XML Event Logs

Introduction

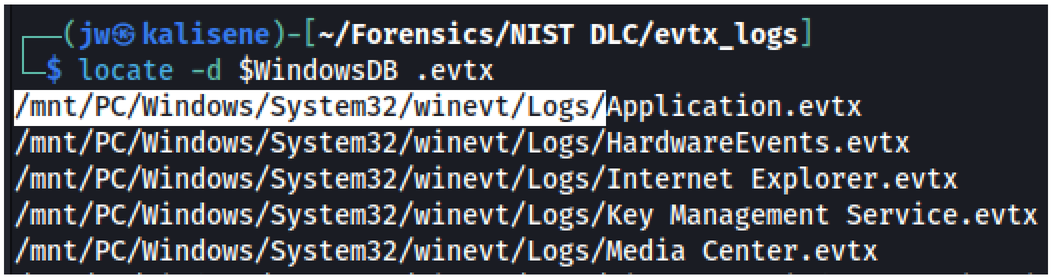
EVTX files, or Windows XML Event Logs, are a type of file that stores information about things happening in the system. Essentially, if something happens, Windows takes note of it. Forensic analysts can then go through these logs to identify *what* happened.

Parsing these files can get overwhelming very quickly. There are already automated systems that can do this for you, but for this tutorial, I'm going to parse them *mostly* by hand. This will give you a low-level view into these files and how they're structured so you can leverage this understanding for similar problems. You may not actually use this technique in the real world, but I appreciate the time it takes to learn something new.

Reading:

- https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx
- https://computingforgeeks.com/the-ultimate-guide-to-windows-event-logging/
- https://github.com/libyal/libevtx/
- https://forensics.wiki/windows_xml_event_log_%28evtx%29/
- https://www.worldtimeserver.com/learn/what-is-utc-time-format/
- https://medium.com/@jcm3/windows-event-logs-tryhackme-walkthrough-b3d91eade1d2

Let's start by locating all EVTX files on the Windows image:



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ locate -d $WindowsDB .evtx
/mnt/PC/Windows/System32/winevt/Logs/Application.evtx
/mnt/PC/Windows/System32/winevt/Logs/HardwareEvents.evtx
/mnt/PC/Windows/System32/winevt/Logs/Internet Explorer.evtx
/mnt/PC/Windows/System32/winevt/Logs/Key Management Service.evtx
/mnt/PC/Windows/System32/winevt/Logs/Media Center.evtx
```

Windows stores them all in the same directory. However, if you count them, there's 54 files, which is a lot to parse. As usual, the internet is your friend. Before you dive into any of these files, ask yourself: What exactly are you looking for?

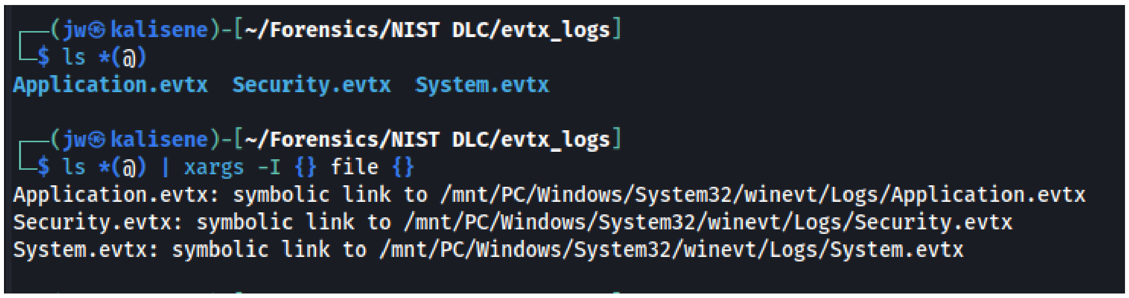
First off, the Data Leakage Case asks you to find "traces of system on/off and user logon/logoff". We can assume these are unique events with their own IDs. To find these IDs, I recommend the following site: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx. I simply searched by keyword to get a small list of event IDs:

- 4624: a user successfully logged on
- 4647: a user initiated logged off
- 4608: the system is starting up
- 4609: the system is shutting down
- 1100: the logging service has stopped

Interestingly, ultimatewindowssecurity.com notes that the event for a normal shutdown doesn't normally occur and you should instead look for event ID 1100. Keep this in mind when you're parsing the logs.

Now the question is which specific logs to look through. I recommend the following link: https://computingforgeeks.com/the-ultimate-guide-to-windows-event-logging/. It briefly describes the events contained a few different logs. The Security log is likely valuable as it tracks user logins, so I'm going to start with that.

With an idea of what to look for, we can start parsing the logs. I made symbolic links to the Application, Security and System logs. In my previous tutorials, I used an *awk* command to execute another command on each file. Now I'm using *xargs*, which I've found to be much more useful and succinct.



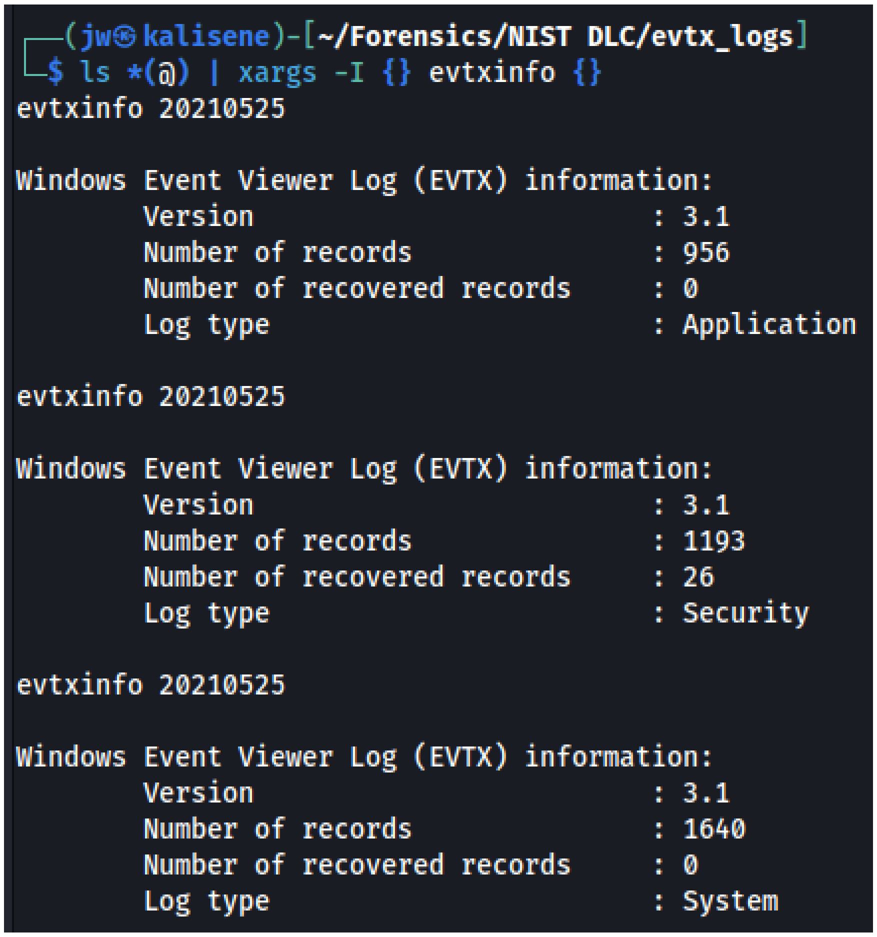
```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ ls *(@)
Application.evtx  Security.evtx  System.evtx

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ ls *(@) | xargs -I {} file {}
Application.evtx: symbolic link to /mnt/PC/Windows/System32/winevt/Logs/Application.evtx
Security.evtx: symbolic link to /mnt/PC/Windows/System32/winevt/Logs/Security.evtx
System.evtx: symbolic link to /mnt/PC/Windows/System32/winevt/Logs/System.evtx
```

Figure 1: This weird ls command only shows links. I wanted to filter out the other files in this directory just to show you the links.

Just like the previous tutorials, we have **libevtx** to help us parse these files:

- **evtxinfo**: Shows information about EVTX files
- **evtxexport**: Exports EVTX files into a readable format

Using them is as easy as modifying my previous command:

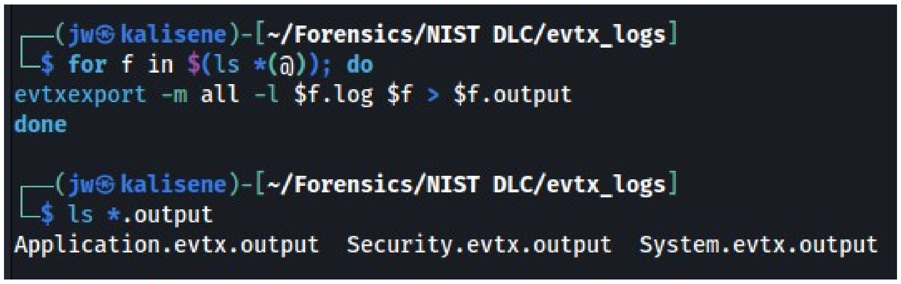


```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ ls *(@) | xargs -I {} evtxinfo {}
evtxinfo 20210525

Windows Event Viewer Log (EVTX) information:
        Version                              : 3.1
        Number of records                    : 956
        Number of recovered records          : 0
        Log type                             : Application

evtxinfo 20210525

Windows Event Viewer Log (EVTX) information:
        Version                              : 3.1
        Number of records                    : 1193
        Number of recovered records          : 26
        Log type                             : Security

evtxinfo 20210525

Windows Event Viewer Log (EVTX) information:
        Version                              : 3.1
        Number of records                    : 1640
        Number of recovered records          : 0
        Log type                             : System
```

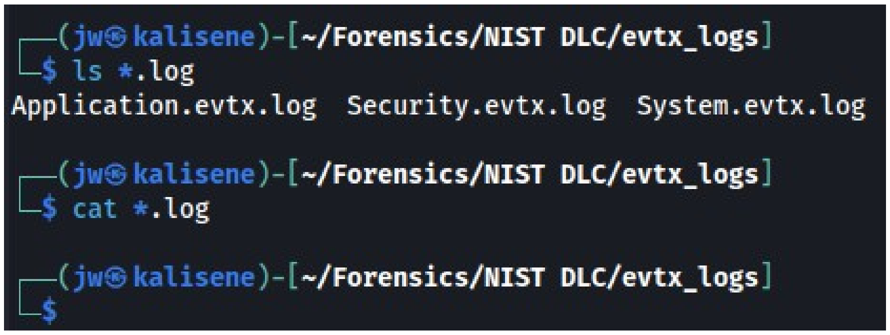
xargs takes the output from the command on the left and sends it as input to the command on the right. The braces identify the output (for example, it's the same as running **evtxinfo Application.evtx**, then **evtxinfo Security.evtx** and so on).

The output shows the numbers of both normal and recovered records. I recommend exporting all the records for each log file. The log type speaks for itself—System.evtx is the System log type.

Moving on to exporting:
I provided the **-l** option just in case **evtxexport** had information to log. There was nothing. I'd still recommend using logs if programs support them just to see if they provide anything useful.

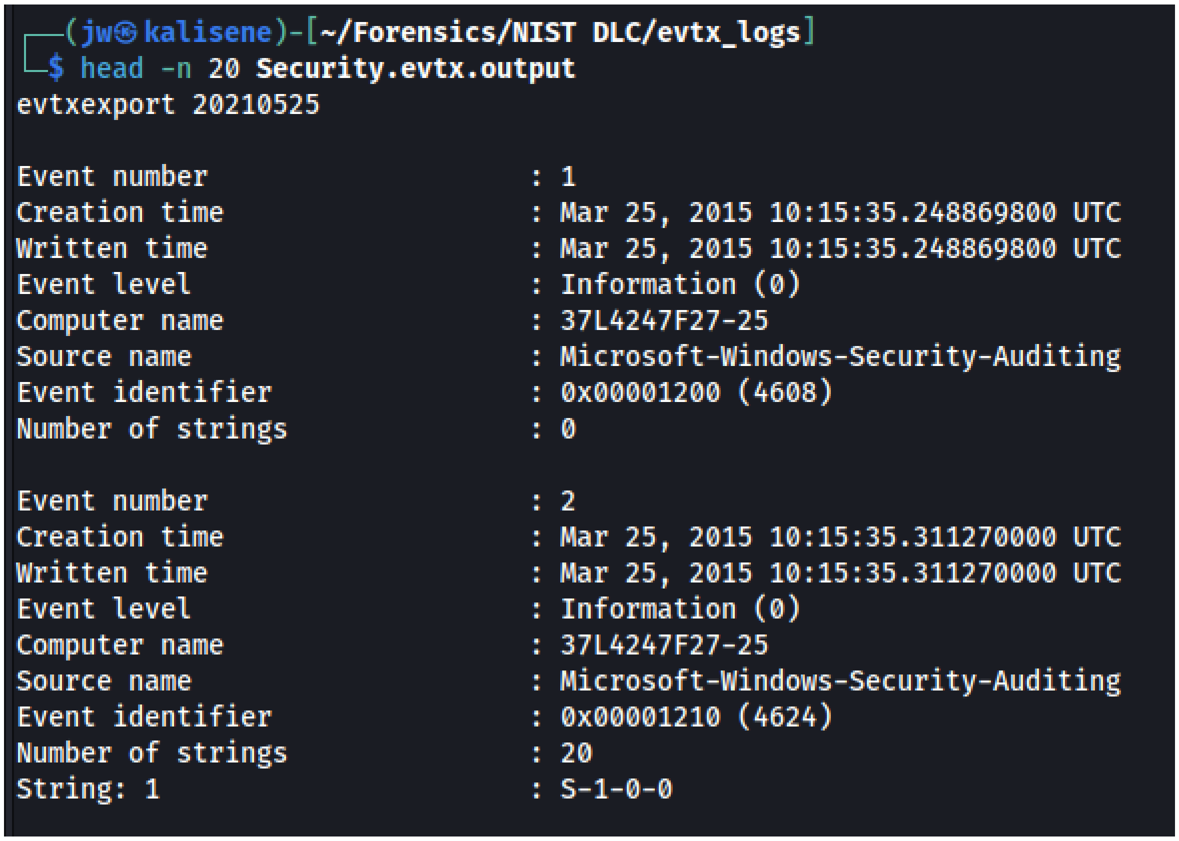


```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ for f in $(ls *(@)); do
evtxexport -m all -l $f.log $f > $f.output
done

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ ls *.output
Application.evtx.output  Security.evtx.output  System.evtx.output
```



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ ls *.log
Application.evtx.log  Security.evtx.log  System.evtx.log

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ cat *.log

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$
```

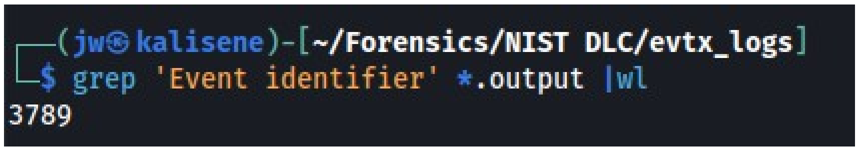
Now let’s look at the exported information:



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ head -n 20 Security.evtx.output
evtxexport 20210525

Event number                    : 1
Creation time                   : Mar 25, 2015 10:15:35.248869800 UTC
Written time                    : Mar 25, 2015 10:15:35.248869800 UTC
Event level                     : Information (0)
Computer name                   : 37L4247F27-25
Source name                     : Microsoft-Windows-Security-Auditing
Event identifier                : 0x00001200 (4608)
Number of strings               : 0

Event number                    : 2
Creation time                   : Mar 25, 2015 10:15:35.311270000 UTC
Written time                    : Mar 25, 2015 10:15:35.311270000 UTC
Event level                     : Information (0)
Computer name                   : 37L4247F27-25
Source name                     : Microsoft-Windows-Security-Auditing
Event identifier                : 0x00001210 (4624)
Number of strings               : 20
String: 1                       : S-1-0-0
```

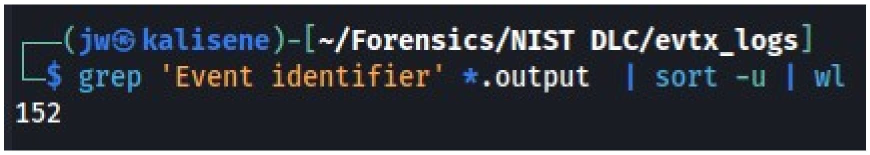
“Event identifier” will have the codes we are looking for. Notice the codes 4608 and 4624, which we identified earlier, show up in the Security output, so this will definitely be an important file.

Now I want to look through all event identifiers in the output files to see how many unique codes there are:



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ grep 'Event identifier' *.output |wl
3789
```

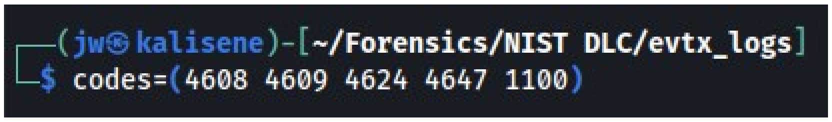
Figure 2: Reminder: I made "wl" an alias for "wc -l" to count the lines.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ grep 'Event identifier' *.output  | sort -u | wl
152
```

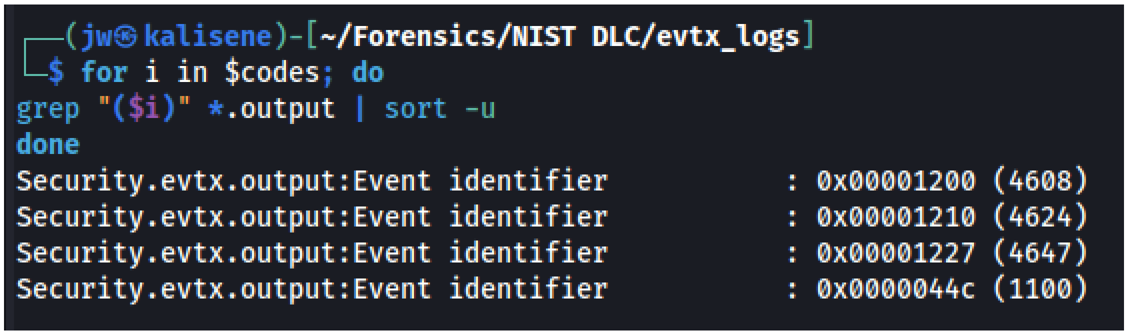
Figure 3: "sort -u" has quickly become one of my favourite tools to filter output.

So that’s 152 unique codes across the three output files. Thankfully, we already know what we want. I built an array with the codes I want to look for:



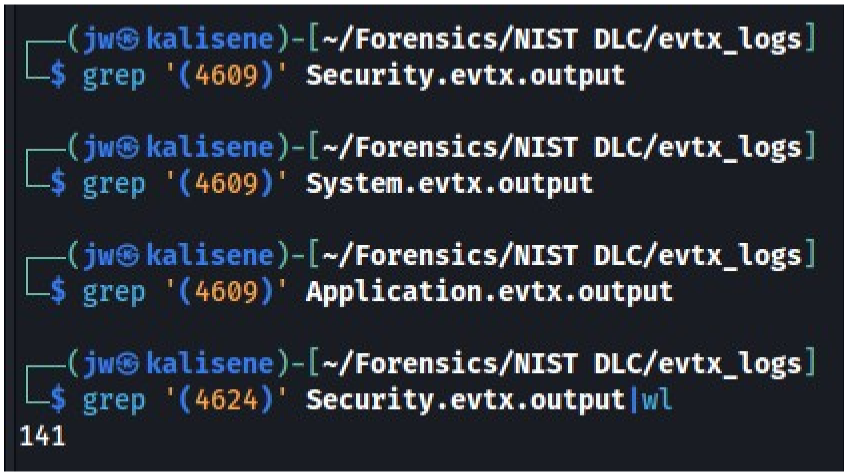
```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ codes=(4608 4609 4624 4647 1100)
```

Then I searched for each code in each output file:



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ for i in $codes; do
grep "($i)" *.output | sort -u
done
Security.evtx.output:Event identifier           : 0x00001200 (4608)
Security.evtx.output:Event identifier           : 0x00001210 (4624)
Security.evtx.output:Event identifier           : 0x00001227 (4647)
Security.evtx.output:Event identifier           : 0x0000044c (1100)
```

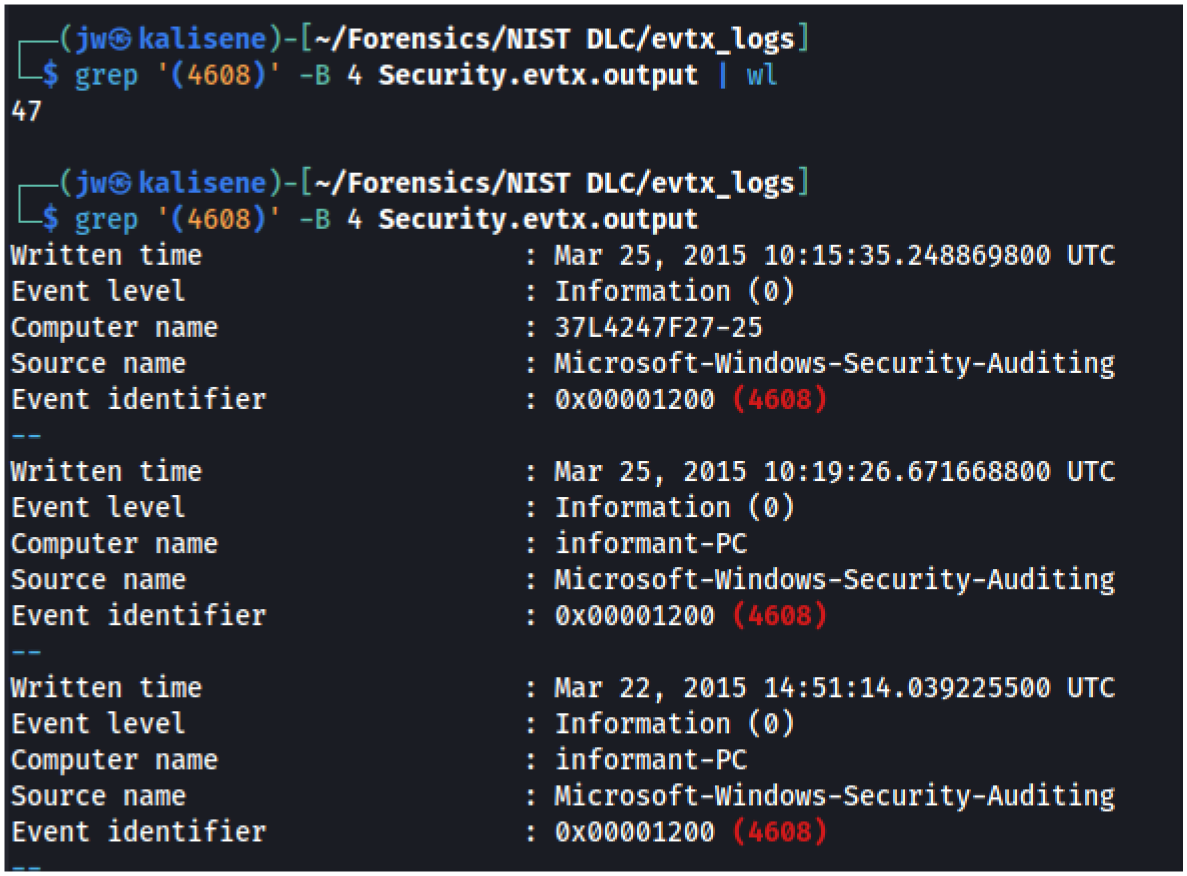
Note that the code 4609 doesn’t show up in any of the output files, but the code 4624 does. In this case, we should assume 1100 indicates the system shut down normally.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ grep '(4609)' Security.evtx.output

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ grep '(4609)' System.evtx.output

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ grep '(4609)' Application.evtx.output

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ grep '(4624)' Security.evtx.output|wl
141
```

This just tells us the codes we are looking for are only in the Security file. However, I’ve found parsing the output file much harder than similar output files in previous tutorials. There are some simple ways to display it, albeit in an unorganized way.

For example, we can search for the event code and a few lines preceding it:

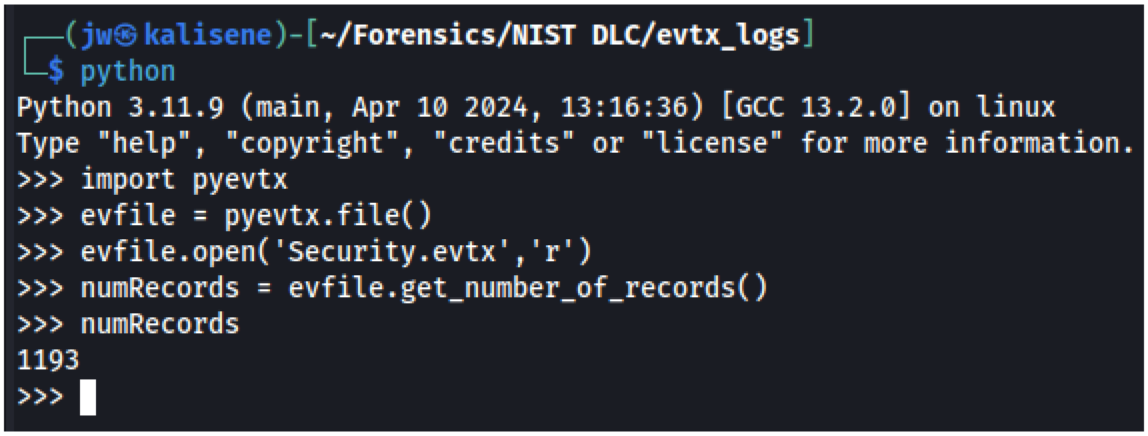


```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ grep '(4608)' -B 4 Security.evtx.output | wl
47

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ grep '(4608)' -B 4 Security.evtx.output
Written time                    : Mar 25, 2015 10:15:35.248869800 UTC
Event level                     : Information (0)
Computer name                   : 37L4247F27-25
Source name                     : Microsoft-Windows-Security-Auditing
Event identifier                : 0x00001200 (4608)
--
Written time                    : Mar 25, 2015 10:19:26.671668800 UTC
Event level                     : Information (0)
Computer name                   : informant-PC
Source name                     : Microsoft-Windows-Security-Auditing
Event identifier                : 0x00001200 (4608)
--
Written time                    : Mar 22, 2015 14:51:14.039225500 UTC
Event level                     : Information (0)
Computer name                   : informant-PC
Source name                     : Microsoft-Windows-Security-Auditing
Event identifier                : 0x00001200 (4608)
```

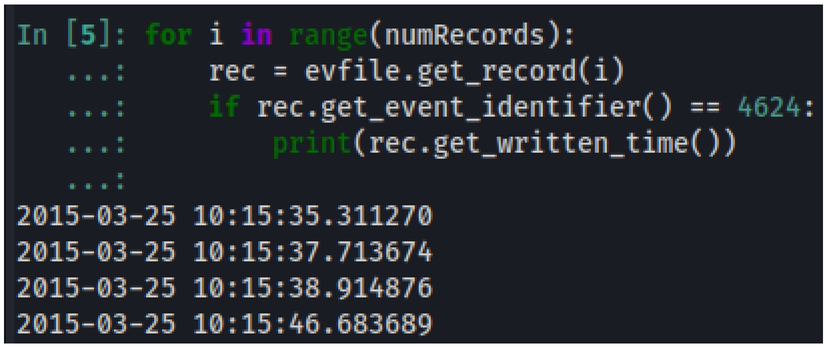
This isn't a ton of information to sort through. However, we aren't quite interested in the event level, computer name or source name at the moment; this is only the information for one code, so there will be more to comb through; and the written time is in UTC, or Coordinated Universal Time, which is different from the time zone you identified earlier in the exercise. So we need a way to isolate *just the information we need* and *convert the written time*.

I decided to pivot to Python to finish this tutorial. Thankfully, the creator of libevtx also made Python bindings with the same library. I'm very familiar with Python, and I recommend it as it provides greater flexibility for a wider range of problems than just what Bash or Zsh can offer.

I won't be going over installation of the Python module—I'll jump straight into basic usage. Firing up the REPL environment, let's look at some basic commands:


```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ python
Python 3.11.9 (main, Apr 10 2024, 13:16:36) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyevtx
>>> evfile = pyevtx.file()
>>> evfile.open('Security.evtx','r')
>>> numRecords = evfile.get_number_of_records()
>>> numRecords
1193
>>> 
```

We can filter out the events based on our list of codes. To illustrate that, I've only used one code:

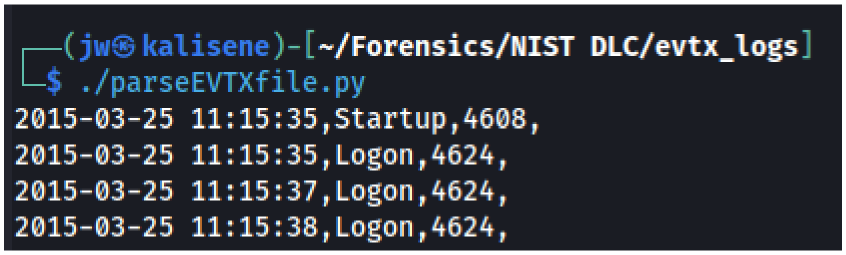

```
In [5]: for i in range(numRecords):
   ...:     rec = evfile.get_record(i)
   ...:     if rec.get_event_identifier() == 4624:
   ...:         print(rec.get_written_time())
   ...:
2015-03-25 10:15:35.311270
2015-03-25 10:15:37.713674
2015-03-25 10:15:38.914876
2015-03-25 10:15:46.683689
```

The exercise also has you isolating the events of interest to a specific time range—0900 to 1800. Once you get the times, you will need to assert the written time, in EST, is within that time range:

1. Get the UTC date
2. Convert to EST date
3. Isolate the hourly time from the EST date
4. If the time is between 0900 and 1800: print date, time and code.

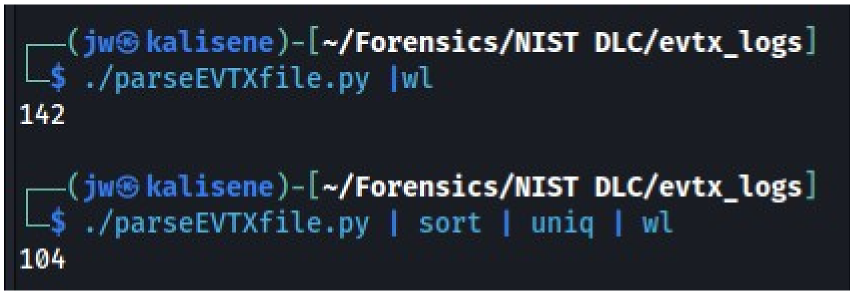
You'll notice the official answer sheet provides the codes with a description. The descriptions aren't part of the **pyevtx** library and isn't included in the **evtxexport** information either, so you'll need to map the codes to their descriptions yourself. This will be useful if you have to explain what happened to non-technical users.

I wrapped up everything in a script. My final output looks like this:



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ ./parseEVTXfile.py
2015-03-25 11:15:35,Startup,4608,
2015-03-25 11:15:35,Logon,4624,
2015-03-25 11:15:37,Logon,4624,
2015-03-25 11:15:38,Logon,4624,
```

Note that there are some duplicate entries. I assume this is a feature of Windows' logging service. Again, if you reference the answer sheet, it ignores duplicates.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ ./parseEVTXfile.py |wl
142

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/evtx_logs]
└─$ ./parseEVTXfile.py | sort | uniq | wl
104
```

Save your output to a file so you can use it later.

Conclusion

This was a much harder task than I initially expected. I spent a lot of time researching the file format/ different tools and playing with the output to see what worked and what made sense. I've appreciated keeping everything limited to the command line, but I had to break out and use Python to finish this task. But this makes sense: