

Log File Analysis with Python

Processing Log Files



Cristian Pascariu

Information Security Professional

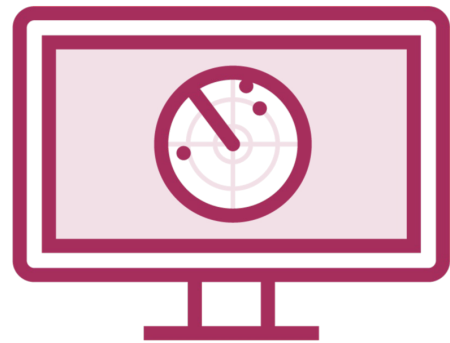
www.cybersomething.com



The Importance of Log Files



Log files have been used for a very long time by IT professionals to troubleshoot services and applications



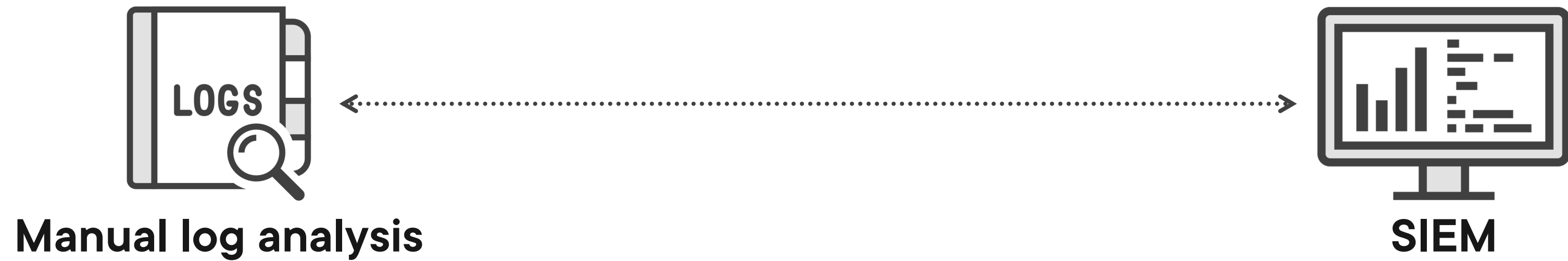
For security people, log files represent a detailed and high-fidelity source of insight that enable defenders to gain visibility



Attackers will try to purge the log files or suspend the logging services in order to cover their tracks



Log Analysis in Practice



Log Analysis in Practice



Manual log analysis



SIEM

Perfect for small investigations

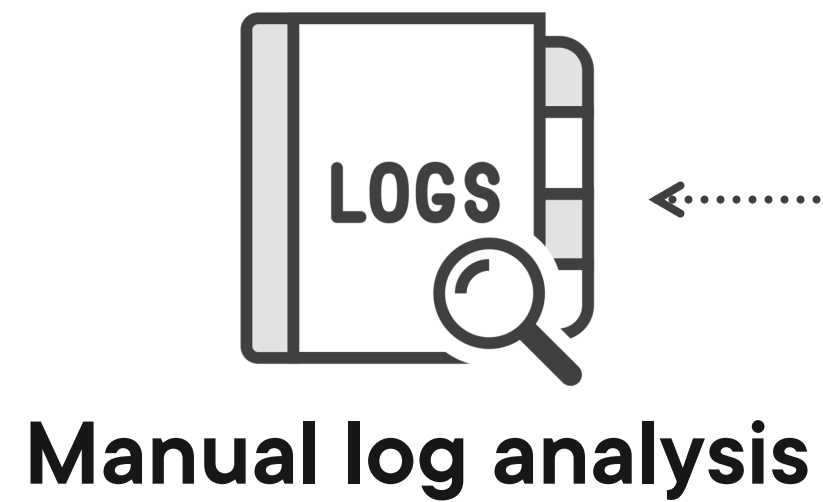
High-fidelity events

Time consuming

Does not scale well



Log Analysis in Practice



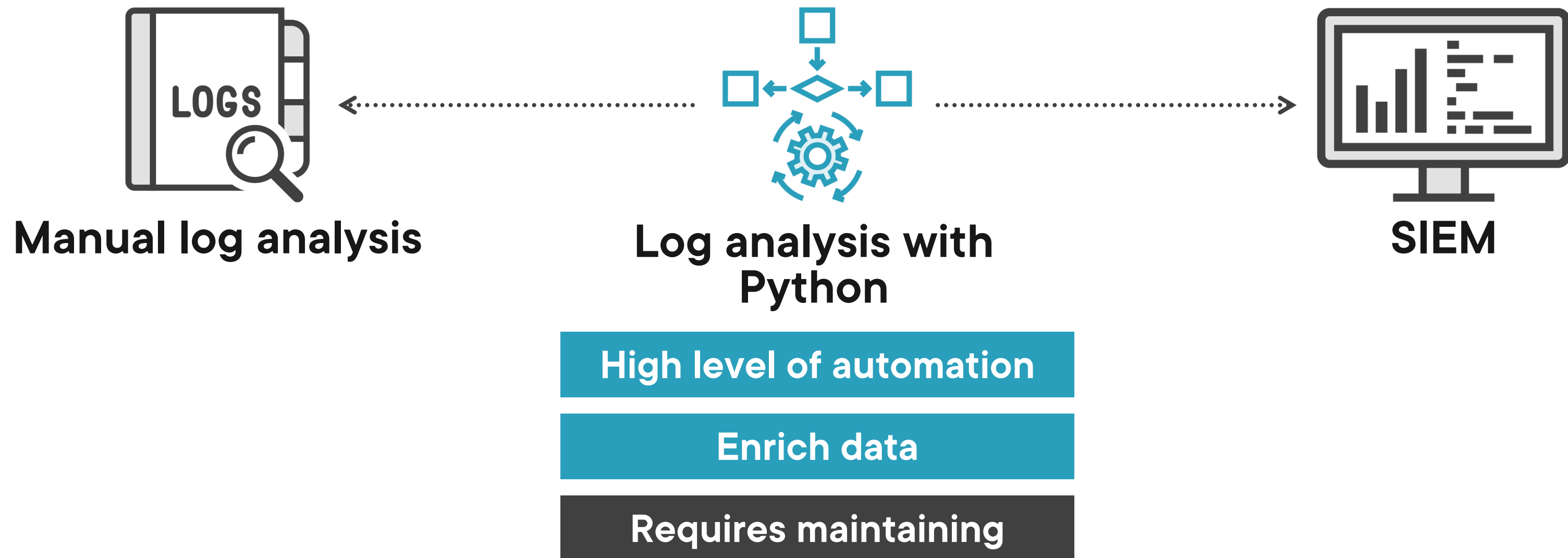
At scale monitoring and correlation

Costly

Limited by product features



Log Analysis in Practice



Overview



Overview of the opportunities and pitfalls when considering automating log analysis

Parse log files

- Based on a standard delimiter
- Using regular expressions to carve out individual fields

Parse windows log files based on the EVTX file format

Consider all scripts as building blocks



Good Use Cases



Automate manual analysis

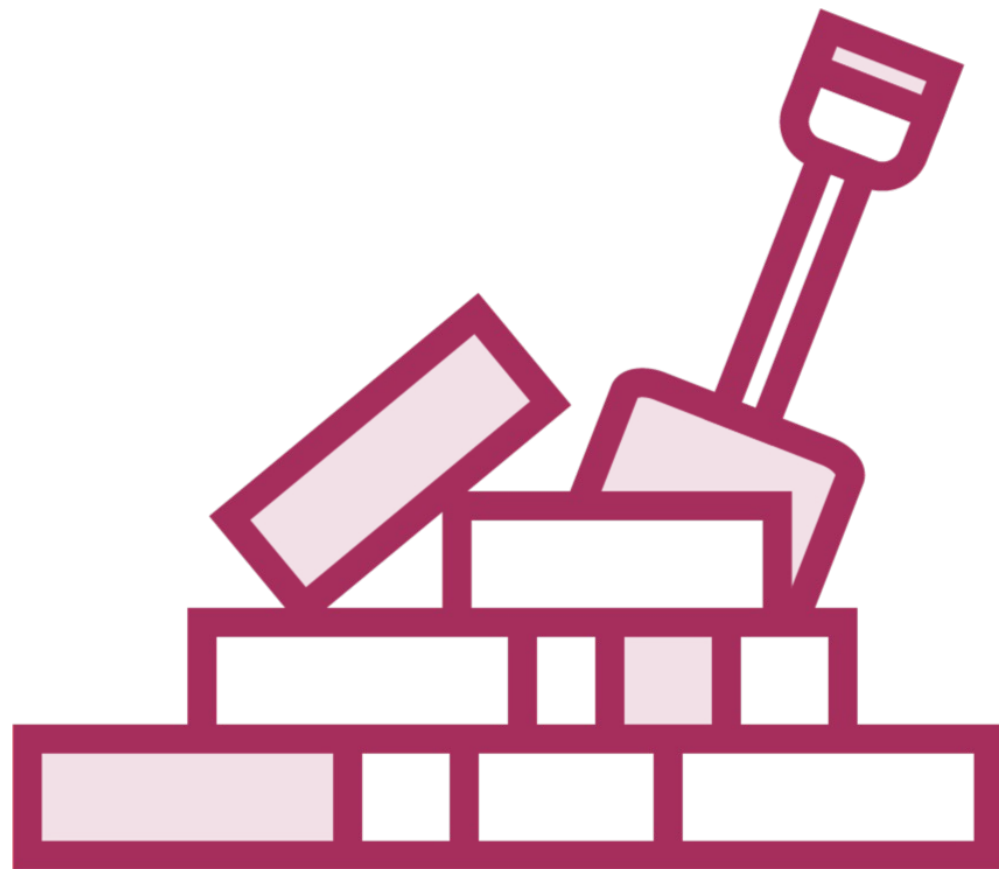
Log enrichment and correlation

Log data analytics beyond what is already offered by existing SIEM solution

Build your own solution



Don't Reinvent the Wheel



Creating a single purpose script

- Focus reusable components

Rewriting tools requires maintaining

Don't build log shipping agents that already exist

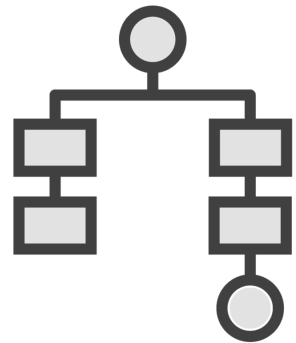


Log File Processing Pipeline



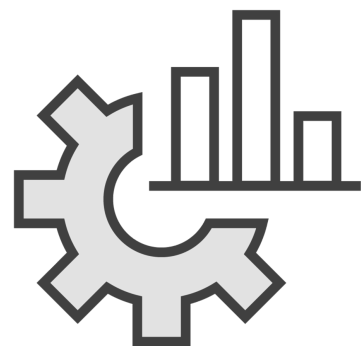
Open log file

Read through all the log entries



Parse log records

Parse log entries into fields



Analytics

Filters, search for indicators, correlate events, enhance log data



Setting up your Development Environment



Python3 is supported on the majority of Linux distributions

- On Windows it can be installed or used through WSL

Use your favorite text editor or IDE

- Vscode is used in all demos



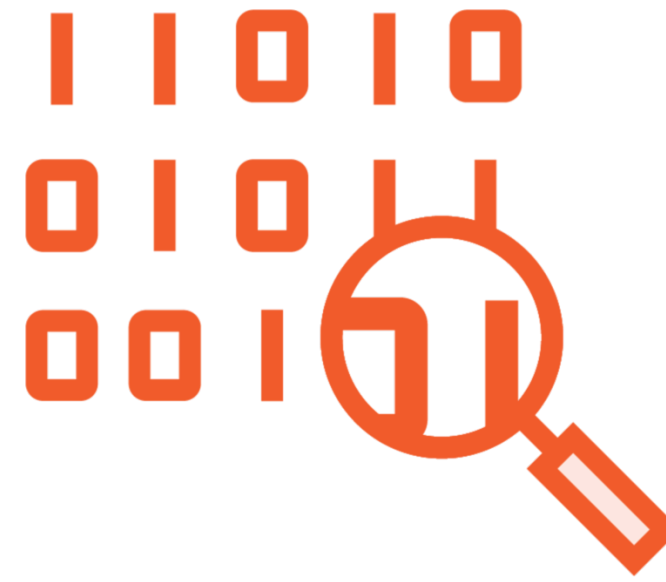
Getting Familiar with the Demo Files



**SMB logs from a
Ransomware
infection**



**Auth logs from a
SSH brute force
attack**



**Zeek logs from a
reverse shell**



**Windows logs
from a backdoor
infection**

Reading Log Files



Log files are essentially text files where every line is basically a log entry

As log files can be quite large, we'll analyze it line by line as opposed to loading it to memory

- Filters when copying entries to memory

Leverage exception handling for parsing errors

```
def openLogFile(path):  
    with open(path) as log_file:  
        for log_entry in log_file.read():  
            #process log record
```

Reading Log Files

Open the log file as a text file and read its content line by line

```
def openLogFile(path):  
    with open(path) as log_file:  
        for log_entry in log_file.read():  
            yield log_entry
```

Reading Log Files

Leverage generators to move detection logic in another function

```
def openLogFile(path):  
    with open(path) as log_file:  
        for log_entry in log_file.read():  
            yield log_entry  
  
def detection():  
    log_file = openLogFile('logfile.log')  
    for log_entry in log_file:  
        #process log record
```

Reading Log Files

Leverage generators to move detection logic in another function

Demo



Walkthrough the log files

Check the Python version

Create a virtual environment to manage dependencies

Write the code to open a log file for retrieving information



Following Best Practices



Leverage version control

- Encourage collaboration within the team or the community

Make your scripts and tools run on other systems

- Manage dependencies with virtual environments

Document your script

- At least with comments



Parsing Log Files



Parse each log entry into individual fields

Enables field-based filtering

Perform operations based on data type

- Apply math functions on numerical data
- Perform timeline analysis based on date and time

Parsing Log Records Based on Separator

Raw log entry

##:##:## hostname event_id event_description

Identifying
separators

##:##:## | hostname | event_id | event_description

##:##:##, hostname, event_id, event_description



Parsing into
individual fields

Timestamp

Hostname

Event ID

Event description



```
import re

def parseZeekConn(log_entry):
    log_data = re.split("\t", log_entry.rstrip())
```

Parse Log Records

Leverage the re module to split the log entry based on the known delimiter

```
import re

def parseZeekConn(log_entry):
    log_data = re.split("\t", log_entry.rstrip())
    r = {}
    r["uid"] = log_data[1]
    # list of fields
```

Parse Log Records

Create a dictionary where the key is the name of the field and the value is the extracted data corresponding to that field

Demo



Parsing Zeek log files based on tab separator



Leveraging Regular Expressions



Log file types may include unstructured data

- There isn't a single delimiter

Leverage regular expressions to carve out each field based on the field type and order

- The challenge lies in identifying patterns

Regular Expression Cheat Sheet

^	Start of the string	{min, max}	Minimum and maximum of instances of the previous RE
\$	End of the string	 	Either operator
.	Matches any character	[]	Character set
\	Escape special character	()	Group
*	Zero or more instances of the previous RE	(?P<group_name>)	Named group
+	One or more instances of the previous RE		



Regular Expressions in Python



The RE module provides access to regular expression functions

Search and match are the two functions that can be used to identify patterns

Python supports the concept of named groups

- Add named groups for each field in the pattern
- Extract these fields as a dictionary



Parsing Logs with Regular Expressions

Log record

10:14:56 : win10-charlie|192.168.55.133|RShare|pwrite|ok|document-Charlie-1.txt

Regular expression

```
^(?P<time>[0-9]{2}:[0-9]{2}:[0-9]{2})  
\s\:\s(?P<hostname>[a-zA-Z0-9\-\_]+)\|  
(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})\|  
(?P<share>[a-zA-Z]+)\|  
(?P<operation>[a-zA-Z]+)\|  
ok\|  
(?P<path>.*)$
```



Parsing Logs with Regular Expressions

Log record

10:14:56 : win10-charlie|192.168.55.133|RShare|pwrite|ok|document-Charlie-1.txt

Regular expression

```
^(?P<time>[0-9]{2}:[0-9]{2}:[0-9]{2})
\s\:(?P<hostname>[a-zA-Z0-9\-\_]+)\|
(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})\|
(?P<share>[a-zA-Z]+)\|
(?P<operation>[a-zA-Z]+)\|
ok\|
(?P<path>.*)$
```



Parsing Logs with Regular Expressions

Log record

10:14:56 : win10-charlie|192.168.55.133|RShare|pwrite|ok|document-Charlie-1.txt

Regular expression

```
^(?P<time>[0-9]{2}:[0-9]{2}:[0-9]{2})
\s\:\s(?P<hostname>[a-zA-Z0-9\-\_]+)\|
(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})\|
(?P<share>[a-zA-Z]+)\|
(?P<operation>[a-zA-Z]+)\|
ok\|
(?P<path>.*)$
```



Parsing Logs with Regular Expressions

Log record

10:14:56 : win10-charlie|192.168.55.133|RShare|pwrite|ok|document-Charlie-1.txt

Regular expression

```
^(?P<time>[0-9]{2}:[0-9]{2}:[0-9]{2})
\s\:(?P<hostname>[a-zA-Z0-9\-\_]+)\|
(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})\|
(?P<share>[a-zA-Z]+)\|
(?P<operation>[a-zA-Z]+)\|
ok\|
(?P<path>.*)$
```



Parsing Logs with Regular Expressions

Log record

10:14:56 : win10-charlie | 192.168.55.133 | RShare | pwrite | ok | document-Charlie-1.txt

Regular expression

```
^(?P<time>[0-9]{2}:[0-9]{2}:[0-9]{2})  
\s\:\s(?P<hostname>[a-zA-Z0-9\ -]+)\|  
(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})\|  
(?P<share>[a-zA-Z]+)\|  
(?P<operation>[a-zA-Z]+)\|  
ok\|  
(?P<path>.*)$
```



Parsing Logs with Regular Expressions

Log record

10:14:56 : win10-charlie|192.168.55.133|RShare|pwrite|ok|document-Charlie-1.txt

Regular expression

```
^(?P<time>[0-9]{2}:[0-9]{2}:[0-9]{2})
\s\:(?P<hostname>[a-zA-Z0-9\ -]+)\|
(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})\|
(?P<share>[a-zA-Z]+)\|
(?P<operation>[a-zA-Z]+)\|
ok\|
(?P<path>.*)$
```



Parsing Logs with Regular Expressions

Log record

10:14:56 : win10-charlie|192.168.55.133|RShare|pwrite|ok|document-Charlie-1.txt

Regular expression

```
^(?P<time>[0-9]{2}:[0-9]{2}:[0-9]{2})
\s\:(?P<hostname>[a-zA-Z0-9\_-]+)\|
(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})\|
(?P<share>[a-zA-Z]+)\|
(?P<operation>[a-zA-Z]+)\|
ok\|
(?P<path>.*)$
```



Parsing Logs with Regular Expressions

Log record

10:14:56 : win10-charlie|192.168.55.133|RShare|pwrite|ok|document-Charlie-1.txt

Regular expression

```
^(?P<time>[0-9]{2}:[0-9]{2}:[0-9]{2})  
\s\:(?P<hostname>[a-zA-Z0-9\ -]+)\|  
(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})\|  
(?P<share>[a-zA-Z]+)\|  
(?P<operation>[a-zA-Z]+)\|  
ok\|  
(?P<path>.*)$
```



```
import re

def parseZeekConn(log_entry):
    log_data = re.search(pattern, log_entry.rstrip())
```

Parse Log Records using Regular Expressions

The re module has two functions, we are using search to identify the pattern in the log entry

Demo



Parse SMB logs using regular expressions

- Leverage CyberChef to build the regex pattern



Working with Windows Logs



Windows uses a proprietary file format

- Logs are stored in EVTX files

Parse files using a specific module

- Python-evtx
- Standalone scripts available

Good for forensics where host logs are not monitored



```
import Evtx.Evtx as evtx

def openEvtxFFile(path):
    with evtx.Evtx(path) as log_file:
        for log_entry in log_file.records():
            yield log_entry.xml()
```

Parse Windows Log Files

Extract each event as an XML object

```
Sys_tag = event.find("System",  
event.nsmap)
```

```
<System>  
    <EventID>  
  
    </EventID>  
</System>  
  
<EventData>  
    <Data name="Key">Value</Data>  
</EventData>
```

```
Sys_tag = event.find("System",  
event.nsmap)
```

```
Event_id = Sys_tag.find("EventID",  
event.nsmap)
```

```
<System>
```

```
    <EventID>
```

```
    </EventID>
```

```
</System>
```

```
<EventData>
```

```
    <Data name="Key">Value</Data>
```

```
</EventData>
```



```
Sys_tag = event.find("System",  
event.nsmap)
```

```
Event_id = Sys_tag.find("EventID",  
event.nsmap)
```

```
EventData = event.find("EventData",  
event.nsmap)
```

```
<System>  
    <EventID>  
  
    </EventID>  
</System>
```

```
<EventData>  
    <Data name="Key">Value</Data>  
</EventData>
```

```
Sys_tag = event.find("System",
event.nsmmap)

Event_id = Sys_tag.find("EventID",
event.nsmmap)

EventData = event.find("EventData",
event.nsmmap)

Data = EventData.getchildren()
```

```
<System>
    <EventID>

    </EventID>
</System>

<EventData>
    <Data name="Key">Value</Data>
</EventData>
```

```
Sys_tag = event.find("System",  
event.nsmap)  
  
Event_id = Sys_tag.find("EventID",  
event.nsmap)  
  
EventData = event.find("EventData",  
event.nsmap)  
  
Data = EventData.getchildren()  
  
Data[0].getattrib("Name")
```

```
<System>  
    <EventID>  
  
    </EventID>  
</System>  
  
<EventData>  
    <Data name="Key">Value</Data>  
</EventData>
```

```
Sys_tag = event.find("System",  
event.nsmap)  
  
Event_id = Sys_tag.find("EventID",  
event.nsmap)  
  
EventData = event.find("EventData",  
event.nsmap)  
  
Data = EventData.getchildren()  
  
Data[0].text()
```

```
<System>  
    <EventID>  
  
    </EventID>  
</System>  
  
<EventData>  
    <Data name="Key">Value</Data>  
</EventData>
```

Demo



Install python-evtx package

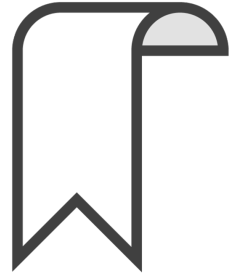
Parse Windows security log

Detect malicious activity

- Filter based on EventID
- Detecting suspicious parent-child process relationship

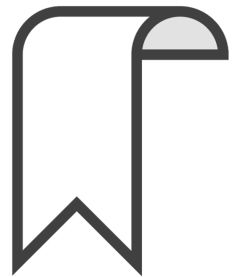


Resources Referenced in this Module



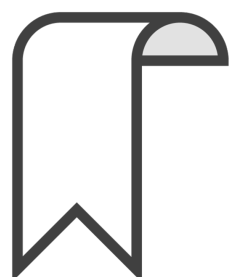
Zeek tab separated values log format

<https://docs.zeek.org/en/master/log-formats.html#zeek-tsv-format-logs>



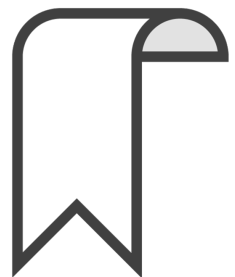
Python RE module

<https://docs.python.org/3/library/re.html>



Python-evtx module

<https://github.com/williballenthin/python-evtx>



MS Windows EventID: 4688

<https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4688>



Summary



Identify opportunities where log analysis can be codified with Python

- Automating manual analysis
- Enriching log data

Basic operations such as reading log files and searching for indicators

Parsed information to prepare it for the next modules where we'll perform deeper data analysis

