Global Information Assurance Certification Paper

Log2Pcap

GIAC (GCIA) Gold Certification

Author: Joaquín Moreno Garijo, bastionado@gmail.com
Advisor: Johannes Ullrich

Accepted: October 4th 2012
(Date your final draft is accepted by your advisor)

Abstract

While handling an incident, either in the identification phase or during the computer forensics analysis, it is necessary to analyze the logs from different servers to identify the events that could be related to the incident. This task is often done using regular expressions with a customized list of patterns designed to identify unusual behavior. This is a time consuming process, requiring up to date signatures. This paper describes a novel tool, Log2pcap, which converts server logs to the standard "pcap" format. A pcap file can then be analyzed using IDS engines like Snort leveraging existing and well maintained signature collections.

1. Introduction

During the analysis of all the available data that are logged, organizations must be able to identify which portions of this information are actionable and pertinent. This tedious process is related with the normalization process, and also, with the parsing log process (Paul, 2011). To help with this tiring work, people devoted to incident handling and computer forensics usually have programs that search for patterns in those logs and identify potential dangerous events (msbachman, 2010) (Worman, 2009).

This paper aims to create a new type of tool to help with this task. The objective is to use the already existing rules in the deployed IDSs (Intrusion Detecting Systems) as a detection method, instead of the traditional parse log tools with white and black lists. To do so, the tool will regenerate the network traffic using the server logs. The network traffic files will then be loaded into the IDS to detect the anomalous or potentially dangerous behaviors.

The chosen programming language has been Python with the Scapy library to recreate the network traffic. The IDS used in this paper is Snort.

1.1. Scapy

Scapy is an interactive packet manipulation tool. It is able to forge network traffic, decode traffic, packet sniffer, send traffic, receive traffic and more functionality. Scapy uses the Python interpreter as a command board (Secdev Scapy project, 2012).

Scapy is probably one of the best libraries to craft network packets without requiring a complex language (Biondi, 2005) and it makes it easy to develop future extensions to the Log2Pcap tool. However, Scapy is not a fast library compared to some C libraries. This is not a critical issue in the case of the Log2Pcap, as the information it is not processed in real time and therefore the impact of the lower speed is not that critical, compared to other tools that need to process network traffic in real time. For this reason, Scapy has been the library chosen to develop Log2Pcap.

Due to the fact that Scapy is implemented in Python, Log2Pcap has been developed in this language.

1.2. Snort

Snort is an open source network intrusion detection system write in C language. Providing technical capacity to perform real-time traffic, protocol analysis and packet logging. It can be used to detect different types of attacks and deviation from normal or expected behavior such as buffer overflows, stealth port scans, malware, web attack and so on (Caswell, Beale, & Baker, 2007).

The tool is common used in knowledge-based approaches, but it also can be use as behavior-based approaches (Debar, 2013) using the dynamic preprocessors (Ashley, 2008).

Snort has been chosen because it is one of the most used open source IDSs and it can read PCAP files, which is a requirement to use Log2Pcap. Furthermore, it is well documented and it has been used in different organizations, countries and CERTS, as an example of the stability and reliability of the tool (US CERT, 2011).

The proposed alternative to Snort is Suricata (http://suricata-ids.org), that has good performance in multithread processors, but it has less documentation than Snort (Albin, 2011).

1.3. Emerging Threats and Source Fire VRT

To detect the possible threats the Snort engine uses an open source rule language, dynamic rules and two specific types of preprocessors: the core preprocessor and the dynamic preprocessor (Caswell, Beale, & Baker, 2007). There are many different rule sets that can be added to the Snort engine to increase the type of threats detected. Some of them have been specifically designed to detect certain malware or exploits, but it is worth to mention that some of the more specialized sets of rules, usually programmed in C, are not free.

Among the free packs of rules (disregarding in this point the license of use of those rules, sometimes limited to personal use) there are two sets that are the most often used in the Snort environment: the one from the Source Fire VRT (www.snort.org/vrt/) and the one from Emerging Threats (www.emergingthreats.net).

In this paper we are going to use mainly the free Source Fire VRT rules, and in some cases both. Although both sets have their pros and cons, it is not the objective of this paper to discuss this.

2. Log2Pcap

Log2Pcap is a proof of concept about how we can combine the Scapy library to recreate the network traffic in order to take advantage of the powerfulness of the IDS and its rules. Due to the fact that there are many different types of protocols and formats of server logs, for the first version of this tool, currently it only supports a few types of logfiles from web servers using the HTTP protocol.

2.1. Uses

The tool was designed with three different purposes:

1. Computer forensics analysis of logs from servers that are potentially compromised or involved in a security incident. The server logs can be obtained from a SIM or a centralized log and the tool can be used to find evidence to help the security experts analyzing the incident. Although the server logs can be taken from the server, this information may have been modified by the intruder and should not be considered reliable.
2. To help in the identification phase of the incident. The tool can be used as part of an unassisted batch process to find potential attacks hidden in the server logs.
3. To evaluate the network perimeter protection: firewalls, IPS, WAF… If the IDS detect attacks from the Log2Pcap output that the perimeter devices did not detect, it may be a signal that your protection infrastructure is not doing its job.

2.2. Limitations of the tool

The tool has currently some limitations that a user needs to be aware of before using it. First of all the emulation of the traffic depends on the amount of information the

logs provide. Some event logs do not save all the information needed to recreate the whole network traffic. In that case Log2Pcap will recreate only part of the traffic.

As an example, some web servers do not log the POST parameters, but they log all the GET parameters. In this sense, it is important to configure logging correctly in order to save all the information that may be necessary in the future, taking into account that this may impact the performance and disk usage of the server. Depending on the service has to use a specific guide to configure the log files, as an example:

- Tomcat: http://tomcat.apache.org/tomcat-7.0-doc/logging.html
- Apache: http://httpd.apache.org/docs/2.2/logs.html
- Nginx: http://wiki.nginx.org/Configuration#Logging

The second limitation comes from the fact that the logs may not be reliable enough. They can be altered by the intruder before they are copied to a safe environment or the server is halted by a security expert to collect evidences during the computer forensics analysis. For this reasons the logs should be always, when possible, taken from a secure remote log server. Another natural limitation of Log2Pcap is that it needs an IDS that supports the PCAP file as input. If the IDS does not support this format, it is necessary to use additional software such Tcpreplay to read the traffic and send it to the mirror port of the IDS. The development of this environment will increase the time and cost of Log2Pcap.

Another relevant limitation is the time needed to recreate the traffic from a log file to a PCAP file and to inject this file to the IDS. Currently to apply a regular expression with a black list or white list to the log is faster than using Log2Pcap.

The last limitation is the loss of correlation between the IDS alert and the line of the log registry that raised the alert. When the IDS reports the alerts after being “fed” with the PCAP file, the security expert loses context information, as he does not know which line of the log generated the alert.

3. Design of the tool

The program has been designed in layers for different purpose to make it easier to extend and improve in the future. They are the parse log layer, the protocol layer and the Scapy function layer as described in the next section.

3.1. Modulation Program

In order to allow future developments of the tool it is important to create a modular application.

The first level is the log parser: it reads the logs and extracts the information needed for the next level, commonly referred as normalization. The second level is the protocol level that needs the basic fields provided by the log parser in order to recreate the protocol. The last level is called by the protocol level to recreate the network traffic using Scapy, as we can see in the next picture:

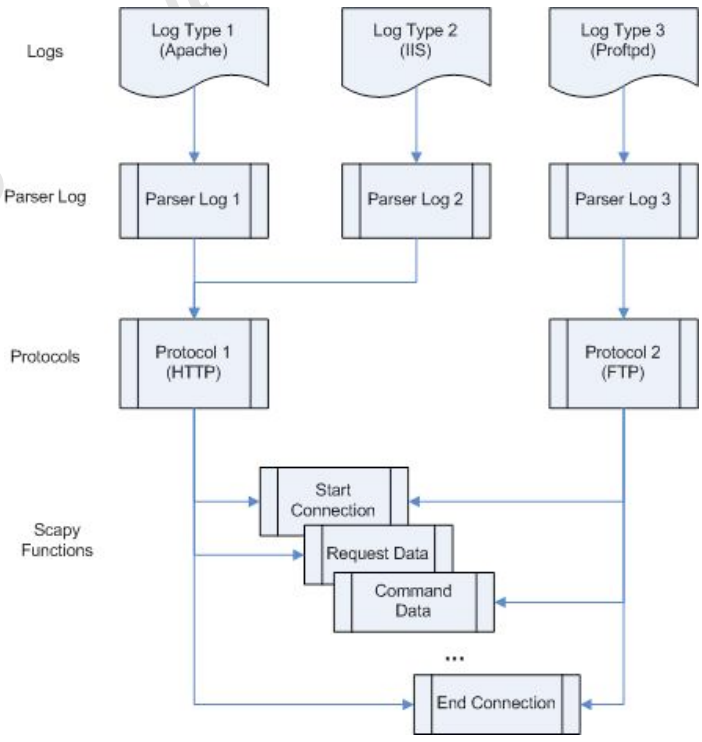

Image 1: diagram layer of the application.

The main advantage of this model it is shown in the example, where a new type of log from a webserver such as Apache Tomcat is used as the source log. To parse this new log format, it would be necessary to develop only one new function to parse the log, that function will then call the standard protocol HTTP function.

3.2. HTTP Protocol

Due to the fact that developing support for all the protocols for all the different types of potential network traffic would require a lot of time and that this paper is a proof of concept with limited assigned time, the first version of the application only has support for the HTTP Protocol.

The Protocol is used very widely and its analysis is required for most incidents, which makes support for it a requirement for computer forensics and intrusion detection.

3.3. Logs support

Log2Pcap currently supports four different webserver log formats with the common or default log configuration. For customized event log formats the user has to adapt the program to match the specific event log configuration.

This part is one of the most challenging tasks because many operating systems have their own log file format that can be changed by the administrator to adapt to his/her needs. Furthermore, some operating systems do not provide the basic information of how they save the log by default.

For this reason Log2Pcap currently only offers support for four different webserver logs, as an example of how the security expert or administrator must adapt the tool to support specific log formats.

3.3.1. Microsoft IIS

Microsoft Internet Information Services or IIS has different types of logs and allows log format customization by the administrator. Log2Pcap supports the two main default log formats: IIS W3C Extended and IIS Log Format.

The first and most common is **W3C Extended**. There is a detailed explanation of the different fields of the log file at the Log Analyzer web site

(*http://loganalyzer.adiscon.com*). The most important for the tool are the following fields:

```
- - SourceIp - DestinationIP HTTPMethod URI - ReturnCode - - - - ProtocolVersion SomeHTTPHeaders
```

A real example of the log can be obtained from the Web of Nihuo Software (Nihuo, 2013):

```
1998-11-19 22:48:39 206.175.82.5 - 208.201.133.173 GET /global/images/navlineboards.gif - 200 540 324 157 HTTP/1.0
Mozilla/4.0+(compatible;+MSIE+4.01;+Windows+95) USERID=CustomerA;+IMPID=01234 http://www.loganalyzer.net
2002-05-02 17:42:15 172.22.255.255 - 172.30.255.255 80 GET /images/picture.jpg – 200
Mozilla/4.0+(compatible;MSIE+5.5;+Windows+2000+Server)
2002-05-24 20:18:01 172.224.24.114 - 206.73.118.24 80 GET /Default.htm - 200 7930 248 31
Mozilla/4.0+(compatible;+MSIE+5.01;+Windows+2000+Server) http://64.224.24.114/
```

The second log format is the IIS Log Format, which can be obtained also from the Web of Nihuo Software (Nihuo, 2013) or the online documentation in the Microsoft (Microsoft TechNet, 2003). The most important fields are the following:

```
SourceIP, -, -, -, -, -, DestinationIp, -, -, -, ReturnCode, -, HTTPMethod, URI, SomeHTTPHeaders
```

A real log entry using this format is shown below:

```
192.168.114.201, -, 03/20/01, 7:55:20, W3SVC2, SALES1, 172.21.13.45, 4502, 163, 3223, 200, 0, GET, /DeptLogo.gif, -,
```

3.3.2. Apache

For the Apache Web Server the tool accepts the default format *Apache Combined* which has the following format:

```
SourceIP - - [] “HTTPMethod URI HTTPProtocol” ReturnCode - - SomeHTTPHeaders
```

A real log file has been obtained from the author own server (Debian 6) and from one challenger of the Honeynet Project (Raffael, Chuvakin, & Tricaud, 2010). This format was verified in other two servers with other Linux distributions, Ubuntu 10.4 LTS and Red Hat EL 5:

```
Own Server:
65.96.235.XXX - - [03/Sep/2012:21:41:57 +0000] "GET /index.php3 HTTP/1.1" 404 120259 "-" "Mozilla/5.00 (Nikto/2.1.5)
(Evasions:None) (Test:multiple_index)"
```

```
HoneyNet Project:
10.0.1.XXX - - [24/Apr/2010:14:33:22 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1"
SxPq9AoAAQ4AAEHGCtEAAAAD 967817
```

3.3.3. IBM WebSeal

The default log from IBM WebSeal does not provide much data:

```
SourceIP - - [-] "HTTPMethod URI HTTPProtocol" ReturnCode -
```

A sample entry from a IBM WebSeal log looks like the following:

```
XX.XX.XX.XX - Unauth [05/Nov/2012:09:01:42 +0200] "GET / HTTP/1.0" 200 103
XX.XX.XX.XX - Unauth [05/Nov/2012:09:07:29 +0200] "GET /$URL HTTP/1.1" 200 471
```

3.3.4. Nginx

The tool accepts the default log configuration for Nginx, which includes the following fields:

```
SourceIP - - -] "HTTPMethod Uri HTTPProtocol" ReturnCode - "-" "UserAgent"
```

A real example log file has been taken from the Slicehost Web page (SliceHost, 2010):

```
80.154.42.XXX - - [23/Aug/2010:15:25:35 +0000] "GET /phpmy-admin/scripts/setup.php HTTP/1.1" 404 347 "-" "ZmEu"
123.65.150.XX - - [23/Aug/2010:03:50:59 +0000] "POST /wordpress3/wp-admin/admin-ajax.php HTTP/1.1" 200 2
"http://www.example.com/wordpress3/wp-admin/post-new.php" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_4; en-US)
AppleWebKit/534.3 (KHTML, like Gecko) Chrome/6.0.472.25 Safari/534.3"
```


4. Code Tool

As described before, Log2Pcap has been developed in Python using the Scapy library to generate and output the traffic to a PCAP file. The tool has three layers: Scapy layer, Protocol layer and the Log Parser layer.

4.1. Scapy Basic Functions

The Scapy main functions recreate the network traffic using the TCP stack. The tool has five different functions that are being used to support the protocol layer functions.

4.1.1. Handshake function

The purpose of this function is to recreate the three way handshake to establish a TCP connection. The client starts sending a *Syn* flag to the server, then the server replies with a *Syn+Ack* flag and finally the client confirms the packet with an *Ack* flag packet (Stevens & Fall, 2011). The initial sequential number for the source and destination are created by the Python random function "*getrandbits*" with a size of 32 bits which is the length for this field in the TCP header (Foundation, 2012).

The code of the function:

```
def handshake(pcapfile, dstIp, dstPort, srcIp, srcPort):
        seqN = random.getrandbits(32)
        ackN = 0
        pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="S",seq=seqN,ack=ackN))
        ackN = random.getrandbits(32)
        pcapfile.write(IP(dst=srcIp,src=dstIp)/TCP(dport=int(srcPort),sport=int(dstPort),flags="SA",seq=ackN,ack=seqN+1))
        pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="A",seq=seqN+1,ack=ackN+1))
        return (seqN+1, ackN+1)
```

4.1.2. End connection

To finish the connection the tool uses two different functions. The first function is ending TCP connections "politely" with a FIN flag packet in both directions; its name is "*finishconnection*". The second function is the "impolite version" that involves a reset packet from the client; this function is named "*resetconnection*" (Stevens & Fall, 2011).

For each line in the log, the *finishconnection* function requires four packets to end the connection, while *resetconnection* only needs one packet. For this reason, using the *resetconnection* function implies a smaller PCAP file than *finishconnection*. Another difference is that the Log2pcap tool is faster if *resetconnection* is used because it needs to save less data in the file and call Scapy functions fewer times. On the other hand, some IDS and network tools identified the reset packet as a network error, a possible DoS attack or a packet crafter. However, Snort does not have problems using the *resetconnection* function.

As an example, we created two different PCAP file using the *resetconnection* function and other with the *finishconnection* function using the same log as input, and then opened the PCAP file in Wireshark (Burns, et al., 2007):

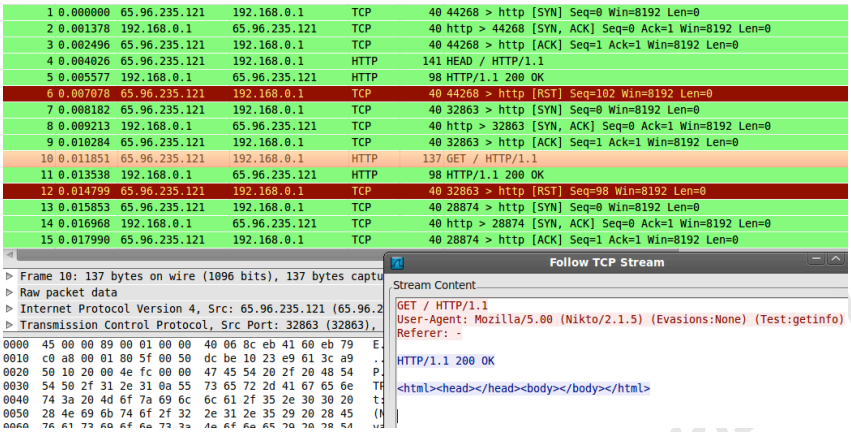

Image 2: PCAP file created using *resetconnection* function

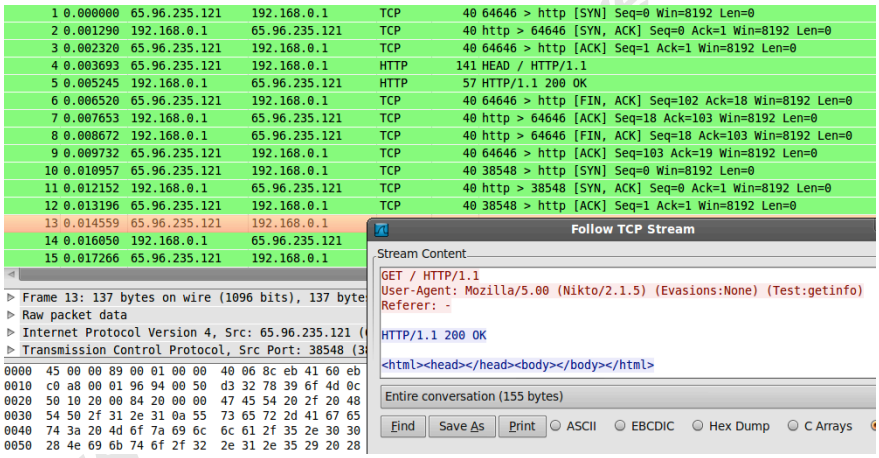

Image 3: PCAP file created using finishconnection function

As we can see Wireshark shows the RST packet in red. For this reason the Log2Pcap tool uses the function *finishconnection* by default, although the user can use *resetconnection* instead of *finishconnection* to increase the performance and reduce the size of the PCAP file.

In the next image a comparative benchmark of speed and file size is shown comparing *finishconnection* and *resetconnection* using an Apache log with 361 lines. Using the *finishconnection* function the file size was 60 kilobyte bigger than using the *resetconnection* function (resulting in 236KB and 176KB respectively) and the processing required 30% more time than in the case of using *resetconnection*:

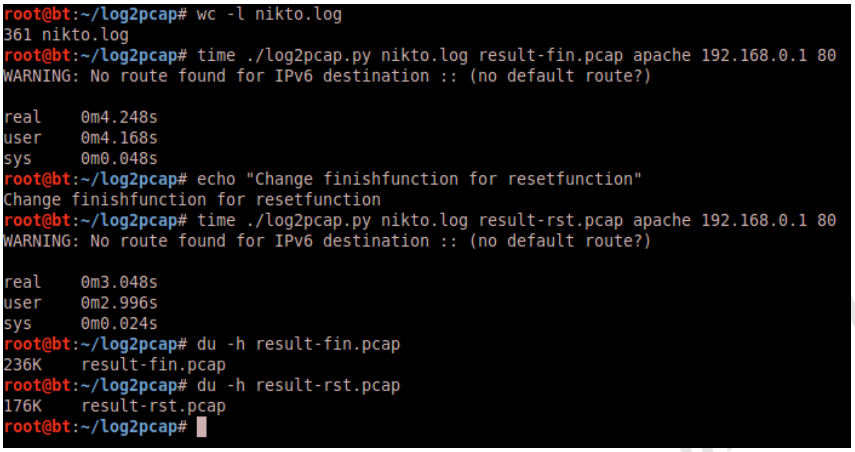

Image 4: Benchmark comparative using Fin o Rst end connection

The code for the *resetconnection* is as follows:

```
def resetconnection(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN):
        pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="R",seq=seqN,ack=ackN))
```

The code for the *finishconnection* is as follows:

```
def finishconnection(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN):
        pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="FA",seq=seqN,ack=ackN))
        seqN = seqN + 1
        pcapfile.write(IP(dst=srcIp,src=dstIp)/TCP(dport=int(srcPort),sport=int(dstPort),flags="A",seq=ackN,ack=seqN))
        pcapfile.write(IP(dst=srcIp,src=dstIp)/TCP(dport=int(srcPort),sport=int(dstPort),flags="FA",seq=ackN,ack=seqN))
        ackN = ackN + 1
        pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="A",seq=seqN,ack=ackN))
```

4.1.3. Send data

Depending on the protocol, data may be send in on direction or it can be send in both directions. For this reason the tool has two different functions to simulate data sending. The first one is "*sendData*", which sends the information in only one way using a function parameter. The other function is "*queryResponse*" which simulates sending the data and receiving a response.

In the *sendData* function the information is in the "*data*" function parameter:

```
def sendData(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN, data):
        pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="A",seq=seqN,ack=ackN)/data)
        if len(query) == 0:
                seqN = seqN + 1
```

```
        else:
                seqN = seqN + len(data)
        return (seqN, ackN)
```

In the function *queryResponse* the data sent is in the function parameter "*query*" and the response data is in the function parameter "*response*":

```
def queryResponse(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN, query, response):
        pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="A",seq=seqN,ack=ackN)/query)
        if len(query) == 0:
                seqN = seqN + 1
        else:
                seqN = seqN + len(query)
        pcapfile.write(IP(dst=srcIp,src=dstIp)/TCP(dport=int(srcPort),sport=int(dstPort),flags="A",seq=ackN,ack=seqN)/response)
        if len(response) == 0:
                ackN = ackN + 1
        else:
                ackN = ackN + len(response)
        return (seqN, ackN)
```

4.2. HTTP Protocol Functions

The second layer of the tool is the Protocol layer. As said above, in this first version Log2Pcap supports only the HTTP Protocol.

This layer has two different functions: "*httpfast*" and "*http*". The *httpfast* function is only required for programmers who need a custom HTTP query or request but it is not intended for use as a standard component for log parsing. The *http* function is the core of the HTTP implementation and it is called by the functions that parse the web server log.

For the implementation of the "*http*" function different types of logs have been analyzed in order to know the type and amount of information they provide, and what of this information is useful for recreate the web traffic. The conclusion is that for the tool to work correctly, the log needs to provide the following HTTP header fields:

- HTTP method: GET, POST, HEAD, OPTIONS, TRACE…
- The URL requested by the client.
- The version of the HTTP protocol.

- The User Agent header field of the client. “It shows information of the web browser used by the web visitor. This is used for statistical purposes and the tracing of protocol violations” (W3C, 1994). Some audit tools and malware have their own signatures.
- The “Referer” header field. “This allows the client to specify, for the server benefit, the address (URI) of the document (or element within the document) from which the URI in the request was obtained” (W3C, 1994).
- The parameters sent by the POST method.
- The return status code from the server. “The data sections of the messages Error, Forward and Redirection responses may be used to obtain human-readable diagnostic information” (W3C, 1998).

The implementation of the *http* function starts generating a random source port for the client higher than 1024 (not reserved port). Then it takes all the values from the function parameters and generates the HTTP query from the client. Finally the function checks the status code provided by the server and generates a specific response depending on the status code. In this implementation, the tool has a specific response for the most common return codes: 200, 302, 400, 401, 403, 404, 414 and 500. If the state code is 200 and the method is “*GET*” or “*POST*” the server replies with a simple HTML web page. The implementation of this function is as follows:

```
def http(pcapfile, dstIp, dstPort, srcIp, url, method, protocol, useragent, referer, param, retCode):
    try:
        srcPort = int(random.getrandbits(16))
        while srcPort < 1024:
            srcPort = int(random.getrandbits(16))

        _query = "%s %s %s\n" % (method, url, protocol)
        _useragent = "User-Agent: %s\n" % str(useragent)
        _referer = "Referer: %s\n" % str(referer)
        if param == "":
            httpQuery = "%s%s%s\n" % (_query, _useragent, _referer)
        else:
            _param = "%s\n" % str(param)
            httpQuery = "%s%s%s%s\n" % (_query, _useragent, _referer, _param)
```

```
                if retCode == "200":
                        _response = "%s %s OK" % (protocol, retCode)
                elif retCode == "302":
                        _response = "%s %s Found" % (protocol, retCode)
                elif retCode == "400":
                        _response = "%s %s Bad Request" % (protocol, retCode)
                elif retCode == "401":
                        _response = "%s %s Unauthorized" % (protocol, retCode)
                elif retCode == "403":
                        _response = "%s %s Forbidden" % (protocol, retCode)
                elif retCode == "404":
                        _response = "%s %s Not Found" % (protocol, retCode)
                elif retCode == "414":
                        _response = "%s %s Request-URI Too Long" % (protocol, retCode)
                elif retCode == "500":
                        _response = "%s %s Internal Server Error" % (protocol, retCode)
                else:
                        _response = "%s %s Other" % (protocol, retCode)

                if retCode == "200" and (method == "GET" or method == "POST"):
                        httpResponse = "%s\n\n<html><head></head><body></body></html>\n\n" % _response
                elif retCode == "302" or retCode == "200":
                        httpResponse = "%s\n\n" % _response
                else:
                        httpResponse = "%s\nConnection: close\n\n" % _response

                (seqN, ackN) = handshake(pcapfile, dstIp, dstPort, srcIp, srcPort)
                (seqN, ackN) = queryResponse(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN, httpQuery, httpResponse)
                finishconnection(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN)
        except:
                sys.stderr.write("HTTP traffic:\nhttpQuery:\n%s\nhttpResponse:\n%s\n-------" % (httpQuery, httpResponse))
```

The "*httpfast*" function has two parameters: *httpQuery* and *httpResponse*. This function does not parse the parameters to prepare the information as the protocol describes, but it is only implemented for specific cases in which the programmer needs to use specific custom queries and responses to recreate the network traffic with any other functionality:

```
def httpfast(pcapfile, dstIp, dstPort, srcIp, httpQuery, httpResponse):
        try:
                srcPort = int(random.getrandbits(16))
                while srcPort < 1024:
                        srcPort = int(random.getrandbits(16))

                (seqN, ackN) = handshake(pcapfile, dstIp, dstPort, srcIp, srcPort)
```

```
                (seqN, ackN) = queryResponse(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN, httpQuery, httpResponse)
                finishconnection(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN)
        except:
                sys.stderr.write("HTTP traffic:\nhttpQuery:\n%s\nhttpResponse:\n%s\n-------" % (httpQuery, httpResponse))
```

4.3. Parse Logs

The process of parsing the logs depends on the format of the input logs provided. The tool has support for four web server logs and five different types of logs. We will use the Apache template as an example, although the parse code for the other logs can be found at the end of this paper.

The function requires as parameters the name of the PCAP file, the name of the log file and the IP address and port of the web server. This information is usually provided to the tool when it is called through the prompt. The following code shows an example:

```
logFile=sys.argv[1]
fileName=sys.argv[2]
template = str(sys.argv[3])
if template == "apache":
        serverIp=sys.argv[4]
        serverPort=sys.argv[5]
        pcapfile = PcapWriter(fileName, append=True)
        for log in open(logFile):
                apache(log, pcapfile, serverIp, serverPort)
        pcapfile.close()
```

First of all, the apache log parser function deletes the newline character and then splits the log line in order to get all the information from the log registry and assign it to the different variables. If the log do not provides some fields that the *http* function needs, the value of these parameters is blank (“”). At the end of the code the *http* function is called.

```
def apache(linelog, pcapfile, serverIp, serverPort):
        try:
                linelog = re.sub(r"[\r\n]+", "", linelog)

                zone = linelog.split("")
                srcIp = zone[0].split(' -')[0]
                query = zone[1].split(' ')
```

```
                method = str(query[0])
                url = str(query[1])
                protocol = str(query[-1])
                useragent = str(zone[5])
                referer = str(zone[3])
                retCode = str(zone[2].split()[0])
                param = ""

                http(pcapfile, serverIp, serverPort, srcIp, url, method, protocol, useragent, referer, param, retCode)
        except:
                sys.stderr.write("Unable to parse %s\n" % linelog)
```

5. Example of use

The tool has been tested in two different scenarios. The first is a prepared demo in a controlled environment while the second scenario is a Honeynet challenge that includes different logs, one of them from an Apache web server.

5.1. Prepared environment

In the first case, the environment includes four computers: the first one is the intruder that launches an audit against the second server, an Apache web server, using the popular *Nikto* audit tool. The third computer is a *centralized log server* that receives the logs from the web server using the *Rsyslog* daemon. The last computer is the *Security Expert* or *Auditor* computer, used to analyze the information from the centralized log:

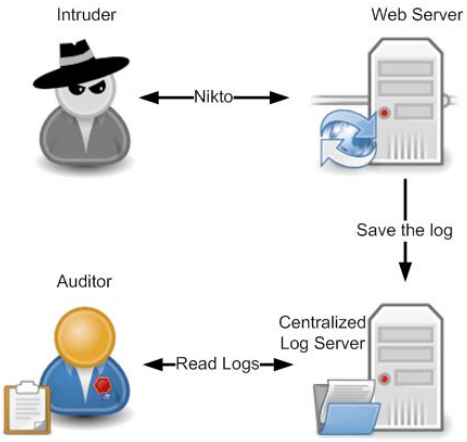

Image 5: diagram of the demo example

5.1.1. Get the logs

The log used was obtained from the *centralized log server* and copied to the *auditor computer* in order to work with the copy and not the original file. The name of the file is "*Diciembre.log*" and it uses the *Apache combined log* format from a Linux Debian 6 server. The log file has nearly 350 queries from the intruder. The image shows that the log file contains HTTP requests potentially made by an intruder using Nikto:

```
root@bt:~/log2pcap# md5sum Diciembre.log
5b70931f1a1fa1b51f27c8369423f74d  Diciembre.log
root@bt:~/log2pcap# head -2 Diciembre.log && tail -2 Diciembre.log
65.96.235.121 - - [11/Dic/2012:21:39:50 +0000] "HEAD / HTTP/1.1" 200 300 "-" "Mozilla/5.00 (Nikto/
2.1.5) (Evasions:None) (Test:Port Check)"
65.96.235.121 - - [11/Dic/2012:21:39:50 +0000] "GET / HTTP/1.1" 200 418 "-" "Mozilla/5.00 (Nikto/2
.1.5) (Evasions:None) (Test:getinfo)"
65.96.235.121 - - [11/Dic/2012:21:43:41 +0000] "GET /scripts/httpodbc.dll HTTP/1.1" 404 120258 "-"
 "Mozilla/5.00 (Nikto/2.1.5) (Evasions:None) (Test:000151)"
65.96.235.121 - - [11/Dic/2012:21:43:42 +0000] "GET /scripts/proxy/w3proxy.dll HTTP/1.1" 404 12025
8 "-" "Mozilla/5.00 (Nikto/2.1.5) (Evasions:None) (Test:000152)"
root@bt:~/log2pcap#
```

Image 6: Apache log from the auditing apache server with Nikto.

The IP of the intruder is 65.96.235.121, a fictional and random IP address created for this demo. It makes several requests to the web server using Nikto v2.1.5 without making use of evasion techniques.

5.1.2. Generate PCAP File

After the log has been obtained, the PCAP file is generated using Log2Pcap. If the program is executed without any options, it shows the supported log formats and the different options available. In this case the template is "*apache*", which requires as options the IP and port of the web server. We have used for this example "192.168.0.1" and "80" respectively. The name of the PCAP file is "*result.pcap*":

```
root@bt:~/log2pcap# ./log2pcap.py
WARNING: No route found for IPv6 destination :: (no default route?)

        Usage: ./log2pcap.py logfile pcapfile template [templateOptions]

        Available Templates:
                - Apache Combine (default) Web Server: apache serverip serverport
                  Example: ./log2pcap.py access.log output.pcap apache 192.168.0.1 80

                - IIS Log Web Server: iis serverport
                  Example: ./log2pcap.py access.log output.pcap iis 80

                - IIS W3C Web Server: iis-w3c serverport
                  Example: ./log2pcap.py access.log output.pcap iis-w3c 80

                - IBM Webseal: webseal serverip serverport
                  Example: ./log2pcap.py access.log output.pcap webseal 192.168.0.1 80

                - Nginx: nginx serverip serverport
                  Example: ./log2pcap.py access.log output.pcap nginx 192.168.0.1 80


root@bt:~/log2pcap# ./log2pcap.py Diciembre.log result.pcap apache 192.168.0.1 80
WARNING: No route found for IPv6 destination :: (no default route?)
root@bt:~/log2pcap# du -h result.pcap
168K    result.pcap
root@bt:~/log2pcap#
```

Image 7: Log2Pcap tool example with apache template

As we can see, the size of the output PCAP file is 168 Kilobyte. The file can then be loaded into programs that support the PCAP format, such as *Wireshark*, *Snort*, *Ethercap* and others (Burns, et al., 2007). In this case we are using *Tcpdump* and *Tcpflow* to show the generated traffic:

```
root@bt:~/log2pcap# tcpdump -nn -r result.pcap -c 9
reading from file result.pcap, link-type RAW (Raw IP)
17:13:34.362319 IP 65.96.235.121.16047 > 192.168.0.1.80: Flags [S], seq 3205127047,
17:13:34.363595 IP 192.168.0.1.80 > 65.96.235.121.16047: Flags [S.], seq 595736834,
17:13:34.364685 IP 65.96.235.121.16047 > 192.168.0.1.80: Flags [.], ack 1, win 8192
17:13:34.366065 IP 65.96.235.121.16047 > 192.168.0.1.80: Flags [.], seq 1:102, ack
17:13:34.367660 IP 192.168.0.1.80 > 65.96.235.121.16047: Flags [.], seq 1:18, ack 1
17:13:34.368959 IP 65.96.235.121.16047 > 192.168.0.1.80: Flags [R], seq 3205127149,
17:13:34.370146 IP 65.96.235.121.7674 > 192.168.0.1.80: Flags [S], seq 1488161307,
17:13:34.371282 IP 192.168.0.1.80 > 65.96.235.121.7674: Flags [S.], seq 441378647,
17:13:34.372353 IP 65.96.235.121.7674 > 192.168.0.1.80: Flags [.], ack 1, win 8192,
root@bt:~/log2pcap# tcpflow -r result.pcap -C -s -e | head -18
HEAD / HTTP/1.1
User-Agent: Mozilla/5.00 (Nikto/2.1.5) (Evasions:None) (Test:Port Check)
Referer: -


HTTP/1.1 200 OK


GET / HTTP/1.1
User-Agent: Mozilla/5.00 (Nikto/2.1.5) (Evasions:None) (Test:getinfo)
Referer: -


HTTP/1.1 200 OK

<html><head></head><body></body></html>


root@bt:~/log2pcap#
```

Image 8: Reading PCAP file with Tcpdump and Tcpflow.

As we can see in the image, the PCAP file shows the network traffic transmitted between the intruder and the web server. In this case, the *http* function has used *resetconnection* instead of *finishconnection* to increase the performance. Next we will read the PCAP file with Snort to look for potential attacks.

5.1.3. Inject the PCAP file in Snort

For this proof of concept, Snort has been configured with the default configuration of a BackTrack 5.2 with Sourcefire VRT rules. The PCAP "*result.pcap*" file will be loaded as input and the attacks detected by Snort will be saved in the file *alert* in the /tmp directory.

```
root@bt:~/log2pcap# du -h result.pcap
168K    result.pcap
root@bt:~/log2pcap# snort -r result.pcap  -c /etc/snort/snort.conf -l /tmp/
```

Image 9: Running Snort with the PCAP file created by Log2Pcap

When executed, Snort shows relevant information about the packets read: number of packets, number of alerts detected, strange requests, etc.:

```
                  Dropped: 0
                Inspected: 0
                  Tracked: 0
===============================================================================
HTTP Inspect - encodings (Note: stream-reassembled packets included):
    POST methods:                   0
    GET methods:                    706
    Headers extracted:              706
    Header Cookies extracted:       0
    Post parameters extracted:      0
    Unicode:                        0
    Double unicode:                 0
    Non-ASCII representable:        0
    Base 36:                        0
    Directory traversals:           0
    Extra slashes ("//"):           18
    Self-referencing paths ("./"):  0
    Total packets processed:        1083
===============================================================================
dcerpc2 Preprocessor Statistics
  Total sessions: 0
===============================================================================
===============================================================================
Snort exiting
root@bt:~/log2pcap# wc -l /tmp/alert
2841 /tmp/alert
root@bt:~/log2pcap# grep '\[\*\*\]' /tmp/alert | sort | uniq | wc -l
38
root@bt:~/log2pcap#
```

Image 10: Snort results

As a result Snort provides a new file called *alert* with 2841 lines and 408 alerts, some of the alerts repeat.

We can see that Snort detected 38 different alerts from one default Nikto auditing. Most of them show that Nikto tried to access default files that usually have sensitive information such as *htpasswd*, *cgi-bin* path, *robots.txt*, path transversal, Trace HTTP method or invalid methods:

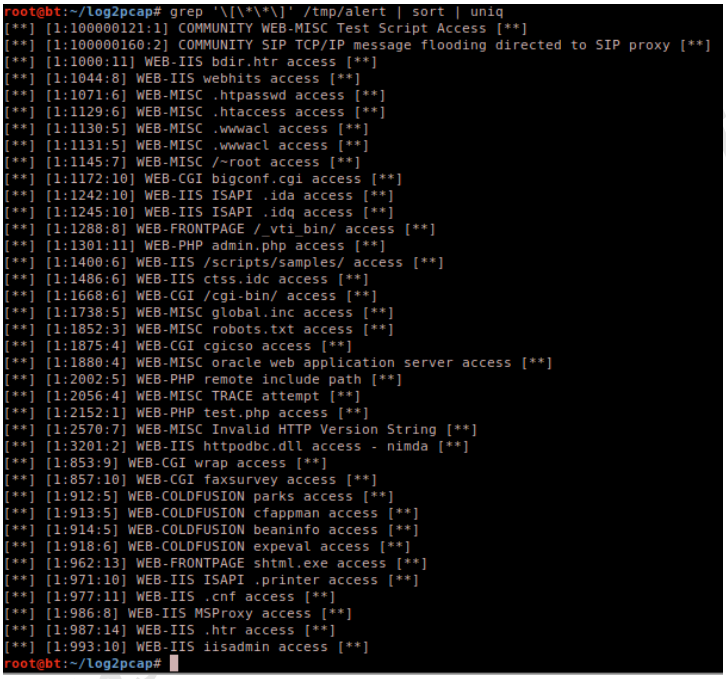

Image 11: Alerts detected by Snort.

As we can see the main current limitation of the program is the loss of context information: the user does not know which log line has generated a specific Snort alert. For this reason it is important not to load more than one day of log in the Log2Pcap tool and use only one log per day.

5.2. Challenge from Honey Net

In September of 2010 the Honey Net project published a forensics challenge with different logs coming from one Linux server (Raffael, Chuvakin, & Tricaud, 2010). One of these logs is from one Apache Web Server that we will feed to Log2Pcap.

In this case, the Apache log is provided in two different files: *access* and *media*. For this reason the files have been joined in only one log file named "*apache.log*". As before, the Log2Pcap has generated the PCAP that has then been read by Snort. This

scenario used the default Snort in Black Hat 5.2: Snort 2.8 with the free Source Fire VRT rules:

```
root@bt:~/log2pcap# ./log2pcap.py apache.log result.pcap apache 192.168.0.1 80
WARNING: No route found for IPv6 destination :: (no default route?)
root@bt:~/log2pcap# du -h result.pcap
408K    result.pcap
root@bt:~/log2pcap# tcpdump -nn -r result.pcap | wc -l
reading from file result.pcap, link-type RAW (Raw IP)
5346
root@bt:~/log2pcap# snort -r result.pcap  -c /etc/snort/snort.conf -l /tmp/
```

Image 12: Log2Pcap are going to inject the result in the Snort.

The Snort has reported only four alerts where only two are distinct:

```
===============================================================================
HTTP Inspect - encodings (Note: stream-reassembled packets included):
    POST methods:                   64
    GET methods:                    1110
    Headers extracted:              1174
    Header Cookies extracted:       0
    Post parameters extracted:      0
    Unicode:                        0
    Double unicode:                 0
    Non-ASCII representable:        0
    Base 36:                        0
    Directory traversals:           0
    Extra slashes ("//"):           0
    Self-referencing paths ("./"):  0
    Total packets processed:        1782
===============================================================================
dcerpc2 Preprocessor Statistics
  Total sessions: 0
===============================================================================
===============================================================================
Snort exiting
root@bt:~/log2pcap# grep '\[\*\*\]' /tmp/alert | wc -l
4
root@bt:~/log2pcap# grep '\[\*\*\]' /tmp/alert | sort | uniq | wc -l
2
root@bt:~/log2pcap# grep '\[\*\*\]' /tmp/alert | sort | uniq
[**] [1:100000160:2] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy
[**] [1:1852:3] WEB-MISC robots.txt access [**]
root@bt:~/log2pcap#
```

Image 13: Snort result from Honey Net challenge.

In this case, Snort does not give the results expected about what happened in the environment. If the Apache log or the PCAP file is analyzed, some requests and user agents that can imply a threat can be found. This information can be confirmed in the solution of the challenge from William Söderberg (Söderberg, 2010).

In order to detect such threats, it is necessary to use other rule sets, in this case the one from Emerging Threats. To use the last version of these rules Snort 2.9 is required. For this reason, a new analyst environment is prepared using a Security Onion (Burks, 2013) environment with Snort 2.9.3 and the Emerging Threats rules.

The Security Onion is one of the best all-in-one environments for intrusion detection systems. Although it is not the purpose of this document to explain how this distribution works, it is necessary to explain the basics steps to read a PCAP file, due to the fact that in Security Onion DAQ is enabled by default and Snort cannot read PCAP files with DAQ.

The first step is to download the Emerging Threats rules and update them with Pulled Pork (JJ, 2013):

```
root@moxilo-virtual-machine:/root/log2pcap# /usr/bin/rule-update
Backing up current downloaded.rules file before it gets overwritten.
Cleaning up downloaded.rules backup files older than 30 days.
Running PulledPork.

    http://code.google.com/p/pulledpork/
     _____ ____
     `----,\    )
      `--==\\  /    PulledPork v0.6.1 the Smoking Pig </////~
       `--==\\/
     .-~~~~-.Y|\\_  Copyright (C) 2009-2011 JJ Cummings
  @_/        /  66\_  cummingsj@gmail.com
    |    \   \   _(")
     \   /-| ||'--'  Rules give me wings!
      \_\  \_\\
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Checking latest MD5 for emerging.rules.tar.gz....
Rules tarball download of emerging.rules.tar.gz....
        They Match
        Done!
Prepping rules from emerging.rules.tar.gz for work....
        Done!
Reading rules...
Generating Stub Rules....
```

Image 14: Updating rules

Then, it is necessary to change the Snort configuration and we are ready to launch Snort with the same options used in this paper:

```
root@moxilo-virtual-machine:/root/log2pcap# cp /etc/nsm/templates/snort/* .
root@moxilo-virtual-machine:/root/log2pcap# cp /etc/nsm/rules/* .
cp: omitting directory `/etc/nsm/rules/backup'
root@moxilo-virtual-machine:/root/log2pcap# grep -v '^#\|^$\|daq' snort.conf >> snort-read.conf
root@moxilo-virtual-machine:/root/log2pcap# grep output snort-read.conf
output unified2: filename snort.unified2, limit 128
root@moxilo-virtual-machine:/root/log2pcap# snort -r result.pcap -c snort-read.conf -l /tmp/
```

Image 15: Configuring Snort and lunch it

When executed, Snort notifies us that three alerts have been detected:

```
===============================================================================
Action Stats:
     Alerts:             3 (  0.051%)
     Logged:             3 (  0.051%)
     Passed:             0 (  0.000%)
Limits:
      Match:             0
      Queue:             0
        Log:             0
      Event:             0
      Alert:             0
Verdicts:
      Allow:          5346 (100.000%)
      Block:             0 (  0.000%)
    Replace:             0 (  0.000%)
  Whitelist:             0 (  0.000%)
  Blacklist:             0 (  0.000%)
     Ignore:             0 (  0.000%)
===============================================================================
```

Image 16: Snort has detected three alerts.

Since the version of Snort is 2.9.3, we need to work with *unified2* files. The easiest way to read this format and parse to traditional Snort alerts is to use *Barnyard2* with the following configuration and options:

```
root@moxilo-virtual-machine:/root/log2pcap# ls -l /tmp/snort.unified2.*
-rw------- 1 root root 624 Jan  4 21:21 /tmp/snort.unified2.1357334516
root@moxilo-virtual-machine:/root/log2pcap# grep -v '^#\|^$' /etc/nsm/templates/barnyard2/barnyard2.conf >> b2aux.conf
root@moxilo-virtual-machine:/root/log2pcap# sed 's/\/etc\/snort\///' b2aux.conf >> barnyard2.conf
root@moxilo-virtual-machine:/root/log2pcap# mkdir /var/log/barnyard2
root@moxilo-virtual-machine:/root/log2pcap# barnyard2 -v -c barnyard2.conf -d /tmp -f snort.unified2
Running in Continuous mode

       --== Initializing Barnyard2 ==--
Initializing Input Plugins!
Initializing Output Plugins!
Parsing config file "barnyard2.conf"
Barnyard2 spooler: Event cache size set to [2048]
Log directory = /var/log/barnyard2

       --== Initialization Complete ==--

  ______   -*> Barnyard2 <*-
 / ,,_  \  Version 2.1.11 (Build 317) TCL
 |o"  )~|  By Ian Firns (SecurixLive): http://www.securixlive.com/
 + '''' +  (C) Copyright 2008-2012 Ian Firns <firnsy@securixlive.com>

Opened spool file '/tmp/snort.unified2.1357334516'
01/03-16:53:20.197836  [**] [1:2013051:2] ET WEB_SERVER pxyscand Suspicious User Agent Inbound [**] [Classification: Att
9.122.56:33931 -> 192.168.0.1:80
01/03-16:53:22.200290  [**] [1:2013051:2] ET WEB_SERVER pxyscand Suspicious User Agent Inbound [**] [Classification: Att
9.122.57:10038 -> 192.168.0.1:80
01/03-16:53:23.338267  [**] [1:2013051:2] ET WEB_SERVER pxyscand Suspicious User Agent Inbound [**] [Classification: Att
9.122.52:6057 -> 192.168.0.1:80
Waiting for new data
```

Image 17: Using Barnyard2 for translates the unified file.

In the result of Snort three alerts about the "*Pxyscand User Agent*" are shown. These alerts provide more information about the incident and in fact it matches with the solution of the challenge:

*"My script reveals all this information. 3 attackers ran a proxy scanner (from same subnet) called pxyscand against the server …" (*Söderberg, 2010).

6. Conclusions

The most important concept of Log2Pcap is the idea that the network traffic can be recreated using the web server logs. Then, the security analysts can use this generated network traffic in all the tools that can read this type of data such as the Intrusion Detection Systems described.

The final purpose of this tool is not to substitute already mature tools, but to give to the security professional a tool that allows new ways to detect potential threats and helps him/her in their daily work with a different point of view.

7. References

Albin, E. (2011, September). *www.hsdl.org.* Retrieved from http://www.hsdl.org/?view&did=691228

Ashley, D. (2008). www.sans.org. Retrieved from http://www.sans.org/reading_room/whitepapers/tools/developing-snort-dynamic-preprocessor_32874

Biondi, P. (2005, May 4). Packet generation and network based attacks with Scapy. Retrieved from www.secdev.org: http://www.secdev.org/conf/scapy_pacsec05.pdf

Burks, D. (2013, January 3). Security Onion Documentation. Retrieved from http://code.google.com/p/security-onion/wiki/Installation

Burns, B., Killion, D., Beauchesne, N., Moret, E., Sobrier, J., Lynn, M., et al. (2007). Security Power Tools. Sebastopol, CA: O'Reilly Media.

Caswell, Beale, & Baker. (2007). Snort IDS and IPS Toolkit. Burlington, MA: Syngress.

Consortium, W. W. (1994). HTTP Request fields. Retrieved from http://www.w3.org/Protocols/HTTP/HTRQ_Headers.html

Debar, H. (2013). www.sans.org. Retrieved from http://www.sans.org/security-resources/idfaq/behavior_based.php

Foundation, P. S. (2012). Python Online Documentation. Retrieved from http://www.python.org/doc/

JJ. (2013). Pulled Pork. Retrieved from http://code.google.com/p/pulledpork/

Lutz, M. (2011). Programming Python. Sebastopol, CA: O'Reilly Media.

Microsoft TechNet. (2003). Microsoft. Retrieved from http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/c93b2856-76c4-4348-9d46-8a60612c3b23.mspx?mfr=true

msbachman. (2010). www.hackthissite.org. Retrieved from http://www.hackthissite.org/articles/read/1082

Nihuo. (2013). Loganalyzer. Retrieved from http://www.loganalyzer.net/log-analyzer/iis-log-analyzer.html

Paul, M. (2011). Official guide to the CSSLP. Boca Raton, Florida: CRC Press.

Raffael, M., Chuvakin, A., & Tricaud, S. (2010). HoneyNet. Retrieved from http://honeynet.org/challenges/2010_5_log_mysteries

Scapy, P. (2012). Scapy Online Documentation. Retrieved from www.secdev.org: http://www.secdev.org/projects/scapy/doc/

SliceHost. (2010). SliceHost. Retrieved from http://articles.slicehost.com/2010/8/27/reading-nginx-web-logs

Söderberg, W. (2010, October 21). Solution to the Honey Net Challenge 5: Log Mysteries. Retrieved from http://honeynet.org/files/william_soderberg_Forensic_Challenge_2010_-_Challenge_5_-_Submission.pdf

Stevens, W. R., & Fall, K. R. (2011). TCP/IP Illustrated, Volume 1, 2nd ed. Arbor, Michigan: Pearson Education.

US CERT. (2011). www.fbiic.gov. Retrieved from https://www.fbiic.gov/public/2011/feb/EWIN-11-035-01A_UPDATE.pdf

Worman, M. (2009). http://computer-forensics.sans.org. Retrieved from http://computer-forensics.sans.org/blog/2009/05/26/perl-fu-regexp-log-file-processing

Appendix: the code of Log2Pcap

The code included in this appendix is the code of Log2Pcap. The tool is available in the following URL: http://code.google.com/p/forensics-log-2-pcap/. Visit the web of the project for future updates of the tool.


```
#!/usr/bin/python
# -*- encoding: utf-8 -*-

from scapy.all import *
import sys

"""
 Joaquín Moreno Garijo @moxilo
 SANS GCIA GOLD Dic 2012 V0.1

"""

############## CORE #################

"""
		Help menu

		program: name of the program
"""
def options(program):
		print "\n\tUsage: " + program + " logfile pcapfile template [templateOptions]\n"
		print "\tAvailable Templates:"
		print "\t\t- Apache Combine (default) Web Server: apache serverip serverport"
		print "\t\t  Example: " + program + " access.log output.pcap apache 192.168.0.1 80\n"
		print "\t\t- IIS Log Web Server: iis serverport"
		print "\t\t  Example: " + program + " access.log output.pcap iis 80\n"
		print "\t\t- IIS W3C Web Server: iis-w3c serverport"
		print "\t\t  Example: " + program + " access.log output.pcap iis-w3c 80\n"
		print "\t\t- IBM Webseal: webseal serverip serverport"
		print "\t\t  Example: " + program + " access.log output.pcap webseal 192.168.0.1 80\n"
		print "\t\t- Nginx: nginx serverip serverport"
		print "\t\t  Example: " + program + " access.log output.pcap nginx 192.168.0.1 80\n"
		print "\n"
		exit(1)

"""
		Create the handshake connection

		pcapfile: file to save the network traffic
		dstIp: destination IP (server IP)
		dstPort: port destination (server port)
		srcIp: source IP (client IP)
		srcPort: source port (client port)

		seqN: sequence number TCP
		ackN: ack number TCP
"""
def handshake(pcapfile, dstIp, dstPort, srcIp, srcPort):
		#Random seq
		seqN = random.getrandbits(32)
		ackN = 0

		#Syn
		pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="S",seq=seqN,ack=ackN))

		#Random ack
		ackN = random.getrandbits(32)

		#Syn+ack, ack
		pcapfile.write(IP(dst=srcIp,src=dstIp)/TCP(dport=int(srcPort),sport=int(dstPort),flags="SA",seq=ackN,ack=seqN+1))
		pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="A",seq=seqN+1,ack=ackN+1))

		return (seqN+1, ackN+1)

"""
		Reset the comunications
```

```
		pcapfile: file to save the network traffic
		dstIp: destination IP (server IP)
		dstPort: port destination (server port)
		srcIp: source IP (client IP)
		srcPort: source port (client port)
		seqN: sequence number TCP
		ackN: ack number TCP
	"""
	def resetconnection(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN):
		pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="R",seq=seqN,ack=ackN))

	"""
		Finish the comunications

		pcapfile: file to save the network traffic
		dstIp: destination IP (server IP)
		dstPort: port destination (server port)
		srcIp: source IP (client IP)
		srcPort: source port (client port)
		seqN: sequence number TCP
		ackN: ack number TCP
	"""
	def finishconnection(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN):
		#Client FA
		pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="FA",seq=seqN,ack=ackN))
		seqN = seqN + 1
		pcapfile.write(IP(dst=srcIp,src=dstIp)/TCP(dport=int(srcPort),sport=int(dstPort),flags="A",seq=ackN,ack=seqN))
		#Server FA
		pcapfile.write(IP(dst=srcIp,src=dstIp)/TCP(dport=int(srcPort),sport=int(dstPort),flags="FA",seq=ackN,ack=seqN))
		ackN = ackN + 1
		pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="A",seq=seqN,ack=ackN))

	"""
		Query and response

		pcapfile: file to save the network traffic
		dstIp: destination IP (server IP)
		dstPort: port destination (server port)
		srcIp: source IP (client IP)
		srcPort: source port (client port)
		seqN: sequence number TCP
		ackN: ack number TCP
		query: body text for the query
		response: body text for the response
	"""
	def queryResponse(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN, query, response):
		#Query
		pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="A",seq=seqN,ack=ackN)/query)

		#Seq incremental
		if len(query) == 0:
			seqN = seqN + 1
		else:
			seqN = seqN + len(query)
		#Response
		pcapfile.write(IP(dst=srcIp,src=dstIp)/TCP(dport=int(srcPort),sport=int(dstPort),flags="A",seq=ackN,ack=seqN)/response)

		#Acq incremental
		if len(response) == 0:
			ackN = ackN + 1
		else:
			ackN = ackN + len(response)

		#Return seqN, ackN
		return (seqN, ackN)

	"""
		SendData

		pcapfile: file to save the network traffic
		dstIp: destination IP (server IP)
		dstPort: port destination (server port)
		srcIp: source IP (client IP)
		srcPort: source port (client port)
		seqN: sequence number TCP
		ackN: ack number TCP
		data: body text for the query
	"""
	def sendData(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN, data):
```

```
		#Query
		pcapfile.write(IP(dst=dstIp,src=srcIp)/TCP(dport=int(dstPort),sport=int(srcPort),flags="A",seq=seqN,ack=ackN)/data)

		#Seq incremental
		if len(query) == 0:
			seqN = seqN + 1
		else:
			seqN = seqN + len(data)

		#Return seqN, ackN
		return (seqN, ackN)

############## PROTOCOLS #################

"""
		HTTP

		pcapfile: file to save the network traffic
		dstIp: destination IP (server IP)
		dstPort: port destination (server port)
		srcIp: source IP (client IP)
		url: url request
		method: HTTP method
		protocol: HTTP protocol
		useragent: usert agent of the client
		referer: referer query
		param: Post Parameters
		retCode: return code
"""
def http(pcapfile, dstIp, dstPort, srcIp, url, method, protocol, useragent, referer, param, retCode):
		try:
			#Random source port
			srcPort = int(random.getrandbits(16))
			while srcPort < 1024:
				srcPort = int(random.getrandbits(16))

			#Client query
			_query = "%s %s %s\n" % (method, url, protocol)
			_useragent = "User-Agent: %s\n" % str(useragent)
			_referer = "Referer: %s\n" % str(referer)
			if param == "":
				httpQuery = "%s%s%s\n" % (_query, _useragent, _referer)
			else:
				_param = "%s\n" % str(param)
				httpQuery = "%s%s%s%s\n" % (_query, _useragent, _referer, _param)

			#Server Response
			if retCode == "200":
				_response = "%s %s OK" % (protocol, retCode)
			elif retCode == "302":
				_response = "%s %s Found" % (protocol, retCode)
			elif retCode == "400":
				_response = "%s %s Bad Request" % (protocol, retCode)
			elif retCode == "401":
				_response = "%s %s Unauthorized" % (protocol, retCode)
			elif retCode == "403":
				_response = "%s %s Forbidden" % (protocol, retCode)
			elif retCode == "404":
				_response = "%s %s Not Found" % (protocol, retCode)
			elif retCode == "414":
				_response = "%s %s Request-URI Too Long" % (protocol, retCode)
			elif retCode == "500":
				_response = "%s %s Internal Server Error" % (protocol, retCode)
			else:
				_response = "%s %s Other" % (protocol, retCode)

			if retCode == "200" and (method == "GET" or method == "POST"):
				httpResponse = "%s\n\n<html><head></head><body></body></html>\n\n" % _response
			elif retCode == "302" or retCode == "200":
				httpResponse = "%s\n\n" % _response
			else:
				httpResponse = "%s\nConnection: close\n\n" % _response

			#Debug
			#print "httpQuery:\n%s\nhttpResponse:\n%s\n-------" % (httpQuery, httpResponse)

			#Initialice connection
			(seqN, ackN) = handshake(pcapfile, dstIp, dstPort, srcIp, srcPort)

			#Query and response HTTP
```

```
                        (seqN, ackN) = queryResponse(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN, httpQuery, httpResponse)

                        #End connection
                        finishconnection(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN)
                except:
                        sys.stderr.write("Unable to create HTTP traffic:\nhttpQuery:\n%s\nhttpResponse:\n%s\n-------" % (httpQuery, httpResponse))

"""
                HTTP fast (Not parse, query and response string)

                pcapfile: file to save the network traffic
                dstIp: destination IP (server IP)
                dstPort: port destination (server port)
                srcIp: source IP (client IP)
                httpQuery: http query to server
                httpResponse: http response from server
"""
def httpfast(pcapfile, dstIp, dstPort, srcIp, httpQuery, httpResponse):
                try:
                        #Debug
                        #print "httpQuery:\n%s\nhttpResponse:\n%s\n-------" % (httpQuery, httpResponse)

                        #Random source port
                        srcPort = int(random.getrandbits(16))
                        while srcPort < 1024:
                                srcPort = int(random.getrandbits(16))

                        #Initialice connection
                        (seqN, ackN) = handshake(pcapfile, dstIp, dstPort, srcIp, srcPort)

                        #Query and response HTTP
                        (seqN, ackN) = queryResponse(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN, httpQuery, httpResponse)

                        #End connection
                        finishconnection(pcapfile, dstIp, dstPort, srcIp, srcPort, seqN, ackN)
                except:
                        sys.stderr.write("Unable to create HTTP traffic:\nhttpQuery:\n%s\nhttpResponse:\n%s\n-------" % (httpQuery, httpResponse))


############## TEMPLATES #################


"""
                apache combined (default)

                Ex:             10.0.1.2 - - [24/Apr/2010:14:33:22 -0700] "GET /feed/ HTTP/1.1" 200 16605 "-" "Apple-PubSub/65.12.1"
SxPq9AoAAQ4AAEHGCtEAAAAD 967817
                                172.16.2.128 - - [03/Sep/2012:23:18:51 +0200] "GET /F2UMWNgN.orig HTTP/1.1" 404 529 "-" "Mozilla/5.00 (Nikto/2.1.5)
(Evasions:None) (Test:map_codes)"

                linelog: line of one log
                pcapfile: file to save the pcap file
                serverIp: server IP (destination IP)
                serverPort: server port (destination port)
"""
def apache(linelog, pcapfile, serverIp, serverPort):
                try:
                        #Delete "\n"
                        linelog = re.sub(r"[\r\n]+", "", linelog)

                        zone = linelog.split("")
                        srcIp = zone[0].split(' -')[0]
                        query = zone[1].split(' ')
                        method = str(query[0])
                        url = str(query[1])
                        protocol = str(query[-1])
                        useragent = str(zone[5])
                        referer = str(zone[3])
                        retCode = str(zone[2].split()[0])

                        #Not saved by default in the log
                        param = ""

                        #Debug
                        #print "dstIp: %s - dstPort: %s - srcIp: %s - srcPort: %s" % (dstIp, dstPort, srcIp, srcPort)
                        #print "Query:\n%s %s %s\n%s\n%s\n%s\nResponse:\n%s" % (method, url, protocol, useragent, referer, param, retCode)

                        #Add to pcap
                        http(pcapfile, serverIp, serverPort, srcIp, url, method, protocol, useragent, referer, param, retCode)
                except:
```

```
                        sys.stderr.write("Unable to parse %s\n" % linelog)

"""
        IIS W3C Extended
        Info:       http://www.loganalyzer.net/log-analyzer/w3c-extended.html
        Format: date time IPsrc - IPdst Method query - codereturn X X X HTTPVersion UserAgent Var Referer
        Ex:         1998-11-19 22:48:39 206.175.82.5 - 208.201.133.173 GET /global/images/navlogboards.gif - 200 540 324 157 HTTP/1.0
Mozilla/4.0+(compatible;+MSIE+4.01;+Windows+95) USERID=CustomerA;+IMPID=01234 http://www.loganalyzer.net
                    2002-05-24 20:18:01 172.224.24.114 - 206.73.118.24 80 GET /Default.htm - 200 7930 248 31 HTTP/1.0
Mozilla/4.0+(compatible;+MSIE+5.01;+Windows+2000+Server) http://64.224.24.114/

        linelog: line of one log
        pcapfile: file to save the pcap file
        serverPort: server port (destination port)
"""
def iisw3c(linelog, pcapfile, serverPort):
        try:
                #Delete "\n"
                linelog = re.sub(r"[\r\n]+", "", linelog)

                zone = linelog.split(' ')
                srcIp = str(zone[2])
                dstIp = str(zone[4])
                referer = str(zone[-1])
                #Some logs have sourcePort... others no! If it has:
                if zone[5].isdigit():
                        serverPort = zone[5]
                        method = str(zone[6])
                        url = str(zone[7])
                        retCode = zone[9]
                        protocol = str(zone[13])
                        useragent = str(zone[14])
                else:
                        method = str(zone[5])
                        url = str(zone[6])
                        retCode = zone[8]
                        protocol = str(zone[12])
                        useragent = str(zone[13])

                #Not saved by default in the log
                param = ""

                #Debug
                #print "dstIp: %s - dstPort: %s - srcIp: %s" % (dstIp, serverPort, srcIp)
                #print "Query:\n%s %s %s\n%s\n%s\n%s\nResponse:\n%s\n--------" % (method, url, protocol, useragent, referer, param,
retCode)

                #Add to pcap
                http(pcapfile, dstIp, serverPort, srcIp, url, method, protocol, useragent, referer, param, retCode)
        except:
                sys.stderr.write("Unable to parse %s\n" % linelog)

"""
        IIS Log File Format
        Info:       http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/c93b2856-76c4-4348-9d46-
8a60612c3b23.mspx?mfr=true
                    http://www.loganalyzer.net/log-analyzer/iis-log-file-format.html
        Format: Ipsrc, -, date, time, -, -, ipdst, -, -, -, returncode, -, methode, query, -,
        Ex:         192.168.114.201, -, 03/20/01, 7:55:20, W3SVC2, SALES1, 172.21.13.45, 4502, 163, 3223, 200, 0, GET, /DeptLogo.gif, -,

        linelog: line of one log
        pcapfile: file to save the pcap file
        serverPort: server port (destination port)
"""
def iis(linelog, pcapfile, serverPort):
        try:
                #Delete "\n"
                linelog = re.sub(r"[\r\n]+", "", linelog)

                zone = linelog.split(', ')
                srcIp = str(zone[0])
                dstIp = str(zone[6])
                method = str(zone[12])
                url = str(zone[13])
                retCode = str(zone[10])
                param = str(zone[14])

                #Not saved by default in the log
                useragent=""
                referer=""
```

```
                        protocol="HTTP/1.1"

                        #Debug
                        #print "dstIp: %s - dstPort: %s - srcIp: %s" % (dstIp, serverPort, srcIp)
                        #print "Query:\n%s %s %s\n%s\n%s\n%s\nResponse:\n%s\n--------" % (method, url, protocol, useragent, referer, param,
retCode)

                        #Add to pcap
                        http(pcapfile, dstIp, serverPort, srcIp, url, method, protocol, useragent, referer, param, retCode)
                except:
                        sys.stderr.write("Unable to parse %s\n" % linelog)

"""
                IBM Webseal default log

                Ex:             XX.XX.XX.XX - Unauth [01/Oct/2011:10:21:17 +0700] "GET / HTTP/1.0" 200 123
                                XX.XX.XX.XX - Unauth [01/Oct/2011:10:21:19 +0700] "GET /index.php HTTP/1.1" 200 432

                log: line of one log
                pcapfile: file to save the pcap file
                serverIp: server IP (destination IP)
                serverPort: server port (destination port)
"""
def webseal(linelog, pcapfile, serverIp, serverPort):
                try:
                        #Delete "\n"
                        linelog = re.sub(r"[\r\n]+", "", linelog)

                        zone = log.split(' ')
                        srcIp = zone[0]
                        query = (log.split("")[1]).split(' ')
                        method = str(query[0])
                        url = str(query[1])
                        protocol = str(query[-1])
                        retCode = zone[len(zone)-2]

                        #Not saved by default in the log
                        useragent=""
                        referer=""
                        param = ""

                        #Debug
                        #print "dstIp: %s - dstPort: %s - srcIp: %s" % (dstIp, serverPort, srcIp)
                        #print "Query:\n%s %s %s\n%s\n%s\n%s\nResponse:\n%s\n--------" % (method, url, protocol, useragent, referer, param,
retCode)

                        #Add to pcap
                        http(pcapfile, dstIp, serverPort, srcIp, url, method, protocol, useragent, referer, param, retCode)
                except:
                        sys.stderr.write("Unable to parse %s\n" % linelog)


"""
                Nginx File Format
                Info:           http://articles.slicehost.com/2010/8/27/reading-nginx-web-logs

                Ex:             80.154.42.54 - - [23/Aug/2010:15:25:35 +0000] "GET /phpmy-admin/scripts/setup.php HTTP/1.1" 404 347 "-" "ZmEu"
                                123.65.150.10 - - [23/Aug/2010:03:50:59 +0000] "POST /wordpress3/wp-admin/admin-ajax.php HTTP/1.1" 200 2
"http://www.example.com/wordpress3/wp-admin/post-new.php" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_4; en-US) AppleWebKit/534.3
(KHTML, like Gecko) Chrome/6.0.472.25 Safari/534.3"

                linelog: line of one log
                pcapfile: file to save the pcap file
                serverIp: server IP (destination IP)
                serverPort: server port (destination port)
"""
def nginx(linelog, pcapfile, serverIp, serverPort):
                try:
                        #Delete "\n"
                        linelog = re.sub(r"[\r\n]+", "", linelog)

                        zone = linelog.split(', ')
                        srcIp = zone[0].split(' ')[0]
                        zone = linelog.split("")
                        query=zone[1].split(' ')
                        method = str(query[0])
                        url = str(query[1])
                        protocol = str(query[-1])
                        useragent=str(zone[5])
                        referer=str(zone[3])
```

```
                        retCode = zone[2].split(' ')[1]

                        #Not saved by default in the log
                        param = ""

                        #Debug
                        #print "dstIp: %s - dstPort: %s - srcIp: %s" % (dstIp, serverPort, srcIp)
                        #print "Query:\n%s %s %s\n%s\n%s\n%s\nResponse:\n%s\n--------" % (method, url, protocol, useragent, referer, param,
retCode)

                        #Add to pcap
                        http(pcapfile, dstIp, serverPort, srcIp, url, method, protocol, useragent, referer, param, retCode)
                except:
                        sys.stderr.write("Unable to parse %s\n" % linelog)


############## MAIN #################

if len(sys.argv) < 4:
                options(str(sys.argv[0]))

#logfile name
logFile=sys.argv[1]

#pcapfile name
fileName=sys.argv[2]

#Select the template of the log
template = str(sys.argv[3])
if template == "apache":
                if len(sys.argv) != 6:
                                options(str(sys.argv[0]))
                serverIp=sys.argv[4]
                serverPort=sys.argv[5]
                pcapfile = PcapWriter(fileName, append=True)
                for log in open(logFile):
                                apache(log, pcapfile, serverIp, serverPort)
                pcapfile.close()
elif template == "iis":
                if len(sys.argv) != 5:
              options(str(sys.argv[0]))
                serverPort=sys.argv[4]
                pcapfile = PcapWriter(fileName, append=True)
                for log in open(logFile):
                                iis(log, pcapfile, serverPort)
                pcapfile.close()
elif template == "iis-w3c":
                if len(sys.argv) != 5:
              options(str(sys.argv[0]))
                serverPort=sys.argv[4]
                pcapfile = PcapWriter(fileName, append=True)
                for log in open(logFile):
                                iisw3c(log, pcapfile, serverPort)
                pcapfile.close()
elif template == "webseal":
                if len(sys.argv) != 6:
                                options(str(sys.argv[0]))
                serverIp=sys.argv[4]
                serverPort=sys.argv[5]
                pcapfile = PcapWriter(fileName, append=True)
                for log in open(logFile):
                                webseal(log, pcapfile, serverIp, serverPort)
                pcapfile.close()
elif template == "nginx":
                if len(sys.argv) != 6:
                                options(str(sys.argv[0]))
                serverIp=sys.argv[4]
                serverPort=sys.argv[5]
                pcapfile = PcapWriter(fileName, append=True)
                for log in open(logFile):
                                nginx(log, pcapfile, serverIp, serverPort)
                pcapfile.close()
else:
                print "\n\tThe template " + template + " doesn't exist!!!\n"
                options(str(sys.argv[0]))
```