Web日志安全分析系统实践

xman21 / 2018-03-11 20:29:21 / 浏览数 27185

零、前言

在社区看到了这篇日志分析的文章--《Web日志安全分析浅谈》，文章整体写的非常棒，对日志分析的作用、难点、工程化建设和攻击溯源等方面进行了全面的描述。去年的

一、系统设计

在开发一个项目之前当然要先做好设计，明白自己想要的是一个什么系统，可以使用哪些技术、算法和硬件设备。我们分成功能设计、数据库设计、算法结构设计、硬件拓扑

1.1功能设计

系统应包括系统监控、用户管理（系统使用人员）、日志管理、实时分析、离线分析等功能，并为用户提供可视化的操作、分析与结果展示界面。功能结构图如图所示：

1.2数据设计

系统使用MySQL数据库，库中需要建立logmanagement数据库，拥有user、offline、online三个数据表，分别为用户表、离线数据表、在线数据表。数据库中的数据表如下

offline数据表用于存储离线日志的分析结果，每一个上传的日志文件对应一条记录，包括名称、大小、类型、起止日期、访问量最高的前10个IP地址、访问量最高的前10个U

online数据表用于存储实时分析的中间结果，数据表的结构如下：

user表是管理员的用户表，用来存储管理员的个人信息。

1.3算法结构设计

系统使用了三种机器学习算法进行恶意攻击的识别：逻辑回归、支持向量机和朴素贝叶斯。同时包含了传统的正则匹配算法，正则虽然无法识别未知攻击，但是在已知攻击的

1.4硬件拓扑设计

为了实现系统对日志的高效收集，使用了Flume框架；为了具有大数据的处理能力，使用了Spark和HDFS做计算和存储。其中Flume与HDFS是完美兼容的，可以很方便的实

1.5前端界面设计

为了提供一个良好的用户交互性能，需要一个便捷的可视化界面，这里选用Flask框架开发一个Web管理平台，包含对服务器状态的监控、日志的管理以及分析结果的可视化

1.6主框架设计

主框架要能够说明系统的总体功能及数据流走向，其中，日志获取有两种途径，Web界面负责接收用户的离线上传，Flume负责实时获取；HDFS负责日志存储，自动将获取

离线分析就是用户通过Web界面将文本日志文件上传进行分析，相对简单，实时分析就需要严格控制数据流的走向。这里就像一个生产者与消费者的模型，Flume不断收集日
Streaming不断的从HDFS读取日志（消费），实时结构如下：

二、系统实现

2.1日志预处理

我们知道一条日志大概是这样的

```
115.28.44.151 - - [28/Mar/2014:00:26:10 +0800] "GET /manager/html HTTP/1.1" 404 162 "-" "Mozilla/3.0 (compatible; Indy Library
```

字段含义为：远程IP - 用户名 时间 请求主体 响应码 请求字节 请求来源 客户端信息
想要对日志进行识别分析，首先要对各字段进行提取，其中攻击识别主要依靠"请求主体"，我们可以如下正则进行提取

```
log_Pattern = r'^(?P<remote_addr>.*?) - (?P<remote_user>.*) \[(?P<time_local>.*?)\] "(?P<request>.*?)" '\
       '(?P<status>.*?) (?P<body_bytes_sent>.*?) "(?P<http_referer>.*?)" "(?P<http_user_agent>.*?)"$'
```

2.2正则匹配

算法的匹配正则来自与网络和一些CMS厂商的的正则代码，经过多次修改测试可以识别常见的已知的Web攻击，包括SQL注入、XSS攻击、命令执行等常见Web漏洞。比如

```
self.SQL_pattern = """/select(\s)+|insert(\s)+|update(\s)+|(\s)+and(\s)+|(\s)+or(\s)+|delete(\s)+|\'|\/\*|\*|\.\.\/
       |\.\/|union(\s)+|into(\s)+|load_file(\s)+|outfile(\s)+"""
self.Webshell_pattern = """(preg_replace.*\/e|`.*?\$.*?`|\bcreate_function\b|\bpassthru\b|\bshell_exec\b|\bexec\b|
       \bbase64_decode\b|\bedoced_46esab\b|\beval\b|\bsystem\b|\bproc_open\b|\bpopen\b|\bcurl_exec\b|\bcurl_multi_exec\b|
       \bparse_ini_file\b|\bshow_source\b|cmd\.exe|KAdot@ngs\.ru|■■■■■■|■■|■■|PHP\s?■■|shell\s?■■■|
       WScript\.shell|PHP\s?Shell|Eval\sPHP\sCode|Udp1-fsockopen|xxddos|Send\sFlow|fsockopen\('(udp|tcp)|SYN\sFlood)|
       z0|z1|z2|z9|caidao"""
self.XSS_pattern = """xss|javascript|vbscript|expression|applet|meta|xml|blink|link|style|script|embed|object|
       iframe|frame|frameset|ilayer|layer|bgsound|title|base|onabort|onactivate|onafterprint|onafterupdate|
       onbeforeactivate|onbeforecopy|onbeforecut|onbeforedeactivate|onbeforeeditfocus|onbeforepaste|onbeforeprint|
       onbeforeunload|onbeforeupdate|onblur|onbounce|oncellchange|onchange|onclick|oncontextmenu|oncontrolselect|
       oncopy|oncut|ondataavailable|ondatasetchanged|ondatasetcomplete|ondblclick|ondeactivate|ondrag|ondragend|
       ondragenter|ondragleave|ondragover|ondragstart|ondrop|onerror|onerrorupdate|onfilterchange|onfinish|onfocus|
       onfocusin|onfocusout|onhelp|onkeydown|onkeypress|onkeyup|onlayoutcomplete|onload|onlosecapture|onmousedown|
       onmouseenter|onmouseleave|onmousemove|onmouseout|onmouseover|onmouseup|onmousewheel|onmove|onmoveend|onmovestart|
       onpaste|onpropertychange|onreadystatechange|onreset|onresize|onresizeend|onresizestart|onrowenter|onrowexit|
       onrowsdelete|onrowsinserted|onscroll|onselect|onselectionchange|onselectstart|onstart|onstop|onsubmit|
       onunload(\s)+"""
```

所有的攻击类型如下所示

2.3数值统计

在所采集海量日志文本中，包含了大量用户行为、交互IP、访问次数等信息，这些信息所表现出的统计特征可以明确地表达一个网络动作，而有些动作通过传统的规则匹配、

在实现中只进行了一定时间内某ip访问频率的计算

```
def check(self,dataRDD,sc):
   """■■■■■■■■■■■■■■■■"""
   data_Memory = dataRDD.collect()
   start = data_Memory[0]
   temp_Time = time.strptime(start[2], "%d/%m/%Y:%H:%M:%S")
   start_Time = datetime.datetime(temp_Time[0],temp_Time[1],temp_Time[2],temp_Time[3],temp_Time[4],temp_Time[5])
   data_Min = [] #■■■■■■■■■■■■■■
   data_Result = []
   label = self.label
   for line in data_Memory:
       temp_Time = time.strptime(line[2], "%d/%m/%Y:%H:%M:%S")
       end_Time = datetime.datetime(temp_Time[0],temp_Time[1],temp_Time[2],temp_Time[3],temp_Time[4],temp_Time[5])
       if (end_Time-start_Time).seconds <= 10:
           data_Min.append(line)
       else:
           data_Result += label(data_Min)
           start_Time = end_Time
           data_Min = []
           data_Min.append(line)
   tempRDD = sc.parallelize(data_Result)
   return tempRDD
```

2.4特征向量

使用机器学习算法的前提是构造好的特征向量，日志的识别主要是针对日志记录中的request、referer和user-agent。request、referer都是URL路径，user-agent是浏览器

和基于机器学习的web异常检测，训练集分为两个部分，一个是恶意的在URL请求，主要收集于github中知名的payload仓库，大约有30000条数据，其中包括SQL注入、Tr

二是正常的URL请求，测试部分包括日志中的request、referer和user-agent，其中request和referer的正常样本基本一致，都是URL请求地址，user-agent虽然并不是URL

User-agent是指了各大浏览器厂商正常的名称，训练集中正常请求部分样例如下：

将上述的训练集一分为二，90%作为训练集集，10%作为测试集并进行打标用于测试。在真正分类的时候，将所有的日志依据request、referer和user-agent这三个部分进行

首先通过长度为N的滑动窗口将文本分割为N-Gram序列，例子中，N取2，窗口滑动步长为1，可以得到如下N-Gram序列：

其中N的取值需要进行多次试验，不同的算法最佳值不同。然后声明一个长度为10000的特征向量，将这些序列映射到特征向量中，并使用TF-IDF生成特征向量的值。词频—

```
def TFIDF(self,badData,goodData,distance,step):
   '''IT-IDF■■■■■■■■■■■■■■■TF-IDF■■'''
   tf = self.tf
   badFeatures = badData.map(lambda line: tf.transform(split2(line,distance,step)))
```

```
    goodFeatures = goodData.map(lambda line: tf.transform(split2(line,distance,step)))
    badFeatures.cache()
    goodFeatures.cache()
    idf = IDF()
    idfModel = idf.fit(badFeatures)
    badVectors = idfModel.transform(badFeatures)
    idfModel = idf.fit(goodFeatures)
    goodVectors = idfModel.transform(goodFeatures)
    badExamples = badVectors.map(lambda features: LabeledPoint(1, features))
    goodExamples = goodVectors.map(lambda features: LabeledPoint(0, features))
    dataAll = badExamples.union(goodExamples)
    return dataAll
```

一个TF-IDF向量如下所示：

其中第一项0.0是向量的标签，表示这是一条恶意的请求，后面是各个分词序列在投影后的坐标及其TF×IDF值。

2.5机器学习算法

三种算法训练完毕后以后的检测只需从本地加载模型即可

```
def train(self,sc):
    # #■■Logistic■SVMWithSGD■■■■
    # dataLogistic = self.TFIDF(bad,good,3,1)
    # #■■SVMWithSGD■■■■
    # dataSVMWithSGD = self.TFIDF(bad,good,3,1)
    # #■■NaiveBayes■■■■
    # dataNaiveBayes = self.TFIDF(bad,good,2,1)
    # ■■■■■■■■■■,iterations■■■■■,step■■■■■
    # modelLogistic = LogisticRegressionWithSGD.train(data=dataLogistic,iterations=10000,step=6)
    # print "train success1"
    # modelLogistic.save(sc,"model/modelLogistic")
    # modelSVMWithSGD = SVMWithSGD.train(data=dataSVMWithSGD,iterations=10000,step=5)
    # print "train success2"
    # modelSVMWithSGD.save(sc,"model/modelSVMWithSGD")
    # modelNaiveBayes = NaiveBayes.train(data=dataNaiveBayes,lambda_=0.1)
    # print "train success3"
    # modelNaiveBayes.save(sc,"model/modelNaiveBayes")
    self.modelLogistic = LogisticRegressionModel.load(sc,"modelLogistic")
    self.modelSVMWithSGD = SVMModel.load(sc,"modelSVMWithSGD")
    self.modelNaiveBayes = NaiveBayesModel.load(sc,"modelNaiveBayes")

def check_Line(self,line,algorithm):
    """■■■■"""
    tf = self.tf
    request_url = line
    check_Result = 0
    if "Logistic" in algorithm:
        check_Result += self.modelLogistic.predict(tf.transform(split2(request_url,3,1)))
    if "SVM" in algorithm:
        check_Result += self.modelSVMWithSGD.predict(tf.transform(split2(request_url,3,1)))
    if "NaiveBayes" in algorithm:
        check_Result += self.modelNaiveBayes.predict(tf.transform(split2(request_url,2,1)))
    print check_Result
    print "model check  :  "+str(check_Result)
    if check_Result>2:
        line.append([-1])
    else:
        line.append([])
    return line

def check(self,test,sc,algorithm="Logistic,SVM,NaiveBayes"):
    """■■■■■■"""
    self.train(sc)
    check_Line = self.check_Line
    temp1 = test.map(lambda line: check_Line(line,algorithm))
    return temp1.collect()
```

三、系统展示

离线日志分析
离线分析包括分析报表和日志管理两个子功能，用户需要在日志管理处上传日志才可通过分析报表查看分析结果（如果直接点击分析报表界面则默认显示最近一次的分析结果

点击每一条记录右侧的查看按钮，即可跳到相应的分析报表界面，分析报表界面包含5个部分，分别为基本信息、访问次数最高的前10个IP、访问次数最高的前10个URL、攻

实时日志分析
实时分析部分包含两个显示界面，一个是访问次数（蓝色）与攻击次数（黑色）的双曲线图表，表示当前时间访问的次数以及当中可能包含的攻击次数，两者同时显示，相互

四、一些问题

1、程序运行起来虽然看起来还可以，但是识别率其实比较一般，一是正则写的不够完善；二是机器学习误报有点高，如果把判别条件放太宽，会出现一些低级分类错误。
2、算法中机器学习其实只是一个二分类，具体的攻击类别要靠正则识别，正则识别不出来而算法识别出来的则为未知攻击类型。
3、这里的实时其实是伪实时。

五、参考文献

http://www.freebuf.com/articles/web/126543.html
http://www.freebuf.com/articles/web/134334.html
http://www.freebuf.com/sectool/126698.html
http://blog.csdn.net/xnby/article/details/50782913

上一篇：某协同系统漏洞利用汇总 下一篇：Encryption 101系列：...

1. 4 条回复


simeon 2018-03-11 22:08:25

好东西，哥们能给一个联系方式吗。

0 回复Ta

北风飘然 2018-03-11 22:59:49

好东西 顶一个 不过mysql这类东西 流量不算太大得公司还可以 像二三线互联网公司估计性能上就差了 不知道用elasticsearc怎么样= =

0 回复Ta

xman21 2018-03-12 09:08:34

@simeon xman_sec@163.com

1 回复Ta

Zthero 2018-04-02 10:57:51

嘿嘿嘿，好东西，学习下，好熟悉的ID，
Xman21的小学弟
Z.thero

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板