全的实体。Web 安全由于其平台的特殊性，通常归纳为网站安全、数据安全以及 Web 平台上运行的应用安全三大类。

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，即一种网页版后门。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，后门文件与网站服务器 Web 目录下的正常的网页文件混在一起，然后就可以使用浏览器来访问 WebShell，得到一个命令执行环境，以控制目标网站服务器，也为后续的提权操作做准备。

通常来说，WebShell 可以简单分为三类：大马、小马和一句话木马。

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。

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

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，可单独使用也可嵌入其它正常文件。

1.1.2 Web 安全形势

随着网络安全事故频发，网络安全的威胁与挑战也越来越大，信息安全已经扩展到民生安全、社会安全、经济安全、基础设施安全、城市安全，乃至政治安全。国家对网络安全的重视程度在不断提升，大众对网络安全的关注在不断升温。

Web 安全方面，2016 年，CNCERT 监测发现约 4 万个 IP 地址对我国境内 8.2 万余个网站植入后门，网站数量较 2015 年增长 9.3%。境外有约 3.3 万个（占全部 IP 地址总数的 84.9%）IP 地址通过向网站植入后门对境内约 6.8 万个网站进行远程控制。其中来自美国的 IP 地址最多占比 14.0%，其次是来自中国香港和俄罗斯的 IP 地址，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。

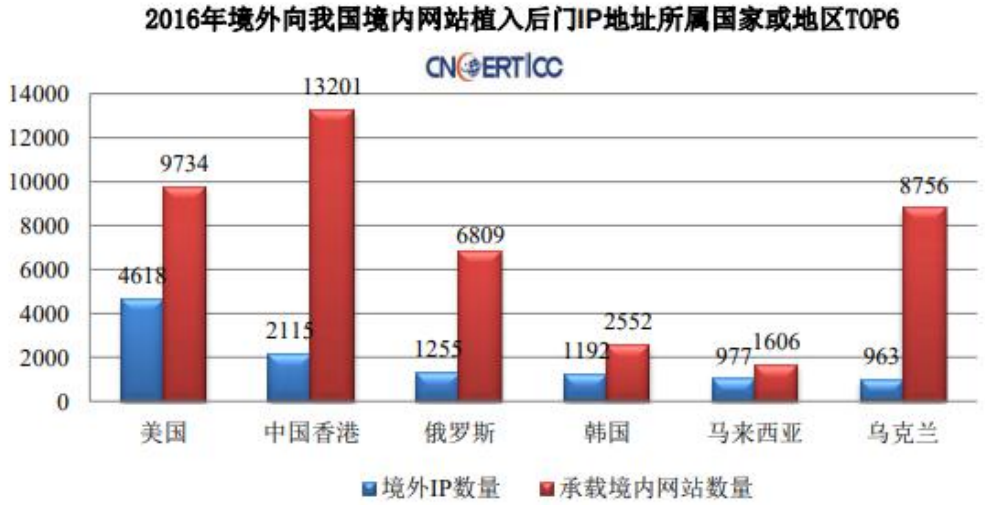


图 1.1 2016 年境外向境内植入后门国家或地区 TOP6

对于 WebShell 攻击，2017 年 1-10 月，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 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%。

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

可见 WebShell 攻击类型排第二位远超第三位，针对 WebShell 攻击的防护迫在眉睫。

对于 AI 在安全方面的应用，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，为我们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共同关注。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。

1.1.3 已有技术局限

现有技术是针对普通的脚本后门、以控制服务器为目的、通常包含较为明显的静态特征或者行为模式，不能对窃密型后门进行有效检测。

由于业务系统更新频繁，WEB 脚本文件相关的属性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，而且对系统造成的性能影响较大，甚至可能对系统稳定性造成影响，基于日志的检测方法，一方面，由于业务功能较多且复杂，部分功能可能 很少会被用到，其日志访问可能会命中某些检测规则从而造成更多的误报，另一方面，大量的日志记录处理起来会对服务器性能产生负担、而且由于日志量巨大检测 过程消耗时间长，检测速度较慢。而窃密型 WebShell 后门往往会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，通过日志分析也很难发现。

1.2 作品简介

本系统针对已有 WebShell 检测系统准确率低、部署复杂、实时性差、不可配置的现状，采用了 WebShell 统计学检测与集成学习结合的统计检测模块、WebShell 动态检测与深度学习结合的动态检测模块和传统检测技术的静态检测模块，这三个模块相互结合并允许用户对多个模块进行自定义配置，多模块、多维度并结合目前流行的 AI 算法实现了对 WebShell 的可配置全方位检测，达到了较高的检测效率。

1.3 作品特色

1.准确率高

本系统采用多技术、多维度的检测方法，并且运用深度学习、集成学习与之

结合对 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、深度学习和传统模型对 WebShell 进行检测测试，检测时间因文件数目与文件大小、主机配置而异，测试发现对于一般 PC、10M 文件系统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，具备较高的独立性。

4.可配置性

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

5.部署简单

本系统部署简便，无特殊的硬件环境限制。采用 B/S 架构提供服务，方便易用。

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，由此导致了企业频繁遭受 Web 攻击（WebShell 攻击），因此企业公司更应加强此类防护，防止被入侵造成重大经济损失。

2.网络安全公司产品

将本作品集成于网络安全软件中能有效提升网络安全软件对 WebShell 的防护，同时由于本产品采用深度学习、集成学习技术，在用户使用过程中也提供了大量样本，从而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 攻击导致的机密文件泄露给党和国家造成重大损失，党政机关应该重点加强此类防护保证自身信息系统安全。

第二章 研究现状与相关技术

2.1 研究现状

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，笼统来讲，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）。

当前，随着机器学习、深度学习等技术的发展，其也更多的应用到 WebShell 检测中：

机器学习方面，如可以通过一种智能检测 WebShell 的机器学习算法，对已知存在 WebShell 和不存在 WebShell 的页面进行特征学习，完成对未知页面的预测。又如基于支持向量机（Support vector machine-SVM）分类算法的检测，通过分析 WebShell 的 HTML 页面，利用 SVM 可以进行黑盒检测，是一种有监督的机器学习系统，对网页的 HTML 页面进行学习，可以在未知源代码的情况下对 WebShell 进行检测。

深度学习方面，如基于多层感知器（Multi-Layer Perceptron，MLP）神经网络的 WebShell 检测方法，其通过多层神经网络训练得到检测模型。

其它检测技术，像根据 WebShell 代码复用思想构建的基于 Simhash 算法的检测技术，其通过构建指纹库，可以做到实时监测和报警。除此之外，还可将各种策略组合在一起构建出检测框架。

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，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，日后 WebShell 将会更加复杂，其逃逸技术越复杂，传递的数据流就越复杂，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；除此之外，机器学习与深度学习必将在更多的应用到检测领域；同时，在被动检测的基础上，利用蜜罐技术、社会工程学、指纹跟踪等技术来实现主动的 WebShell 溯源。

2.2 基本原理

2.2.1 静态检测

所谓基于静态特征的检测，即不需要执行此文件，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，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，将文件内容与之匹配，便可实现检测 WebShell 的目的。

静态特征检测是最基础的检测技术，同时也是最常见的一种，高级一点，便会涉及语法语义分析。优点是快速方便，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。

静态检测只能查找已知的 WebShell，会有误报和漏报的问题。

误报：特征库中的某些敏感函数，程序的正常功能也会用到。如比如 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。

2.2.2 动态检测

WebShell 上传到服务器，总得需要执行以实现功能，其在执行过程中表现出来的特征，便是动态特征。

Opcode 是计算机指令中的一部分，PHP 作为一门动态脚本语言，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，接着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。运行 php 脚本，分析 opocode 可以认为是动态检测方法

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。因为是黑名单思想，此种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。进一步，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。

2.2.3 其他检测技术

笼统来讲，WebShell 检测思路即两种：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，从文件代码入手即是静态检测。WebShell 运行后，观察 WebShell 的字节码和通信即动态检测。

除了动静两大检测思路，从其它角度讲，还会有三种检测模式：基于流量模式、基于 agent 模式和基于日志分析模式。基于流量动态检测，基于 agent 即静态检测，使用 WebShell 一般不会在系统日志中留下记录，但是会在网站的 Web 日志中留下 WebShell 页面的访问数据和数据提交记录。日志分析检测技术通过大量的日志文件建立请求模型从而检测出异常文件，故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。

2.3 相关技术

2.3.1 随机森林

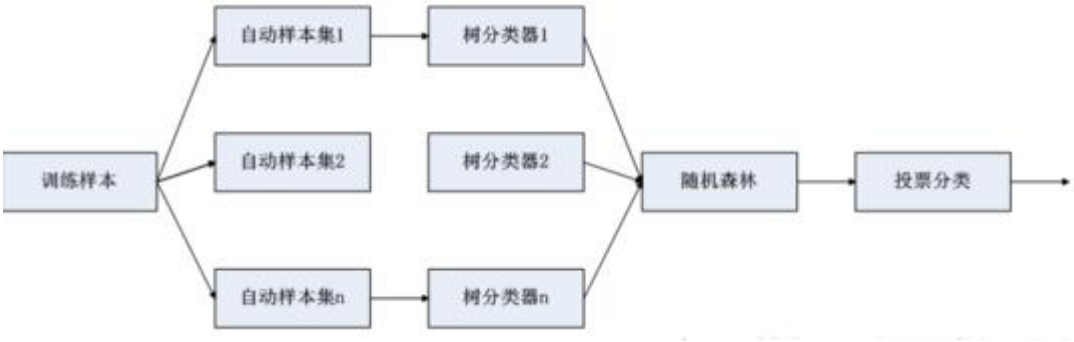


图 2.1 随机森林

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。

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。

2.3.2 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）的相互联接构成，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，我们假设所有的输入（包括输出）之间是相互独立的。对于很多任务来说，这是一个非常糟糕的假设。如果你想预测一个序列中的下一

个词，你最好能知道哪些词在它前面。RNN 之所以循环的，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。换一种方式思考，可以认为 RNN 记忆了到当前为止已经计算过的信息。

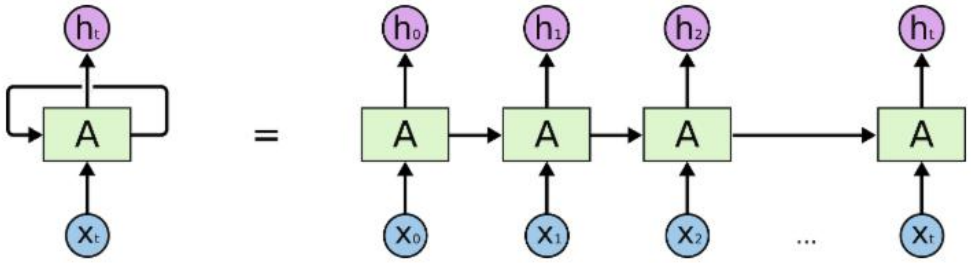


图 2.2 RNN

2.3.3 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其优势在于会持续思考，能够记住长周期有用信息，可以明确的将前面事件传递给后面的事件得以让信息持续传递下去，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，RNN 本质上是与序列和列表相关的。他们是对于这类数据的最自然的神经网络架构）。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，这个重复的模块只有一个非常简单的结构，例如一个 tanh 层。

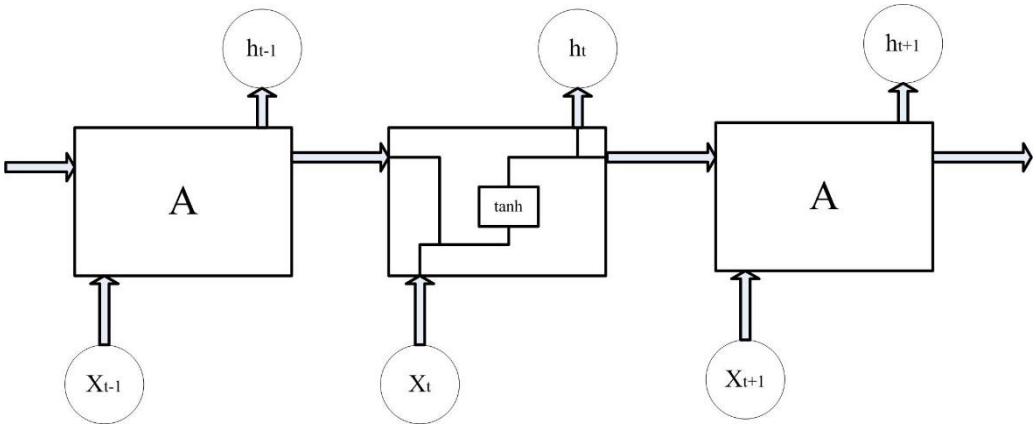


图 2.3 标准 RNN 中的重复模块包含单一的层

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，以一种非常特殊的方式进行交互。

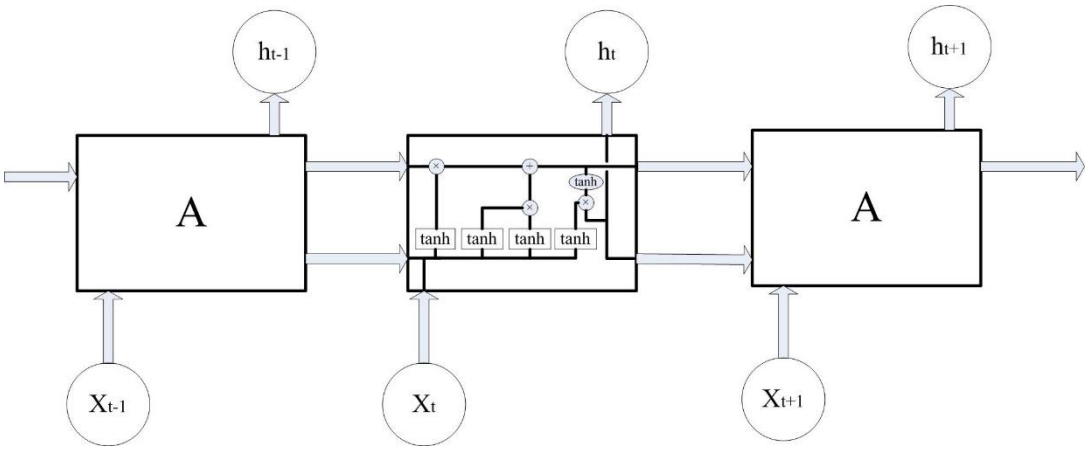


图 2.4 LSTM 中的重复模块

四个交互的层其中最关键的就是细胞状态，即模型最上方的水平线：

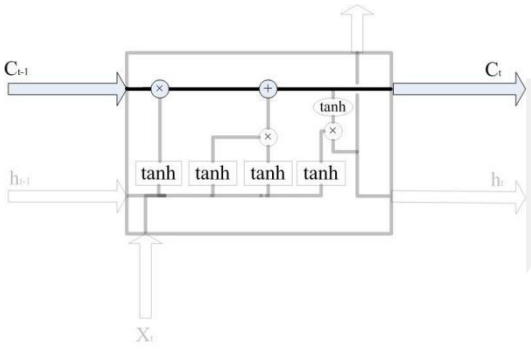


图 2.5 细胞状态（黑线标记）

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。其分别是忘记门层（确定丢弃信息）：

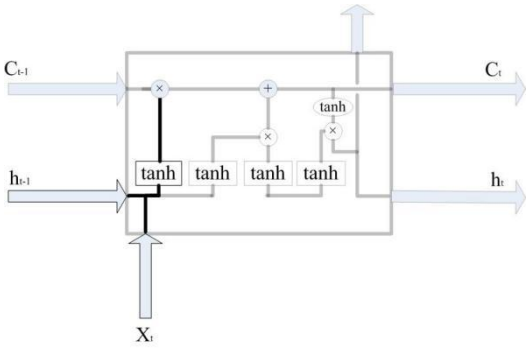


图 2.6 忘记门层（黑线标记）

输入门层（确定更新信息）：

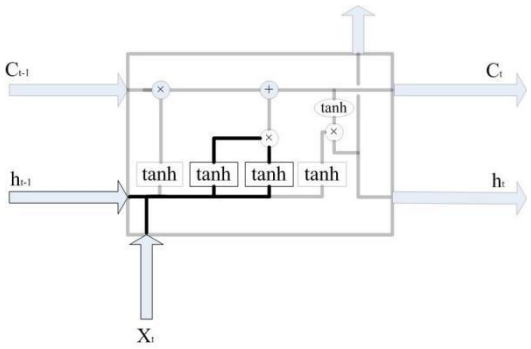


图 2.7 输入门层（黑线标记）

更新门层（更新细胞状态）：

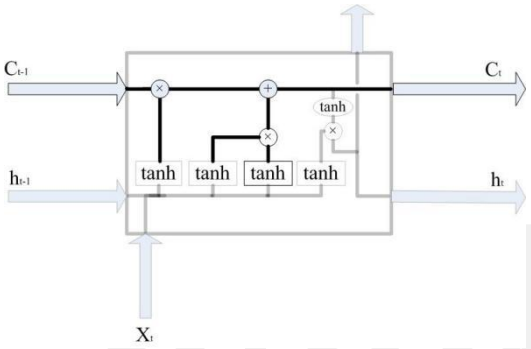


图 2.8 更新门层（黑线标记）

本系统由 128 个单元构成：

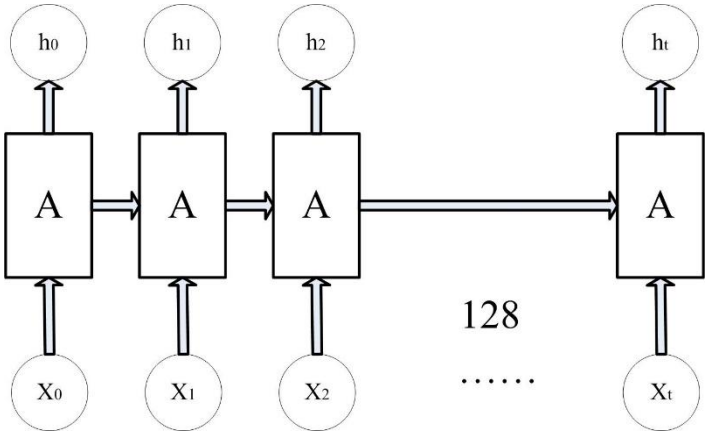


图 2.9 输出结构

通过输入 128 个特征值产生 128 个输出。

2.3.4 集成学习

集成学习通过将多个学习器进行结合，常可获得比单一学习器显著优越的泛化性能，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”，即学习器间具有差异。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。

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。

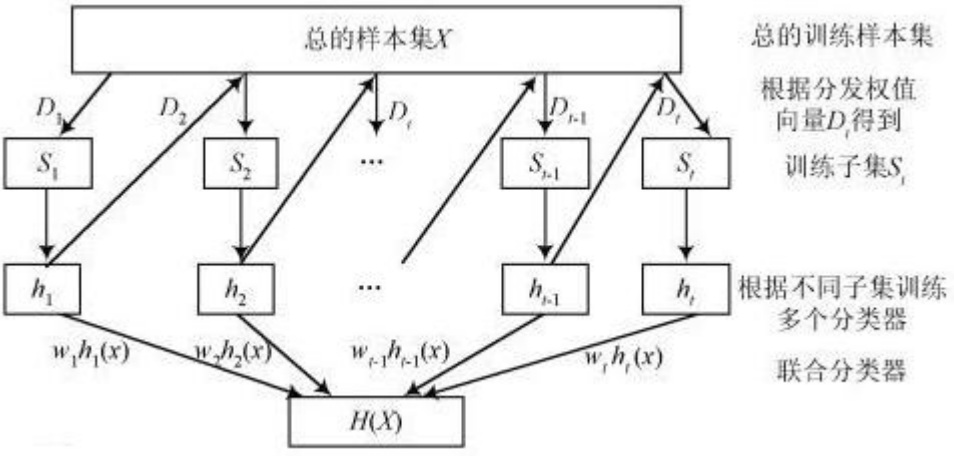


图 2.10 boosting

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：

$$H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$$

$$\ell_{\exp}(H \mid \mathcal{D}) = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}}[e^{-f(\boldsymbol{x})H(\boldsymbol{x})}] \ .$$

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。

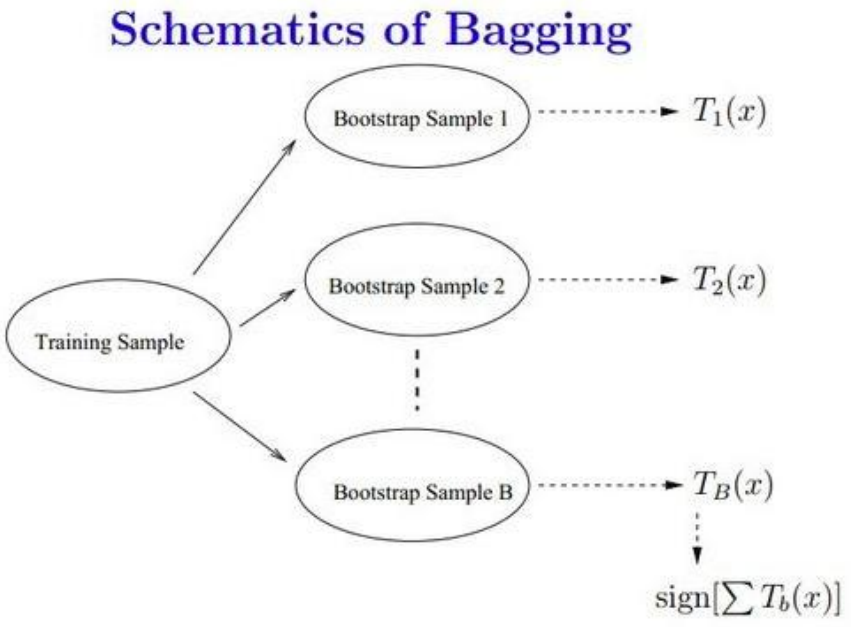


图 2.11 bagging

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，我们可以采样出 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。

第三章 作品设计与实现

3.1 系统方案

本系统采用了 WebShell 统计学检测与集成学习结合的统计检测模块、WebShell 动态检测与深度学习结合的动态检测模块和传统静态检测的静态检测模块，这三个模块相互结合并允许用户对多个模块进行自定义配置，多模块、综合策略并结合目前流行的 AI 算法实现了对 WebShell 的可配置综合策略检测。

模块图如下：

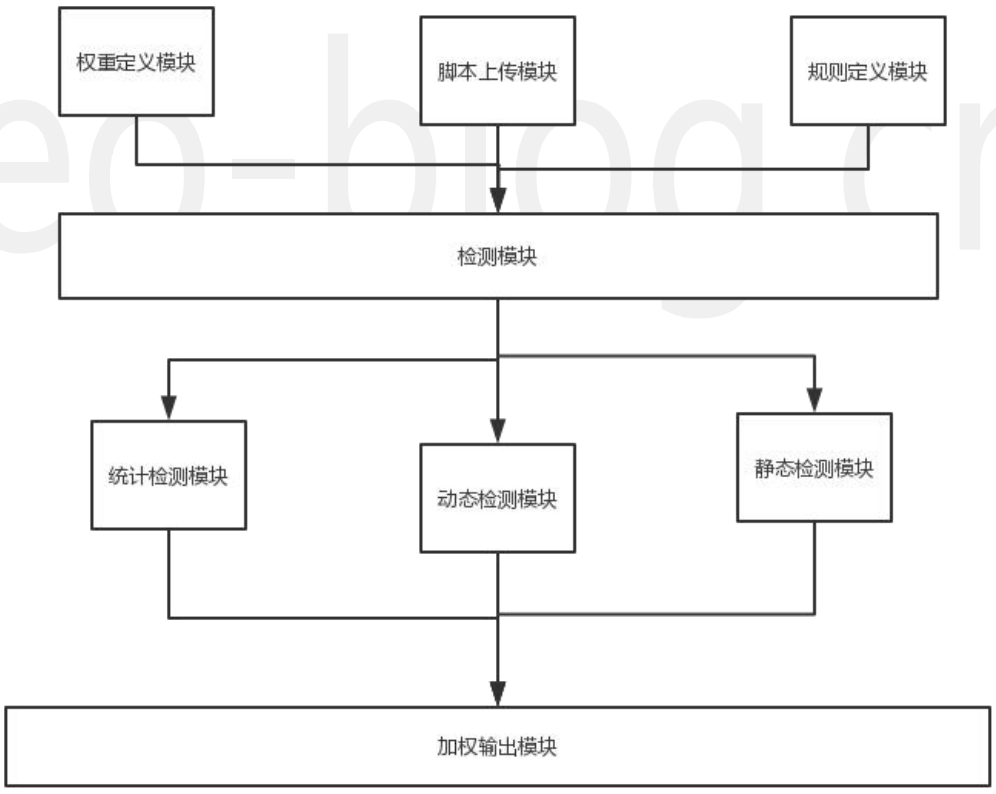


图 3.1 模块图

1. 用户上传待检测 php 脚本，配置好三个检测模块各自权重（可选），上传自定义规则（可选）。

2. 系统将用户上传的 php 脚本经过三个不同模块（静态检测模块可扩展）的检

测得到三组列表，每组列表中存放着单个上传的 php 脚本为 WebShell 的概率。

3. 如果用户自定义了三个模块权重则按照用户自定义权重计算总概率否则使用默认权重计算总概率。

4. 输出文件名，对应的三个模块的概率和总概率。

程序流程图如下：

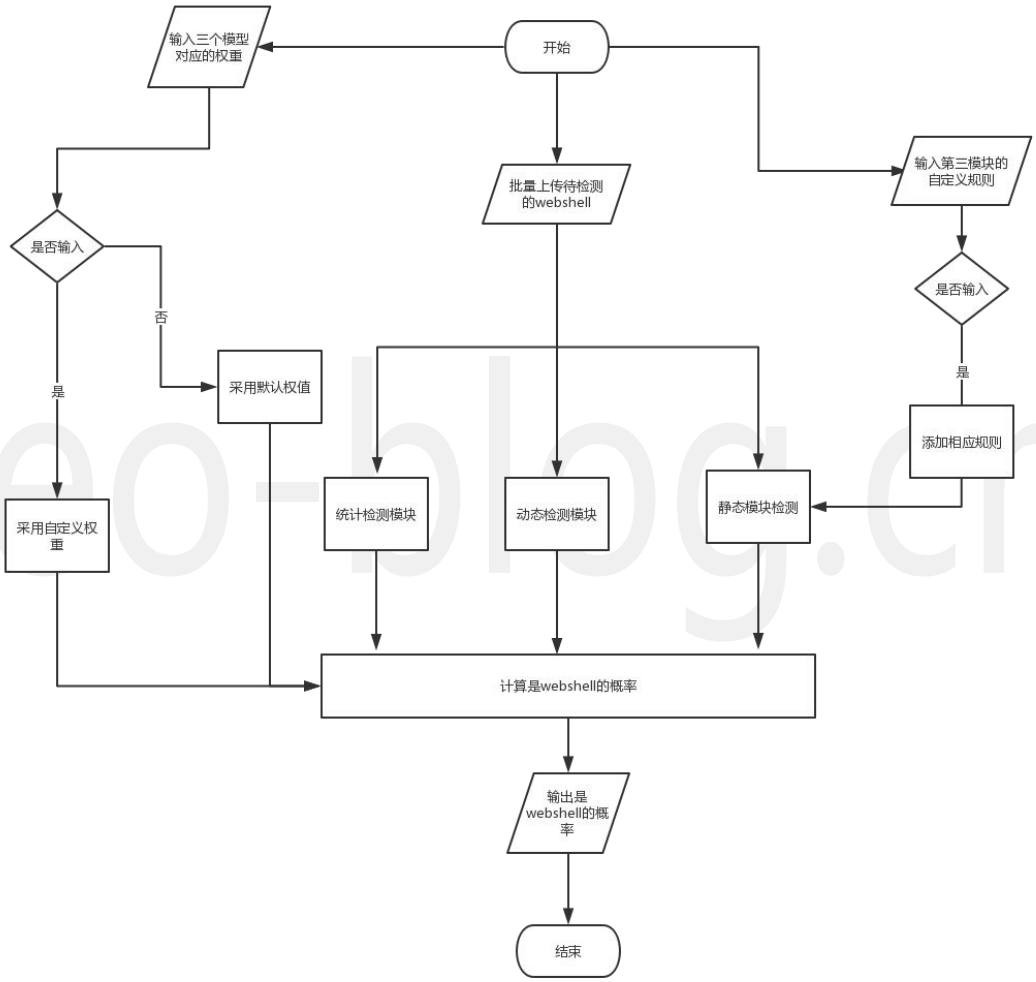


图 3.2 流程图

3.2 系统实现

3.2.1 预处理

样本：

本系统白样本使用主流开源 CMS 源代码（phpcms、wordpress、phpmyadmin、smarty），黑样本采用 github 中的开源 WebShell 项目，使用去重工具对样本进行去重。

去重后总计白样本数量 6051　黑样本数量 2131。

统计检测模块数据处理：

去除样本中的注释和 html 代码块，进行编码处理，再基于词袋 N-gram 和 TF-IDF 进行特征提取得到特征向量。

动态检测模块数据处理：

对于每个样本 运行

php_bin+" -dvld.active=1 -dvld.execute=0 "+file_path

获取对应 php 样本的 opcode 再采用 N-gram 分词得到特征向量

3.2.2 统计检测模块

模型构建：

本系统一开始使用了单一的机器学习算法（SVM 朴素贝叶斯 生成树）对样本进行训练和预测并没有得到很好的效果。后采用集成学习算法和深度学习算法，深度学习算法中（LSTM）对长序列并没有收到较好的收敛效果，集成学习中采用（RF）对样本进行训练预测调优中起到了较好的效果。

模型保存与 WebShell 验证：

使用 sklearn 中的 joblib 对 RF 模型进行了保存，构建了 check_WebShell 函数对上传的 php 样本采用上述数据处理方式使用保存后的模型进行验证，输出该 php 文件为 WebShell 的概率。

```
def check_webshell(clf,dir):
    name_list = []
    files_list = []
    g = os.walk(dir)
    for path, d, filelist in g:
        for filename in filelist:
            if filename.endswith('.php'):
                fulepath = os.path.join(path, filename)
                name_list.append(fulepath)
                t = load_file(fulepath)
                files_list.append(t)
    x=files_list
    CV = CountVectorizer(ngram_range=(2, 4), decode_error="ignore",max_features=
                                        token_pattern = r'\b\w+\b',min_df=1, max_
    x=CV.fit_transform(x).toarray()
    transformer = TfidfTransformer(smooth_idf=False)
    x_tfidf = transformer.fit_transform(x)#tf-idf  the regularilizm of a word to
    x = x_tfidf.toarray()
    y_pred = clf.predict_proba(x)
    all_php=len(x)
    for i in range(len(y_pred)):
        if y_pred[i][1] > 0 :
            print "%s proba is %s" % (name_list[i],y_pred[i])
```

图 3.4 主要代码

3.2.3 动态检测模块

模型构建：

使用 Kera 构建神经网络：

第一层使用嵌入层对数据维度进行处理

第二层使用 LSTM 层进行序列收敛

第三层使用全连接层

模型保存：

```
from keras.models import load_model

model.save('my_model.h5')  # creates a HDF5 file 'my_model.h5'
```

图 3.5 模型保存

使用 Keras 自带模块对 LSTM 网络进行保存。

WebShell 检测：

对于每个样本 运行 php_bin+" -dvld.active=1 -dvld.execute=0 "+file_path

获取对应 php 样本的 opcode 再采用 N-gram 分词得到向量

将向量输入 LSTM 模型中返回结果。

3.2.4 静态检测模块

本模块基于静态特征，python 语言编写，实现了对 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，作为一个整体模块导入至主文件，可对特定一目录进行扫描检测，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。

程序的功能简介就是如此，下面介绍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。

举例说明，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，初步通过字符串查找，在文件内容中搜寻是否含有 preg_replace 函数，再进一步通过正则，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：

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：

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：

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。

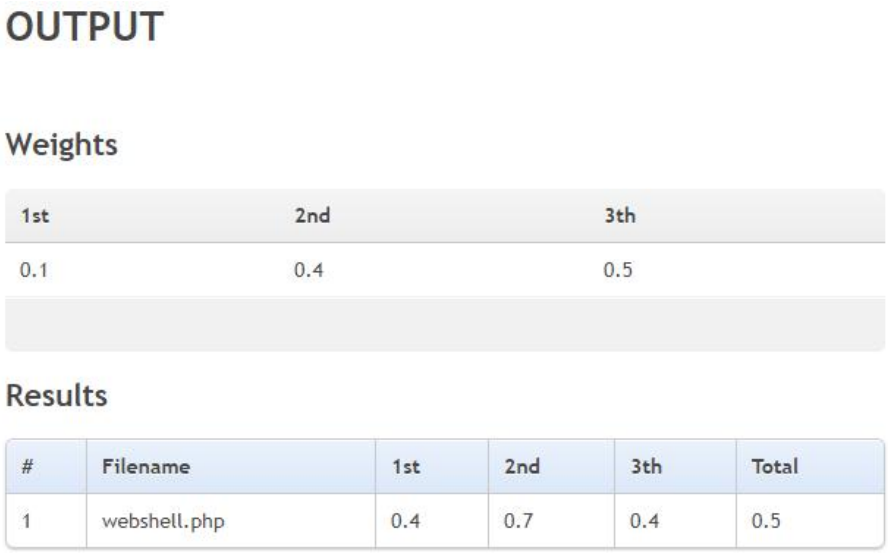
3.2.5 Web UI

Web UI 使用 Tornado (Python Web Server)实现 B/S 架构，实现客户端与服务端、前端与后端交互。

通过 python 获取输入运用了 Web 的批量文件上传功能，将三个模块整合，并引入总概率计算和自定义规则功能，返回相应输出。

图 3.6 引导页

图 3.7 输入页面



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 3.8 输出页面

使用样例如下：

Input：

批量文件上传（1.php 2.php 3.php）
输入统计检测模块权重 0.4
输入动态检测模块权重 0.2
输入静态检测模块权重 0.4
添加自定义规则（eval 字样检测）

表 3.1 INPUT

Output：

模块	统计检测模块	动态检测模块	静态检测模块
权重	0.4	0.2	0.4

文件名	统计检测模块判断样本是 WebShell 的概率	动态检测模块判断样本是 WebShell 的概率	静态检测模块判断样本是 WebShell 的概率	加权总概率
1.php	0.1	0.1	0.1	0.1
2.php	0.4	0.8	0.7	0.7
3.php	0.9	0.9	0.8	0.9

表 3.2 OUTPUT

3.3 开发环境

操作系统：Ubuntu16.04LTS（64 位）

统计检测模块采用了 sklearn 构建 RF 模型

动态检测模块采用了 Keras 框架构建 LSTM 神经网络

静态检测模块使用 python3 构建静态匹配

Web UI 使用 Tornado (Python Web Server)实现 B/S 架构 实现服务端与客户端、前端与后端交互

第四章 作品测试与评价

4.1 系统测试

4.1.1 测试环境

主机：Lenovo g50-70　CPU：inter core i5　　Memory：8G

操作系统：Windows10 专业版 1709

浏览器 Google Chrome 版本 ：62.0.3202.94（正式版本） （64 位）

4.1.2 测试方案

1.测试样例选择

白样本在 github 中选择一开源项目

https://github.com/wdnb/msg php 留言板

黑样本在 github 中选择

https://github.com/tanjiti/WebShellSample/tree/master/PHP

2. 访问测试系统 批量上传 PHP 脚本（首次上传白样本）

3. 自定义模块权重与规则

4. 查看测试结果

5.访问测试系统 批量上传 PHP 脚本（上传黑样本）

6.自定义模块权重与规则

7.查看测试结果

4.1.3 测试流程

1.访问测试系统 批量上传 PHP 脚本（首次上传白样本）

自定义模块权重与规则

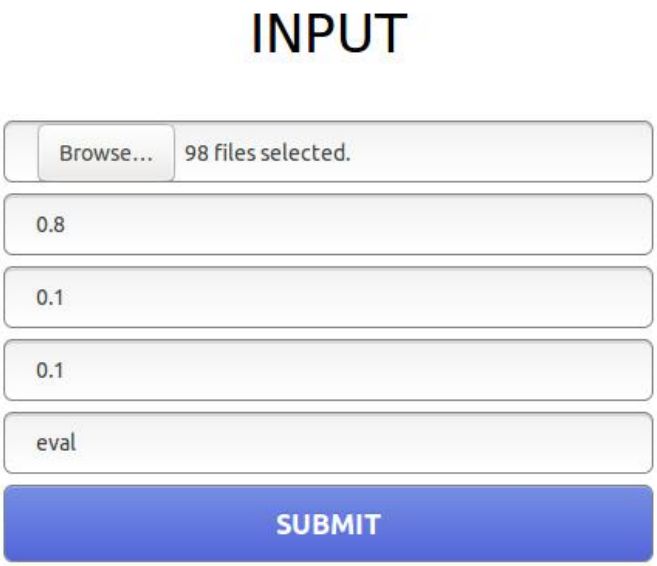


图 4.1 白样本测试

查看测试结果

2.访问测试系统 批量上传 PHP 脚本（上传黑样本）

自定义模块权重与规则

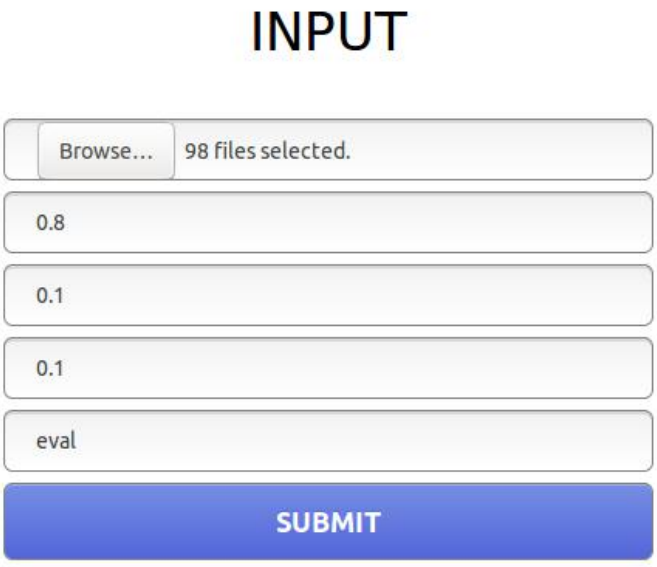


图 4.2 黑样本测试

查看测试结果。

4.1.4 测试结果

取系统判断 WebShell 概率大于等于 0.7 为 WebShell 临界。

对于上传的白样本测试结果如下：

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
6	wp-trackback.php	0.218134791802	0.243837	0.6	0.395264560074
7	upgrade.php	0.568739241105	0.243837	0.6	0.605627229656
8	vars.php	0.734008107171	0.239203	0.6	0.704325152984
9	version.php	0.868790793517	0.243837	0.6	0.785658161103
10	user.php	0.44	0.243486	0.6	0.528348555505

图 4.3 白样本结果

98 个白样本中判断是 WebShell 的总概率>0.7 的样本数为 4 个，准确率为(98-4)/98=0.9591

对于上传的黑样本测试结果如下：

5	wp-gallery.php	0.3	0.243837	0.9	0.740438368499
6	systemsinf.php	0.61177420809	0.243837	0.9	0.759144821283
7	feeds.php	0.765671479868	0.243837	0.2	0.208378657291
8	priv8.php	0.42	0.224753	0.9	0.747447529775
9	response41.php	0.838603573489	0.243837	0.2	0.212754582909
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 4.4 黑样本结果

98 个黑样本中判断是 WebShell 的总概率>0.7 的样本数为 85 个，准确率为 85/98=0.8673

4.1.5 结果分析

由上面的测试结果可知，对于白样本的检测该系统达到了较好的效果，对于黑样本的检测，还有较大上升改进空间。另外，测试过程中上传多个文件和检测过程消耗了较多时间，对于提高检测速度也有一定改进空间。

4.2 系统评价

4.2.1 准确率(accuracy)

准确率(accuracy),其定义是：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 4.1 混淆矩阵

真实类别	预测类别	
	正例	负例
正例	TP	FN
负例	FP	TN

4.2.2 精确率(precision)

精确率(precision)的公式是

P=TPTP+FPP=TPTP+FP

,它计算的是所有”正确被检索的 item(TP)”占所有”实际被检索到的(TP+FP)”的比例.

4.2.3 召回率(recall)

召回率(recall)的公式是

R=TPTP+FNR=TPTP+FN

,它计算的是所有”正确被检索的 item(TP)”占所有”应该检索到的 item(TP+FN)”的比例。

4.2.4 F1 值

F1 值就是精确值和召回率的调和均值,也就是

2F1=1P+1R2F1=1P+1R

调整下也就是

F1=2PRP+R=2TP2TP+FP+FN

在对样本 2-gram 分词和 tf-idf 特征提取 去除代码注释后

随机 2-8 划分测试、训练集

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

使用 rf 训练后的预测情况（准确率(Accuracy),精确率(Precision),召回率(Recall)和 F1-Measure）如下

第一次训练：

metrics.accuracy_score:

0.979591836735

metrics.precision_score:

0.968503937008

metrics.recall_score:

0.964705882353

metrics.f1_score:

0.966601178782

第二次训练：

metrics.accuracy_score:

0.979591836735

metrics.precision_score:

0.968503937008

metrics.recall_score:

0.964705882353

metrics.f1_score:

0.966601178782

表 4.2 四个指标

次数	Accuracy	Precision	Recall	F1
第一次训练	0.98	0.97	0.96	0.97
第二次训练	0.98	0.97	0.96	0.97

可见，四个指标均达到了实际应用效率。

4.2.5 工作特征（ROC）

工作特征（ROC）是 LSTM 分类器的评价指标。它通过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。TPR 在 FPR 尽量小的时候接近 100%，模型的性能也就越高。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。

$$\mathrm{TPR} = \frac{\sum True\ \ Positive}{\sum True\ \ Positive + \sum False\ \ Negative}$$

$$\mathrm{FPR} = \frac{\sum False\ \ Positive}{\sum False\ \ Positive + \sum True\ \ Negative}$$

本系统在动态检测模块模型生成时，利用测试样本（总样本的 20%）生成的 ROC 其 AUC 值已达到 0.8，满足现实需要。

4.2.6 综合横向比较

选取 100 个白样本与 100 个黑样本分别在 SVM、LSTM 与本系统进行检测，得到准确率如下：

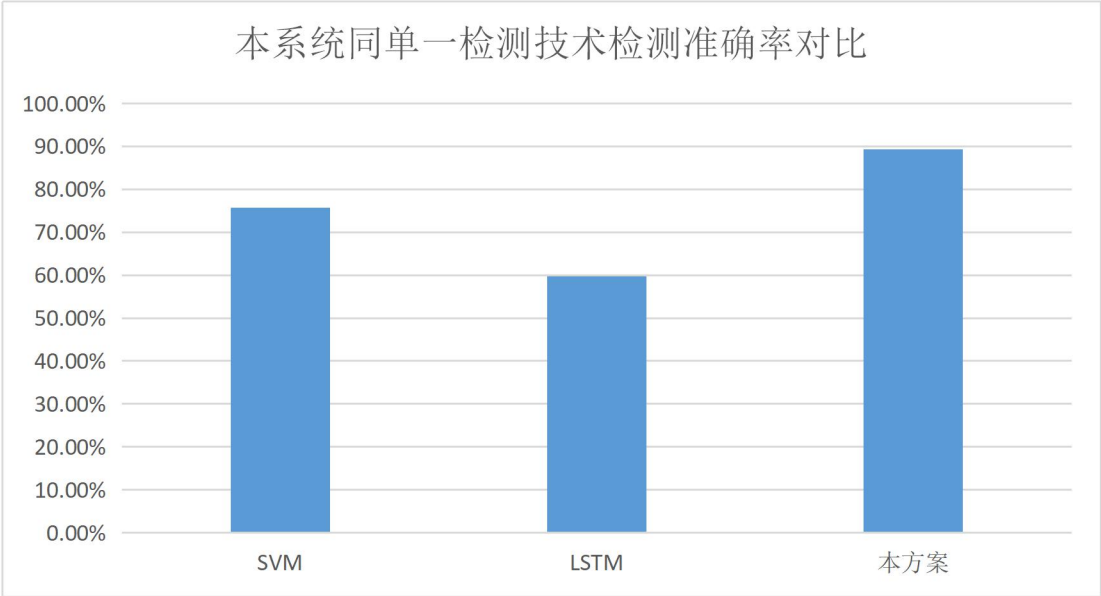


图 4.5 同单一检测技术对比

可见本系统的综合检测策略较单一检测方式具有较大优势，得到了较高的检测率。

选取 100 个白样本与 100 个黑样本分别在 360、D 盾等主流杀软与本系统进行检测，得到查杀率、误杀率分别如下：

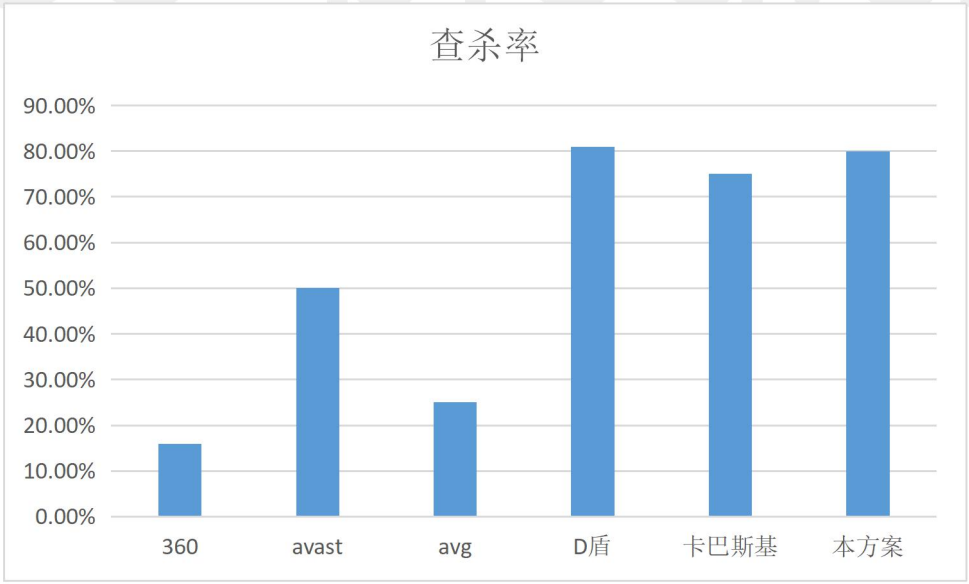


图 4.6 查杀率

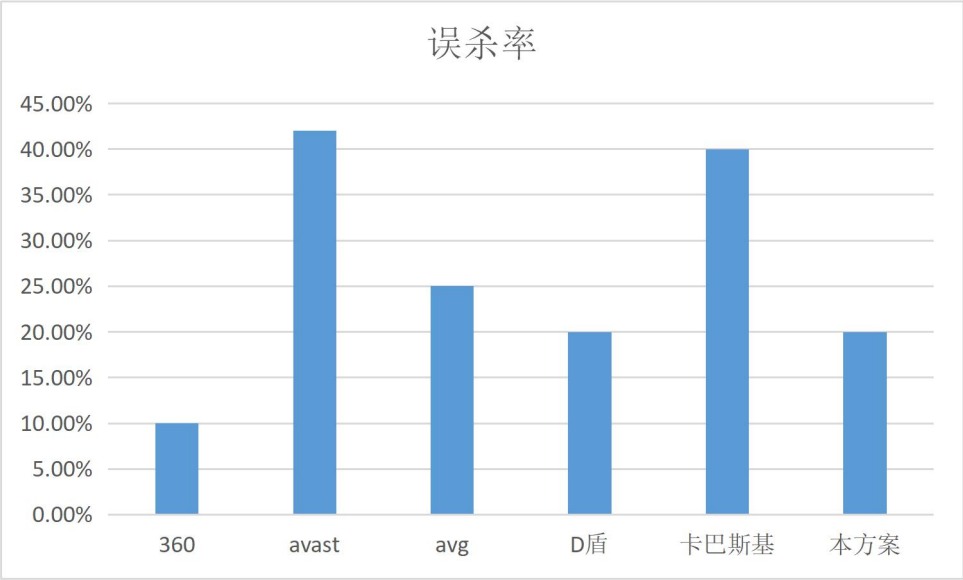


图 4.7 误杀率

可见本系统查杀率与误杀率均达到了实际应用要求，但仍有较大提升空间。

第五章 创新性说明

本作品提供的 WebShell 检测功能，克服了传统技术的繁杂性和滞后性，为用户提供了实时的检测，大大提高了检测效率，同时还具有一定的创新性，主要体现在如下几个方面：

1.采用多种检测技术：本系统采用多模块、多维度检测技术对 WebShell 进行检测，弥补了当前 WebShell 检测系统单一的检测方式带来的准确率不高的问题。

2.将深度学习、集成学习与 WebShell 检测相结合：深度学习、集成学习技术是目前很火的一门技术，广泛应用于人工智能、计算机视觉、物体识别和检测、场景分类、根据图像生成文字描述等领域，本系统运用了深度学习、集成学习技术对 WebShell 的检测时间、准确率有较明显提高。

3.可自定义配置规则：本系统不同已有 WebShell 检测系统可以对三个检测模块进行自定义配置，引入自己定制的规则，具备一定的可配置性。

第六章 总结

随着网络技术的飞速发展，智能化程度的不断提高，人们的生活也越来越便捷，与此同时也带来了很多难以预料的威胁。网络安全问题也成为了当今人们越来越重视的话题。

本系统针对已有 WebShell 检测系统准确率低、部署复杂、实时性差、不可配置的现状，采用了 WebShell 统计学检测与集成学习结合的统计检测模块、WebShell 动态检测与深度学习结合的动态检测模块和传统静态检测的静态检测模块，这三个模块相互结合并允许用户对多个模块进行自定义配置，多模块、多维度结合目前流行 AI 算法实现了对 WebShell 的可配置全方位检测，达到了较高的检测效率。

在未来的工作中，我们认为本作品还可以从以下几个方面进行改进：

1.增加检测方式，增加基于流量与日志的检测方式进一步提升该系统的检测准确率。

2.不断更新扩大训练样本集，改进学习算法，进一步提高本系统识别的精准度。

3.加大对于动态检测模块的改进，采用server hook检测与opcode检测结合，进一步发挥动态检测技术的优势。

由于时间与水平有限，本作品设计与测试或多或少存在缺陷，还远远不够完善，在未来的工作中需要进行更加系统、更加全面、更深层次的设计与测试，找到并修复作品的各种漏洞,以提高作品 WebShell 的检测能力。

参考文献

[1].董师师, 黄哲学. 随机森林理论浅析[J]. 集成技术, 2013

[2].Leo Breiman .Random Forests[J]. Machine Learning,2001

[3].Leo Breiman. Bagging predictors[J]. Machine learning, 1996

[4].周志华.机器学习[M].清华大学出版社,2016

[5].Hochreiter&J.Schmidhuber.Long short-term memory[J]. Neural computation,1997

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

[7].李航.统计学习方法[M].清华大学出版社,2012

[8].朱小虎.理解 LSTM 网络[EB/OL].简书,2015

[9].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

[10].360 互联网安全中心.2017 年中国网站安全形势分析报告[R],2018

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

[12]. 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

[13]. 叶飞,龚俭,杨望. 基于支持向量机的 WebShell 黑盒检测[N].《南京航空航天大学学报》,2015

[14]. 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

[15]. 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

[16]. 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

[17]. 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

[18]. Oleksii Starov,Johannes Dahse †,Syed Sharique Ahmad,Thorsten Holz, Nick Nikiforakis. No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells[C]. WWW '16 Proceedings of the 25th International Conference on World Wide Web, 2016

[19]. Truong Dinh Tu, Cheng Guang, Guo Xiaojun, Pan Wubin. Webshell Detection Techniques in Web Applications[C]. Computing,Communication and Networking Technologies (ICCCNT), 2014 International Conference on ,2014