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

How to use

python ApacheLogAnalyzer.py (log file path)

For example, ‘python ApacheLogAnalyzer.py ./CTF1.log’

Then, you will get following files and directory.

1. SimpleIPList.csv
   - Unique IP address are in lines
2. DetailIPList.csv
   - Unique IP address with country and number of hits are in lines
3. Activity
   - Activity is directory and there are many (unique IP).csv files in Activity. Each csv files have activity which is consist of W3C extended log format fileds per unique IP address.
4. sqli_list.csv
   - It consists of suspicious lines as SQL injection.
5. rfi_list.csv
   - It consists of suspicious lines as Remote File Inclusion.
6. webshell_list.csv
   - It consists of suspicious lines as web shell
7. exception.txt
   - If some line is not matched with W3C extended log format, the line is saved in exception.txt

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_substa-
    tus>\d+) (?P<sc_win32_status>\d+) (?P<time_taken>\d+)')
2.  m = p.match(line)
```

   By grouping fileds, 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.

```
if(not ip in ip_info_dict):
 ip_info_dict[ip] = [country, 1]
 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'

```
ip_info_reader = geolite2.reader()
ip_info_dict = {}
ip_info = ip_info_reader.get(ip)
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.

```
if(not ip in ip_info_dict):
            country=''
            if ip_info == None :
                country = 'NO INFO'
            elif 'country' in ip_info:
                country = ip_info['country']['names']['en']
            elif 'continent' in ip_info:
                country = ip_info['continent']['names']['en']
            else:
                country = 'NO INFO'
            ip_info_dict[ip] = [country, 1]
            writer1.writerow([ip])
        else :
            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.

```
for ip in ip_info_dict:
        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.

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

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

```
csvfile3 = open('activity/'+ip+'.csv', 'a')
        writer3 = csv.writer(csvfile3)
        writer3.writerow(activity)
        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.

```
for element in sqli_filter :
            rule = element.findtext("rule")
            p = re.compile(rule)
            m = p.match(cs_uri_query)
            if m:
                description = element.findtext("description")
                writer4.writerow(activity + [description])
                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.

```
<filter>
    <id>82</id>
    <rule><![CDATA[/exec(\s|\+)+(s|x)p\w+/ix]]></rule>
    <description>Regex for detecting SQL Injection attacks on a MS SQL Server</descr
iption>
    <tags>
        <tag>sqli</tag>
    </tags>
    <impact>3</impact>
</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.

```
for element in rfi_filter :
            rule = element.findtext("rule")
            p = re.compile(rule)
            m = p.match(cs_uri_query)
            if m:
                description = element.findtext("description")
                writer5.writerow(activity + [description])
                break
```

One example of xml is this.

```
<filter>
        <id>14</id>
        <rule><![CDATA[(?:#@~\^\w+)|(?:\wscript:|@import[^\w]|;base64|base64,)|(?:\w
\s*\([\w\s]+,[\w\s]+,[\w\s]+,[\w\s]+,[\w\s]+,[\w\s]+\))]]></rule>
        <description>Detects possible includes, VBSCript/JScript encodeed and packed
 functions</description>
        <tags>
            <tag>xss</tag>
            <tag>csrf</tag>
            <tag>id</tag>
            <tag>rfe</tag>
        </tags>
        <impact>5</impact>
    </filter>
```

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

   In the beginning I could'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.

```
for element in webshell_filter :
            rule = element.findtext("rule")
            p = re.compile(rule)
            m = p.match(cs_uri_stem+cs_uri_query)
            if m :
                description = element.findtext("description")
                writer6.writerow(activity + [description])
                break
```