

Developer Guide

LogAnalyzer

Structure

1. [ApacheLogAnalyzer.py](#)

This python script reads apache logs line by line and parse the line as W3C extended log format. Then, it goes through the following process.

- I. list of unique IP addresses as a flat text file
- II. list of unique IP addresses with country and number of hits as a flat text file
- III. list of all activity per IP address to individual flat text files per IP
- IV. detect SQLi with found entries to flat text file
- V. detect remote file inclusion with found entries to flat text file
- VI. detect web shells with found entries to flat text file

All files are saved as csv file format and detect SQLi, RFI, webshell using regex filter.

2. [filters](#)

Filter directory has three xml files. These xml files have regular expression filter, so

[ApacheLogAnalyzer.py](#) read these xml and can detect sqli, rfi, webshell

- I. rfi_filter.xml
- II. sqli_filter.xml
- III. webshell_filter.xml

Pre-Installation

1. geolite2

For getting a country information from IP address, we need geolite2

pip install maxminddb-geolite2

2. tqdm

tqdm instantly make loops show a smart progress meter

pip install tqdm

Code-Description

1. list of unique IP addresses as a flat text file

To separate fields from line, I used regular expression.

```
1. p = re.compile('(P<date>\d{4}-\d{2}-\d{2}) (P<time>\d{2}:\d{2}:\d{2}) (P<s_ip>\d{1,3}[\.]\d{1,3}[\.]\d{1,3}[\.]\d{1,3}) (P<cs_method>\S+) (P<cs_uri_stem>\S+) (P<cs_uri_query>.+?) (P<s_port>443|80) (P<cs_username>.+?) (P<c_ip>\d{1,3}[\.]\d{1,3}[\.]\d{1,3}[\.]\d{1,3}) (P<User_Agent>.+?) (P<Referer>[-]|<|a href.*|([?http.*|\S+) (P<sc_status>\d+) (P<sc_substatus>\d+) (P<sc_win32_status>\d+) (P<time_taken>\d+)')
```

```
2. m = p.match(line)
```

By grouping fields, I could get an IP address from line.

```
1. ip = m.group('c_ip')
2. ip_info = ip_info_reader.get(ip)
```

Then, I could check unique using dictionary data structure.

```
1. if(not ip in ip_info_dict):
2.     ip_info_dict[ip] = [country, 1]
3.     writer1.writerow([ip])
```

If IP address are not in dictionary, script add IP address to dictionary and write to csv file by writer1.writerow API.

2. list of unique IP addresses with country and number of hits as a flat text file

I could get country information from IP address using python module 'geolite2'

```
1. ip_info_reader = geolite2.reader()
2. ip_info_dict = {}
3. ip_info = ip_info_reader.get(ip)
4. country = ip_info['country']['names']['en']
```

And I inserted value (country, hit) to key value in dictionary. If line get a not unique IP

address, script increase hit. Hit's initial value is 1.

```
1. if(not ip in ip_info_dict):
2.     country=''
3.     if ip_info == None :
4.         country = 'NO INFO'
5.     elif 'country' in ip_info:
6.         country = ip_info['country']['names']['en']
7.     elif 'continent' in ip_info:
8.         country = ip_info['continent']['names']['en']
9.     else:
10.        country = 'NO INFO'
11.        ip_info_dict[ip] = [country, 1]
12.        writer1.writerow([ip])
13.    else :
14.        ip_info_dict[ip][1] += 1
```

Because of the hit, script doesn't write to csv file directly. After all lines are read, script write

all IP and IP's information to csv file at once.

```

1. for ip in ip_info_dict:
2.     writer2.writerow([ip]+ip_info_dict[ip])

```

3. list of all activity per IP address to individual flat text files per IP

activity is consist of all files of log format.

```

1. activity = list(m.groups())

```

In each line, script write activity to individual csv file.

```

1. csvfile3 = open('activity/'+ip+'.csv', 'a')
2.     writer3 = csv.writer(csvfile3)
3.     writer3.writerow(activity)
4.     csvfile3.close()

```

Because of this step, this script spend pretty long time to process all lines.

4. detect SQLi with found entries to flat text file

SQL injection needs manipulation of uri query. So what can I do for detecting SQL injection is applying regex to uri query. Script parse the xml which has SQL injection detection regex, and match regex with uri query of each lines.

```

1. for element in sqli_filter :
2.     rule = element.findtext("rule")
3.     p = re.compile(rule)
4.     m = p.match(cs_uri_query)
5.     if m:
6.         description = element.findtext("description")
7.         writer4.writerow(activity + [description])
8.         break

```

XML has the following code form. XML has many number of filters, and script uses all filters

when detecting SQL injection. I can get this filter from

https://github.com/PHPIDS/PHPIDS/blob/master/lib/IDS/default_filter.xml.

```

1. <filter>
2.     <id>82</id>
3.     <rule><![CDATA[/exec(\s|\+)(s|x)p\w+/ix]]></rule>
4.     <description>Regex for detecting SQL Injection attacks on a MS SQL Server</description>
5.     <tags>
6.         <tag>sqli</tag>
7.     </tags>
8.     <impact>3</impact>
9. </filter>

```

5. detect remote file inclusion with found entries to flat text file

This process is similar to before. Scripts parse rfi XML, and match regex with uri query.

```

1. for element in rfi_filter :
2.     rule = element.findtext("rule")
3.     p = re.compile(rule)
4.     m = p.match(cs_uri_query)
5.     if m:
6.         description = element.findtext("description")
7.         writer5.writerow(activity + [description])
8.         break

```

One example of xml is this.

```

1. <filter>
2.     <id>14</id>
3.     <rule><![CDATA[(?:#@\~^\w+)|(?:\wscript:|@import[^\w]|;base64|base64,)|(?:\w
4.         \s*\([\w\s]+,[\w\s]+,[\w\s]+,[\w\s]+,[\w\s]+,[\w\s]+\)))]></rule>
5.     <description>Detects possible includes, VBScript/JScript encoded and packed
6.         functions</description>
7.     <tags>
8.         <tag>xss</tag>
9.         <tag>csrf</tag>
10.        <tag>id</tag>
11.        <tag>rfe</tag>
12.    </tags>
13.    <impact>5</impact>
14. </filter>

```

6. detect web shells with found entries to flat text file

In the beginning I couldn't find how can I detect webshell with only apache log. But fortunately, I saw 'shell.php' in uri_stem. At least if attacker did not change webshell's name, script can detect webshell using famous webshell list. So I insert webshell's name into filter. Script match webshell's name with both uri_query and uri_stem.

```

1. for element in webshell_filter :
2.     rule = element.findtext("rule")
3.     p = re.compile(rule)
4.     m = p.match(cs_uri_stem+cs_uri_query)
5.     if m :
6.         description = element.findtext("description")
7.         writer6.writerow(activity + [description])
8.         break

```