27.1.5 Lab - Convert Data into a Universal Format

This lab has been updated for use on NETLAB+.
www.netdevgroup.com

Objectives

Part 1: Normalize Timestamps in a Log File

Part 2: Normalize Timestamps in an Apache Log File

Part 3: Log File Preparation in Security Onion Virtual Machine

Background / Scenario

This lab will prepare you to learn where log files are located and how to manipulate and view log files. **Log entries** are generated by network devices, operating systems, applications, and various types of programmable devices. A file containing a time-sequenced stream of log entries is called a **log file**.

By nature, log files record events that are relevant to the source. The syntax and format of data within log messages are often defined by the application developer.

Therefore, the terminology used in the log entries often varies from source to source. For example, depending on the source, the terms login, logon, authentication event, and user connection, may all appear in log entries to describe a successful user authentication to a server.

It is often desirable to have a consistent and uniform terminology in logs generated by different sources. This is especially true when all log files are being collected by a centralized point.

The term *normalization* refers to the process of converting parts of a message, in this case a log entry, to a common format.

In this lab, you will use command line tools to manually normalize log entries. In Part 2, the timestamp field will be normalized. In Part 3, the IPv6 field will be normalized.

Note: While numerous plugins exist to perform log normalization, it is important to understand the basics behind the normalization process.

Instructions

Part 1: Normalize Timestamps in a Log File

Timestamps are used in log entries to specify when the recorded event took place. While it is best practice to record timestamps in *UTC*, the format of the timestamp varies from log source to log source. There are two common timestamp formats, known as *Unix Epoch* and *Human Readable*.

Unix Epoch timestamps record time by measuring the number of seconds that have passed since *January 1, 1970*.

Human Readable timestamps record time by representing separate values for year, month, day, hour, minute, and second.

The *Human Readable Wed, 28 Jun 2017 13:27:19 GMT* timestamp is the same as *1498656439* in Unix Epoch.

From a programmability standpoint, it is much easier to work with *Epoch* as it allows for easier addition and subtraction operations. From an analysis perspective; however, *Human Readable* timestamps are much easier to interpret.

Converting Epoch to Human Readable Timestamps with AWK

AWK is a programming language designed to manipulate text files. It is very powerful and especially useful when handling text files where the lines contain multiple fields, separated by a delimiter character. Log files contain one entry per line and are formatted as delimiter-separated fields, making *AWK* a great tool for normalizing.

Consider the *applicationX_in_epoch.log* file below. The source of the log file is not relevant.

```
2|Z|1219071600|AF|0
3|N|1219158000|AF|89
4|N|1220799600|AS|12
1|Z|1220886000|AS|67
5|N|1220972400|EU|23
6|R|1221058800|OC|89
```

The log file above was generated by what we will call application X. The relevant aspects of the file are:

- The columns are separated, or delimited, by the **|** character. Therefore, the data has five columns.
- The third column contains timestamps in *Unix Epoch*.
- The file has an extra line at the end. This will be important later in the lab.

Assume that a log analyst needs to convert the timestamps to a human-readable format. Follow the steps below to use *AWK* to easily perform the manual conversion:

a. Launch the **Workstation** VM and log in with username **analyst** and password **cyberops**.

b. Launch a **terminal** window and use the **cd** command to change to the **/home/analyst/lab.support.files/** directory. A copy of the file shown above is stored there.

```
[analyst@secOps ~]$ cd /home/analyst/lab.support.files/
[analyst@secOps lab.support.files]$ ls -l
total 580
-rw-r--r-- 1 analyst analyst    649 Jun 28 18:34 apache_in_epoch.log
-rw-r--r-- 1 analyst analyst    126 Jun 28 11:13 applicationX_in_epoch.log
drwxr-xr-x 4 analyst analyst   4096 Aug  7 15:29 attack_scripts
-rw-r--r-- 1 analyst analyst    102 Jul 20 09:37 confidential.txt
<output omitted>
[analyst@secOps lab.support.files]$
```

c. Issue the following AWK command to convert and print the result on the terminal:

Note: Up arrow can be used to edit the typing errors in the previous command entry.

```
[analyst@secOps lab.support.files]$ awk 'BEGIN {FS=OFS="|"}
{$3=strftime("%c",$3)} {print}' applicationX_in_epoch.log
2|Z|Mon 18 Aug 2008 11:00:00 AM EDT|AF|0
3|N|Tue 19 Aug 2008 11:00:00 AM EDT|AF|89
4|N|Sun 07 Sep 2008 11:00:00 AM EDT|AS|12
1|Z|Mon 08 Sep 2008 11:00:00 AM EDT|AS|67
5|N|Tue 09 Sep 2008 11:00:00 AM EDT|EU|23
6|R|Wed 10 Sep 2008 11:00:00 AM EDT|OC|89
||Wed 31 Dec 1969 07:00:00 PM EST
```

```
[analyst@secOps lab.support.files]$
```

The command above is an AWK script. It may seem complicated. The main structure of the AWK script above is as follows:

- *awk* – This invokes the *AWK* interpreter.
- *'BEGIN* – This defines the beginning of the script.
- *{}* – This defines actions to be taken in each line of the input text file. An AWK script can have several actions.
- *FS = OFS = "|"* **–** This defines the field separator (i.e., delimiter) as the bar (|) symbol. Different text files may use different delimiting characters to separate fields. This operator allows the user to define what character is used as the field separator in the current text file.
- *$3* **–** This refers to the value in the third column of the current line. In the *applicationX_in_epoch.log*, the third column contains the timestamp in epoch to be converted.
- *strftime* **-** This is an *AWK* internal function designed to work with time. The %c and $3 in between parenthesis are the parameters passed to *strftime*.
- *applicationX_in_epoch.log* **–** This is the input text file to be loaded and used. Because you are already in the *lab.support.files* directory, you do not need to add path information, */home/analyst/lab.support.files/applicationX_in_epoch.log*.

The first script action that defined in the first set of curly brackets is to define the field separator character as the "|". Then, in the second set of curly brackets, it rewrites the third column of each line with the result of the execution of the *strftime()* function. *strftime()* is an internal AWK function created to handle time conversion. Notice that the script tells the function to use the contents of the third column of each line before the change (*$3*) and to format the output (*%c*).

Were the *Unix Epoch* timestamps converted to *Human Readable* format? Were the other fields modified? Explain.

Compare the contents of the file and the printed output. Why is there the line, **||Wed 31 Dec 1969 07:00:00 PM EST**?

---

d. Use **nano** (or your favorite text editor) to remove the extra empty line at the end of the file and run the **AWK** script again by using the up-arrow to find it in the command history buffer.

```
[analyst@secOps lab.support.files]$ nano applicationX_in_epoch.log
```

Is the output correct now? Explain.

---

e. While printing the result on the screen is useful for troubleshooting the script, analysts will likely need to save the output in a text file. Redirect the output of the script above to a file named **applicationX_in_human.log** to save it to a file:

```
[analyst@secOps lab.support.files]$ awk 'BEGIN {FS=OFS="|"}
{$3=strftime("%c",$3)} {print}' applicationX_in_epoch.log >
applicationX_in_human.log
[analyst@secOps lab.support.files]$
```

What was printed by the command above? Is this expected?

_____

f. Use **cat** to view the **applicationX_in_human.log**. Notice that the extra line is now removed and the timestamps for the log entries have been converted to human readable format.

```
[analyst@secOps lab.support.files]$ cat applicationX_in_human.log
2|Z|Mon 18 Aug 2008 11:00:00 AM EDT|AF|0
3|N|Tue 19 Aug 2008 11:00:00 AM EDT|AF|89
4|N|Sun 07 Sep 2008 11:00:00 AM EDT|AS|12
1|Z|Mon 08 Sep 2008 11:00:00 AM EDT|AS|67
5|N|Tue 09 Sep 2008 11:00:00 AM EDT|EU|23
6|R|Wed 10 Sep 2008 11:00:00 AM EDT|OC|89
[analyst@secOps lab.support.files]$
```

Part 2: Normalize Timestamps in an Apache Log File

Similar to what was done with the *applicationX_in_epoch.log* file, Apache web server log files can also be normalized. Follow the steps below to convert Unix Epoch to Human Readable timestamps. Consider the following *Apache* log file, *apache_in_epoch.log*:

```
[analyst@secOps lab.support.files]$ cat apache_in_epoch.log
198.51.100.213 - - [1219071600] "GET
/twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables
HTTP/1.1" 401 12846
198.51.100.213 - - [1219158000] "GET
/twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523
198.51.100.213 - - [1220799600] "GET /mailman/listinfo/hsdivision HTTP/1.1" 200 6291
198.51.100.213 - - [1220886000] "GET /twiki/bin/view/TWiki/WikiSyntax HTTP/1.1" 200
7352
198.51.100.213 - - [1220972400] "GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1" 200
5253
198.51.100.213 - - [1221058800] "GET
/twiki/bin/oops/TWiki/AppendixFileSystem?template=oopsmore&m1=1.12&m2=1.12 HTTP/1.1"
200 11382
```

The *Apache* Log file above contains six entries which record events related to the *Apache* web server. Each entry has seven fields. The fields are delimited by a space:

- The first column contains the *IPv4* address, *198.51.100.213*, of the web client placing the request.
- The second and third columns are not used and a "**-**" character is used to represent no value.
- The fourth column contains the timestamp in *Unix Epoch* time, for example *[1219071600]*.
- The fifth column contains text with details about the event, including URLs and web request parameters. All six entries are *HTTP GET* messages. Because these messages include spaces, the entire field is enclosed with quotes.
- The sixth column contains the *HTTP* status code, for example *401*.
- The seventh column contains the size of the response to the client (in bytes), for example *12846*.

As in Part 1, a script will be created to convert the timestamp from *Epoch to Human Readable*.

a. First, answer the questions below. They are crucial for the construction of the script.

In the context of timestamp conversion, what character would work as a good delimiter character for the Apache log file above?

How many columns does the Apache log file above contain?

In the Apache log file above, what column contains the Unix Epoch Timestamp?

b. In the **Workstation VM** terminal, a copy of the Apache log file, apache_in_epoch.log, is stored in the /home/analyst/lab.support.files.

c. Use an **awk** script to convert the timestamp field to a human readable format. Notice that the command contains the same script used previously, but with a few adjustments for the delimiter, timestamp field, and file name.

```
[analyst@secOps lab.support.files]$ awk 'BEGIN {FS=OFS=" "}
{$4=strftime("%c",$4)} {print}' apache_in_epoch.log
```

Was the script able to properly convert the timestamps? Describe the output.

d. Before moving forward, think about the output of the script.

Can you guess what caused the incorrect output? Is the script incorrect? What are the relevant differences between the **applicationX_in_epoch.log** and **apache_in_epoch.log**?

e. To fix the problem, the square brackets must be removed from the timestamp field before the conversion takes place. Adjust the script by adding two actions before the conversion, as shown below:

```
[analyst@secOps lab.support.files]$ awk 'BEGIN {FS=OFS=" "}
{gsub(/\[|\]/,"",$4)}{print}{$4=strftime("%c",$4)}{print}'
apache_in_epoch.log
```

Notice after specifying space as the delimiter with **{FS=OFS=" "}**, there is a regular expression action to match and replace the square brackets with an empty string, effectively removing the square brackets that appear in the timestamp field. The second action prints the updated line so the conversion action can be performed.

- *gsub()* **–** This is an internal AWK function used to locate and substitute strings. In the script above, *gsub()* received three comma-separated parameters, described below.
- */\[|\]/* **–** This is a regular expression passed to *gsub()* as the first parameter. The regular expression should be read as '*find "[" OR "]"*'. Below is the breakdown of the expression:
    - The first and last "/" character marks the beginning and end of the search block. Anything between the first "/" and the second "/" are related to the search. The "\" character is used to escape the following "[". Escaping is necessary because "[" can also be used by an operator in regular expressions. By escaping the "[" with a leading "\", we tell the interpreter that the "]" is part of the content and not an operator. The "|" character is the

OR operator. Notice that the "|" is not escaped and will therefore, be seen as an operator. Lastly, the regular expression escapes the closing square bracket with "\]", as done before.

- *""* – This represents no characters, or an empty string. This parameter tells *gsub()* what to replace the "[" and "]" with, when found. By replacing the "[" and "]" with "", *gsub()* effectively removes the "[" and "]" characters.
- *$4* – This tells *gsub()* to work only on the fourth column of the current line, the timestamp column.

Note: Regular expression interpretation is a *SECOPS* exam topic. Regular expressions are covered in more detail in another lab in this chapter. However, you may wish to search the Internet for tutorials.

f. In the Workstation VM terminal, execute the adjusted script, as follows:

```
[analyst@secOps lab.support.files]$ awk 'BEGIN {FS=OFS=" "}
{gsub(/\[|\]/,"",$4)}{print}{$4=strftime("%c",$4)}{print}'
apache_in_epoch.log
```

Was the script able to properly convert the timestamps this time? Describe the output.

_____

_____

_____

g. Shut down **CyberOps Workstation** VM if desired.

Part 3: Log File Preparation in Security Onion

Because log file normalization is important, log analysis tools often include log normalization features. Tools that do not include such features often rely on plugins for log normalization and preparation. The goal of these plugins is to allow log analysis tools to normalize and prepare the received log files for tool consumption.

The **Security Onion** appliance relies on a number of tools to provide log analysis services. *ELK, Zeek, Snort* and *SGUIL* are arguably the most used tools.

ELK (Elasticsearch, Logstash, and Kibana) is a solution to achieve the following:

- Normalize, store, and index logs at unlimited volumes and rates.
- Provide a simple and clean search interface and API.
- Provide an infrastructure for alerting, reporting and sharing logs.
- Plugin system for taking actions with logs.
- Exist as a completely free and open-source project.

Zeek (formerly called Bro) is a framework designed to analyze network traffic passively and generate event logs based on it. Upon network traffic analysis, Zeek creates logs describing events such as the following:

- TCP/UDP/ICMP network connections
- DNS activity
- FTP activity
- HTTPS requests and replies
- SSL/TLS handshakes

Snort and SGUIL

Snort is an *IDS* that relies on pre-defined rules to flag potentially harmful traffic. *Snort* looks into all portions of network packets (headers and payload), looking for patterns defined in its rules. When found, Snort takes the action defined in the same rule.

SGUIL provides a graphical interface for Snort logs and alerts, allowing a security analyst to pivot from SGUIL into other tools for more information. For example, if a potentially malicious packet is sent to the organization web server and Snort raised an alert about it, *SGUIL* will list that alert. The analyst can then right-click that alert to search the *ELSA* or Bro databases for a better understanding of the event.

Note: The directory listing maybe different than the sample output shown below.

Step 1: Start Security Onion VM.

Launch the **Security Onion** VM (username: **analyst** / password: **cyberops**).

Step 2: Zeek Logs in Security Onion

a. Open a terminal window in the Security Onion VM. Right-click the Desktop. In the pop-up menu, select **Open Terminal**.

b. **Zeek** logs are stored at **/nsm/bro/logs/**. As usual with Linux systems, log files are rotated based on the date, renamed and stored on the disk. The current log files can be found under the current directory. From the terminal window, change directory using the following command.

```
analyst@SecOnion:~$ cd /nsm/bro/logs/current
analyst@SecOnion:/nsm/logs/current$
```

c. Use the **ls -l** command to see the log files generated by **Zeek**:

Note: Depends on the state of the virtual machine, there may not be any log files yet.

Step 3: Snort Logs

a. Snort logs can be found at **/nsm/sensor_data/**. Change directory as follows.

```
analyst@SecOnion:/nsm/bro/logs/current$ cd /nsm/sensor_data
analyst@SecOnion:/nsm/sensor_data$
```

b. Use the ls -l command to see all the log files generated by Snort.

```
analyst@SecOnion:/nsm/sensor_data$ ls -l
total 12
drwxrwxr-x 7 sguil sguil 4096 Jun 19 18:09 seconion-eth0
drwxrwxr-x 5 sguil sguil 4096 Jun 19 18:09 seconion-eth1
drwxrwxr-x 7 sguil sguil 4096 Jun 19 18:32 seconion-import
```

c. Notice that Security Onion separates files based on the interface. Because the **Security Onion VM** image has two interfaces configured as sensors and a special folder for imported data, three directories are kept. Use the **ls –l seconion-eth0** command to see the files generated by the eth0 interface.

```
analyst@SecOnion:/nsm/sensor_data$ ls -l seconion-eth0
total 28
drwxrwxr-x 2 sguil sguil 4096 Jun 19 18:09 argus
drwxrwxr-x 3 sguil sguil 4096 Jun 19 18:09 dailylogs
drwxrwxr-x 2 sguil sguil 4096 Jun 19 18:09 portscans
drwxrwxr-x 2 sguil sguil 4096 Jun 19 18:09 sancp
drwxr-xr-x 2 sguil sguil 4096 Jun 19 18:24 snort-1
-rw-r--r-- 1 sguil sguil 5594 Jun 19 18:31 snort-1.stats
-rw-r--r-- 1 root  root     0 Jun 19 18:09 snort.stats
```

Step 4: Various Logs

a. While the **/nsm/** directory stores some log files, more specific log files can be found under **/var/log/nsm/**. Change directory and use the **ls** command to see all the log files in the directory.

```
analyst@SecOnion:/nsm/sensor_data$ cd /var/log/nsm/
analyst@SecOnion:/var/log/nsm$ ls
eth0-packets.log              sid_changes.log
netsniff-sync.log             so-elastic-configure-kibana-dashboards.log
ossec_agent.log               so-elasticsearch-pipelines.log
pulledpork.log                so-sensor-backup-config.log
seconion-eth0                 so-server-backup-config.log
seconion-import               sosetup.log
securityonion                 so-zeek-cron.log
sensor-clean.log              squert-ip2c-5min.log
sensor-clean.log.1.gz         squert-ip2c.log
sensor-clean.log.2.gz         squert_update.log
sensor-newday-argus.log       watchdog.log
sensor-newday-http-agent.log  watchdog.log.1.gz
sensor-newday-pcap.log        watchdog.log.2.gz
sguil-db-purge.log
```

Notice that the directory shown above also contains logs used by secondary tools such as **OSSEC** and **Squert**.

b. ELK logs can be found in the **/var/log** directory. Change directory and use the **ls** command to list the files and directories.

```
analyst@SecOnion:/var/log/nsm$ cd ..
analyst@SecOnion:/var/log$ ls
alternatives.log    debug            kern.log.1     samba
alternatives.log.1  debug.1          kern.log.2.gz  sguild
apache2             debug.2.gz       kibana         so-boot.log
apt                 dmesg            lastlog        syslog
auth.log            domain_stats     lightdm        syslog.1
auth.log.1          dpkg.log         logstash       syslog.2.gz
auth.log.2.gz       dpkg.log.1       lpr.log        syslog.3.gz
boot                elastalert       mail.err       syslog.4.gz
boot.log            elasticsearch    mail.info      unattended-upgrades
bootstrap.log       error            mail.log       user.log
btmp                error.1          mail.warn      user.log.1
btmp.1              error.2.gz       messages       user.log.2.gz
cron.log            faillog          messages.1     wtmp
cron.log.1          freq_server      messages.2.gz  wtmp.1
cron.log.2.gz       freq_server_dns  mysql          Xorg.0.log
curator             fsck             nsm            Xorg.0.log.old
daemon.log          gpu-manager.log  ntpstats
daemon.log.1        installer        redis
daemon.log.2.gz     kern.log         salt
```

c. Take some time to Google these secondary tools and answer the questions below:

For each one of the tools listed above, describe the function, importance, and placement in the security analyst workflow.

Reflection

Log normalization is important and depends on the deployed environment.

Popular tools include their own normalization features, but log normalization can also be done manually.

When manually normalizing and preparing log files, double-check scripts to ensure the desired result is achieved. A poorly written normalization script may modify the data, directly impacting the analyst's work.