基于机器学习的webshell检测

要获取到PHP opcode，需要添加一个PHP 的插件 VLD

结构：

1. php文件提取+遍历网站目录（load\_php\_opcode(phpfilename)函数）
2. 提取php文件的opcode信息

执行命令：'php.exe -dvld.active=1 -dvld.execute=0+phpfilename‘

可获得该文件的opcode代码，

-dvld.active=1 是 VLD 的基础参数，表示激活 VLD 模式。

-dvld.execute=0，表示只显示中间代码，不执行生成的中间代码。

在python中可利用subprocess模块的subprocess.check\_output()函数执行这段命令

利用re模块的re.findall()将命令的结果进行正则匹配提取到opcode的有效信息

最后将这些信息用Python join() 方法将序列中的元素以指定的字符连接生成一个新的字符串返回出来。

2、递归获取 php opcde（recursion\_load\_php\_file\_opcode(dir)函数）

通过os.walk() 方法在目录树中游走输出在目录中的文件名,返回的是一个三元组(root,dirs,files)。

其中：

root 所指的是当前正在遍历的这个文件夹的本身的地址

dirs 是一个 list ，内容是该文件夹中所有的目录的名字(不包括子目录)

files 同样是 list , 内容是该文件夹中所有的文件(不包括子目录)

遍历files列表，得到每个文件的文件名，并通过os.path.join(root, filename)拼接得到完整的文件路径，再利用自定义的load\_php\_opcode(phpfilename)函数即可得到整个目录文件中所有php文件的opcode。

1. 训练

分别创建白名单文件夹whitelist和黑名单文件夹blacklist，下载正常的php项目程序代码，放在白名单文件夹里，搜集一些webshell文件放在黑名单文件夹下。

1、生成需要使用的数据，写入文件（prepare\_data()函数）

利用上文已经自定义的recursion\_load\_php\_file\_opcode(dir)函数分别获取whitelist和blacklist文件夹的文件的opcode，将它们存入white\_opcodes.txt和black\_opcodes.txt文件里(在进行这些操作前先检查white\_opcodes.txt和black\_opcodes.txt是否存在，若存在则跳过这步)

分别读取white\_opcodes.txt和black\_opcodes.txt每一行的代码，过滤掉空格，将他们分别存在white\_file\_list和black\_file\_list中

用len（）计算两个列表的长度len\_white\_file\_list和len\_black\_file\_list

把白名单和黑名单中的PHP opcode 分别贴上 【0】和【1】的标签（

y\_white = [0] \* len\_white\_file\_list

y\_black = [1] \* len\_black\_file\_list

）

返回所有PHP opcode 的集合数据 X（有序）和标签y（有序）

X = white\_file\_list + black\_file\_list

y = y\_white + y\_black

1. 整理上文生成的数据（method1()函数）

运用countVectorizer + TF-IDF

cv=CountVectorizer(ngram\_range=(3,3),decode\_error="ignore",token\_pattern=r'\b\w+\b')

X = cv.fit\_transform(X).toarray()

CountVectorizer会将文本中的词语转换为词频矩阵，再通过fit\_transform函数计算各个词语出现的次数。

（其中：ngram\_range=(3, 3)，词组切分的长度范围；

decode\_error="ignore"，遇到不能解码的字符将报UnicodeDecodeError错误，设为ignore将会忽略解码错误

token\_pattern=r'\b\w+\b'， 过滤规则，表示token的正则表达式

）

有些词在文本中尽管词频高，但是并不重要，这个时候就可以用TF-IDF技术。TF-IDF倾向于过滤掉常见的词语，保留重要的词语。具体如下：

transformer = TfidfTransformer(smooth\_idf=False)

X = transformer.fit\_transform(X).toarray()

上面的为用sklearn进行TF-IDF预处理：方法是在用 CountVectorizer 类向量化之后再调用 TfidfTransformer 类进行预处理。

格式化后X之后

使用train\_test\_split函数来获取打乱的随机的测试集和训练集：

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=0)

这时候，黑名单中的文件和白名单中的文件排列顺序就被随机打乱了，但是X[i] 和 y[i] 的对应关系没有改变，训练集和测试集在总数聚集中分别占比60%和40%。

通过GaussianNB ：Scikit-learn 对朴素贝叶斯算法的实现：

gnb = GaussianNB()

gnb.fit(x\_train, y\_train)

执行完这个语句以后，我们就会得到一个已经训练完成的gnb训练对象

（可通过joblib的dump将模型保存到本地）

我们用测试集(X\_test) 去预测得到我们的y\_pred 值（预测出来的类型）：

y\_pred = gnb.predict(x\_test)

metrics.accuracy\_score(y\_test, y\_pred) 对比原本的 y\_test 和 用训练算法得到的结果 y\_pred。

metrics.confusion\_matrix(y\_test, y\_pred)#计算混淆矩阵来评估分类的准确性。

1. 检测

训练完成后，即可将新的检测内容放到gnb中进行检测，判断类型并输出。

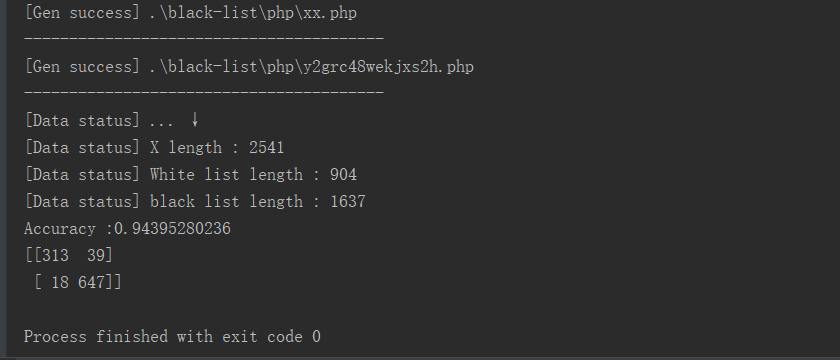
具体操作为：将待检测文件的opcode中的有效内容提取出来

与white\_file\_list + black\_file\_list一起组成X，将他们用CountVectorizer 类向量化之后再调用 TfidfTransformer 类进行预处理，得到格式化之后的X，再通过X[-1:]将格式化后的待检测php的opcode数据拿出来，

gnb = joblib.load('save/gnb.pkl')

y\_p = gnb.predict(X[-1:])

进行预测，判断是否为webshell。



机器学习的步骤：

1. 准备数据
2. 文本特征提取
3. 划分数据集
4. 选择算法进行训练
5. 对比训练结果

webshell的访问特征(主要特征)

少量ip对其发起访问

总的访问次数少

该页面属于孤立页面

防范：

所有用户提交的信息 post get 或是其他形式提交的数据 都要单独写个过滤函数处理一遍，养成习惯(intval，strip\_tags，mysql\_real\_escape\_string)

经常检查有没有一句话木马 eval($\_POST[ 全站搜索php代码有没有这样的源代码

文件要命名规范 至少让自己可以一目了然，哪些php文件名字有问题

如用开源代码，有补丁出来的话，尽快打上补丁