# IT Innovations: A Portfolio of Projects from my Internship

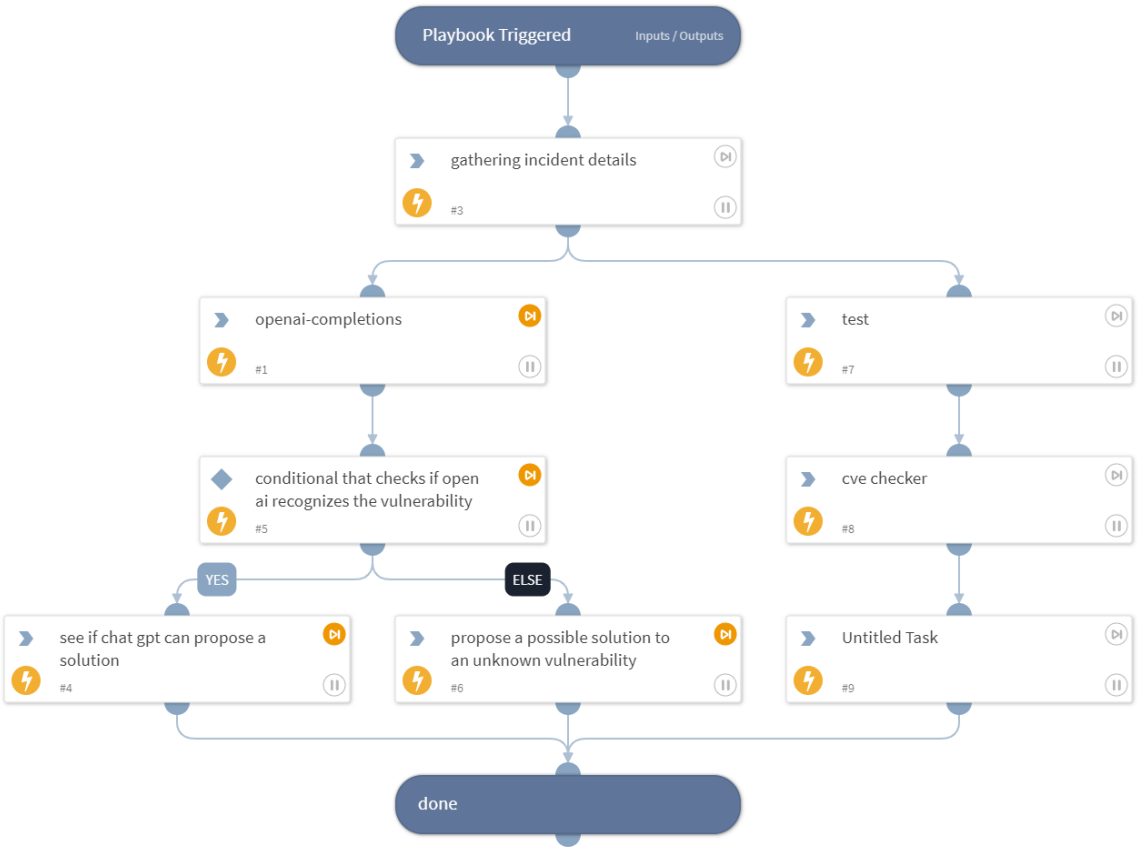
## Table of Contents

## Technologies used/ researched:

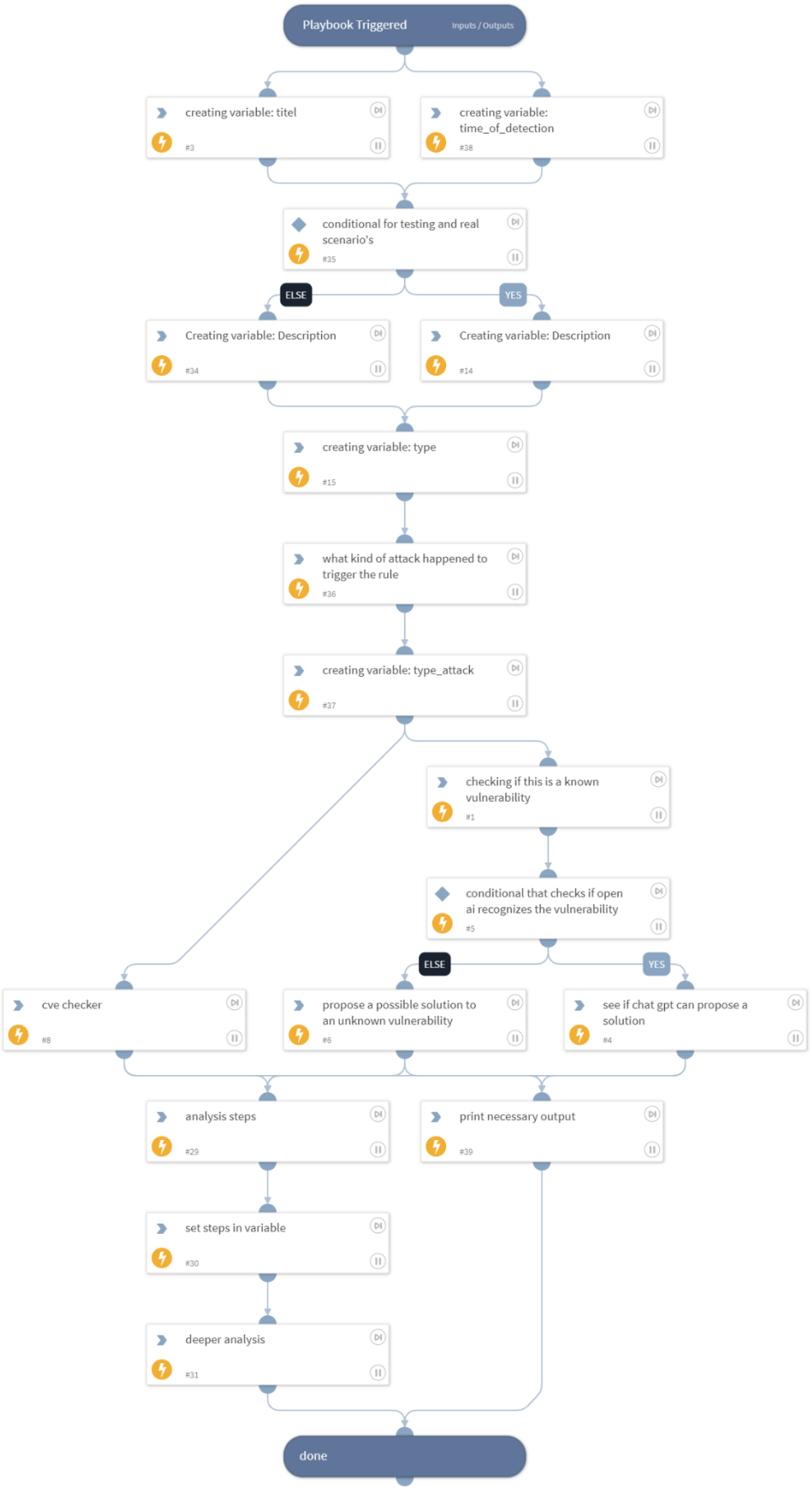
- Xsoar
- Xsoar ml
- Python
- Python scikit
- Openai ChatGPT

## Xsoar playbooks

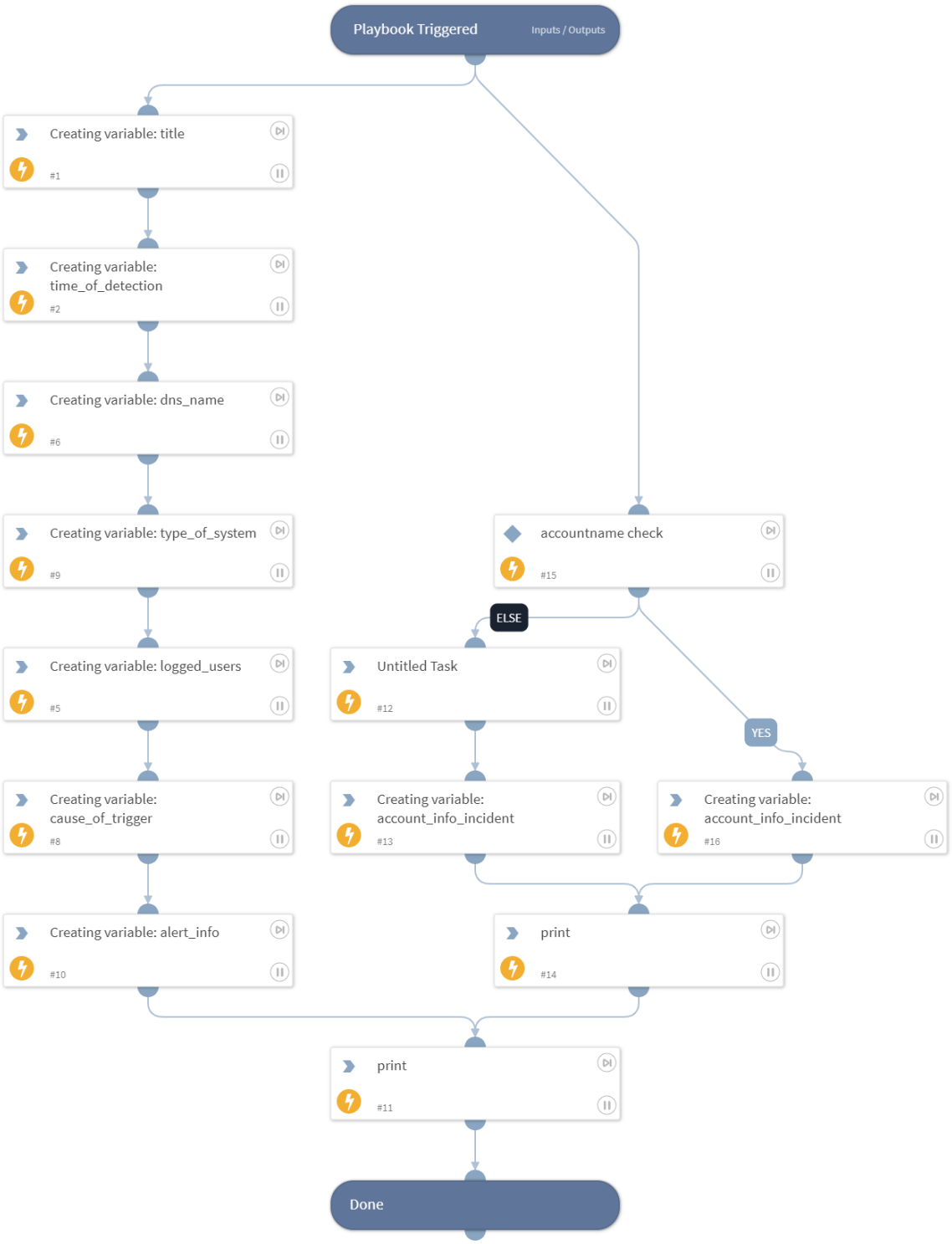
### Xsoar chatgpt test run:



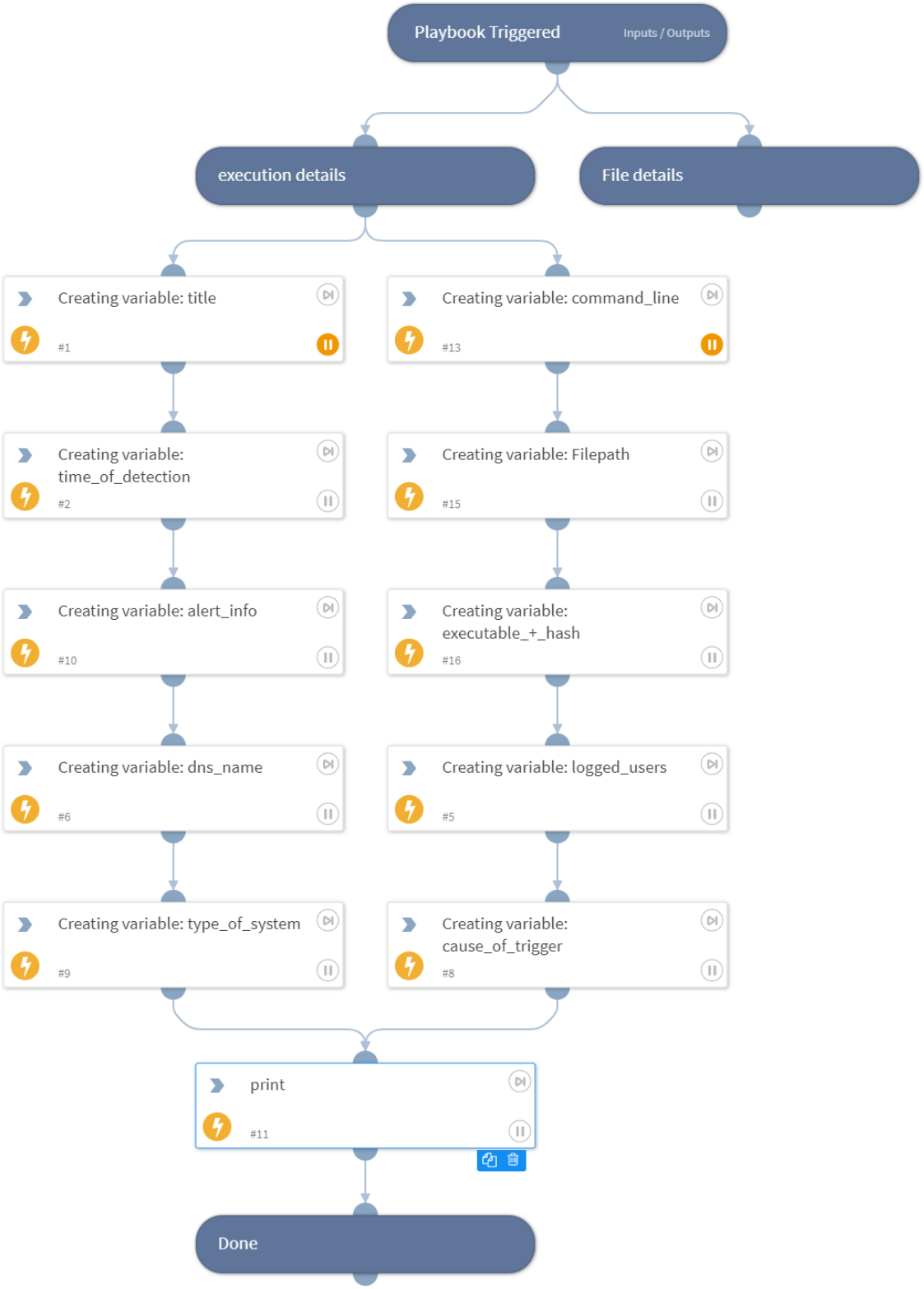
## Playbook for investigation:


Playbook Triggered
Inputs / Outputs
creating variable: titel
#3
creating variable: time_of_detection
#38
conditional for testing and real scenario's
#35
ELSE
YES
Creating variable: Description
#34
Creating variable: Description
#14
creating variable: type
#15
what kind of attack happened to trigger the rule
#36
creating variable: type_attack
#37
checking if this is a known vulnerability
#1
conditional that checks if open ai recognizes the vulnerability
#5
ELSE
YES
cve checker
#8
propose a possible solution to an unknown vulnerability
#6
see if chat gpt can propose a solution
#4
analysis steps
#29
print necessary output
#39
set steps in variable
#30
deeper analysis
#31
done

## Investigation automated response Microsoft defender:



## Investigation automated response crowdstrike(unfinished):



## Setting check:


Playbook Triggered
Inputs / Outputs
creating variable email_subject
#4
creating variable email_text
#3
creating variable: email_from
#14
creating variable: email_to
#15
running automatic mail classifier
#7
checking if multiple openai classifier
#9
ELSE
YES
creating variable: mail_variable
#10
creating variable: mail_variable
#11
checking for empty in mail_variable
#12
YES
ELSE
settting mail.check into context
#8
setting mail.check into context
#13
Done

## Xsoar Scripts(anonymized)(double-click on the code to see full script)

Indicator_dynamic:

```python
# Import necessary libraries
import urllib3
import demistomock as demisto
from CommonServerPython import *
import requests

# Disable insecure warnings
urllib3.disable warnings()

# Function to get links from a website using a session with authentication
# Not currently used because the program doesn't have a service account
# def get_cegeka_docs(url, username, password):
#     try:
#         session = requests.Session()
#         login_url = "https://www.example.com/"
#         session.post(login url, data={'username': username, 'password':
password})
#         response = session.get(url)
#         response.raise_for_status()
#     except requests.exceptions.HTTPError as e:
#         print(f"HTTP error occurred: {e}")
#         return None
#     except requests.exceptions.RequestException as e:
#         print(f"An error occurred: {e}")
#         return None
#     links = re.findall(r'href=[\'"]?([^\'" >]+)', response)
#     return links
#
# Function to get a link from a list of links based on a tag (mitreid)
# Not currently used because manual links are being used instead
# def get_attackpattern_link(tag, links):
#     result = []
#     for link in links:
#         if tag in link:
#             result.append(link)
#     return result

# This function retrieves the link for a given MITRE ID from a hardcoded
dictionary of links
def get_link_manual(tag):
    list_of_links = {'code':"https://www.example.com/"
                     'code2':"https://www.example.com/"}

    # If the given tag is in the dictionary of links, return the
corresponding link
    if tag in list_of_links:
        link = list_of_links[tag]
    # If the given tag is not in the dictionary of links, return an error
message
    else:
        link = "this mitreid does not have an existing link"
    return link


# This is the main function that gets called when the script runs
def main():
    # get the 'indicator' argument from the Demisto inputs
    indicator_tag = demisto.args().get('indicator')
    # extract the 'CustomFields' dictionary from the 'indicator' tag
    custom_field_tag = indicator_tag.get("CustomFields", {})
    # extract the 'mitreid' field from the 'CustomFields' dictionary
```

dynamic setting script:

```python
import traceback
import demistomock as demisto
from CommonServerPython import *
import requests
from typing import Any, Dict
import json

''' COMMAND FUNCTION '''

# This definition is made for if ChatGPT doesn't give the desired format
output
# and this reformats and returns the input in a way for human readability
def get_info_stripe() -> list[Dict]:
    # Get data from the context with key 'mail.check' and store it as a
string
    data_str = demisto.get(demisto.context(), 'mail.check')
    # Add opening curly braces to the string
    data_str = '{' + data_str
    # Convert string to a dictionary
    data_dict = json.loads(data_str)
    # Get values for 'Confidence', 'Explanation', 'Category', and 'Risk
Score' from the dictionary
    confidence = data_dict.get('Confidence')
    explanation = data_dict.get('Explanation')
    category = data_dict.get('Category')
    risk_score = data_dict.get('Risk Score')
    has_ran = ""
    # Check if 'data_dict' is not None
    if data_dict is not None:
        has_ran = "yes"
    else:
        has_ran = "no"
    results = list()
    result_dict = {
        'check': "mailcheck",
        'has ran': has ran,
        'confidence': confidence,
        'explanation': explanation,
        'category': category,
        'risk_score': risk_score
    }
    # Append the 'result_dict' dictionary to the 'results' list
    results.append(result_dict)
    # Return the 'results' list
    return results

# This definition is if ChatGPT gives the desired formatted output
# and then returns a table with the data nicely categorized
def get_info() -> List[Dict]:
    # Get data from the context with key 'mail.check' and store it as a
dictionary
    data_dict = demisto.get(demisto.context(), 'mail.check')
    # Convert dictionary to a string
    data_str = json.dumps(data_dict)
    # Convert string to a dictionary
    data = json.loads(data_str)
    # Get values for 'Confidence', 'Explanation', 'Category', and 'Risk
Score' from the dictionary
    confidence = data.get('Confidence')
    explanation = data.get('Explanation')
    category = data.get('Category')
```

## xsoar chatgpt:

indicator with chatgpt:

```
import urllib3
import demistomock as demisto
from CommonServerPython import *
import json
from typing import Any, Dict
import requests
import traceback
# Disable insecure warnings
urllib3.disable_warnings()


def chatgpt(tag):
    #value prompt is used to give a prompt to chatgpt and then together
with the discription have chatgpt generate a response
    value prompt = "give me a incident response to an incident indicator
with the following description: "
    data_prompt = tag
    if data_prompt is not None:
        promptstring = value_prompt + data_prompt
    else:
        raise ValueError('No description was found')

    prompt = promptstring[0:2000]

    #response is a variable that gets the data that is returned from the
chatgpt response wich is triggered using the openai-completions command
from openai integration
    response = execute_command("openai-completions", dict(prompt=prompt))
    meta = None
    context = None

    #in this if statement we check for the response and gather it in an
easy format to print
    if response and isinstance(response, dict):
        choices = response.get('choices', [])
        context = [{'text': choice.get('text')} for choice in choices]

    #here we return the data that was gathered in this definition
    return CommandResults(
        readable_output=tableToMarkdown('OpenAI - classifier', context,
meta, removeNull=True),
        outputs_prefix='OpenAI.classifier',
        outputs=context,
    )


def main():
    #here we gather data from the indicator
    indicator_tag = demisto.args().get('indicator')
    #here we take the customfields from the indicator data
    custom_field_tag = indicator_tag.get("CustomFields", {})
    #here we take the custom field description to be used in a definition
    tag = custom field tag.get("description")
    chatgpt_awnser = chatgpt(tag)
    print(chatgpt_awnser)



if __name__ in ("__builtin__", "builtins"):
    main()
```

chatgpt script:

```python
# import required libraries
import demistomock as demisto
from CommonServerPython import *
import requests
import traceback
import json
from typing import Any, Dict

''' STANDALONE FUNCTION '''
# Disable insecure warnings
requests.packages.urllib3.disable_warnings()  # disable warnings about
insecure requests

''' COMMAND FUNCTIONS '''
# define reputations_command function that sends a prompt to and retrieves
a response from OpenAI's GPT-3 API
def reputations_command(args: dict) -> CommandResults:
    """Enter an instruction and watch the OpenAI API respond with a
classifier that attempts to match the context
    or pattern you provided

    :type args: ``dict``
    :param args:  arguments

    :return: CommandResults instance of the OpenAI classifier API response
    :rtype: ``CommandResults``
    """
    # define a default prompt and retrieve the prompt string from args
    value_prompt = "give me a precise score from 0 to 100 on if this email
is risky with 100 being high risk. explain your analysis according to the
parameters. give me a precise score from 0 to 100 on your confidence of
analysis. classify this email from only one of items from the following
list as precisely as possible: CEO Fraud, Phishing, Junk, Other or Social
Engineering. do not use a category which does not exist in the
aforementioned list. The JSON format should be as follows and only as
follows {{{ \"Confidence\":\"Value\", \"Explanation\": \"Value\",
\"Category\": \"Value\", \"Risk Score\":\"Value\" }}}. Here is the email: "
    data_prompt = args.get('prompt', False)
    promptstring = value_prompt + data_prompt
    # truncate prompt string to 2000 characters (GPT-3 API limit)
    prompt = promptstring[0:2000]

    # raise error if prompt not provided
    if not data_prompt:
        raise ValueError('No prompt argument was provided')

    # retrieve API parameters from args or use defaults
    model = args.get('model', 'text-davinci-003')
    temperature = args.get('temperature') or 0.2
    max_tokens = args.get('max_tokens') or 256
    top_p = args.get('top_p') or 1
    frequency_penalty = args.get('frequency_penalty') or 0
    presence_penalty = args.get('presence_penalty') or 0

    # send prompt to OpenAI API and retrieve response
    response = execute_command("openai-completions", dict(prompt=prompt))
    # extract metadata and context information from response if response is
valid
    meta = None
    context = None
    if response and isinstance(response, dict):
```

## python scikit:

dataset_searcher:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load the dataset
data = pd.read_csv('cegeka dataset.csv', usecols=['soarId', 'summary',
'description', 'incidentType', 'verdict'], dtype={'soarId': int})

# Remove rows that contain NaN values
data.dropna(inplace=True)

# Convert the text data to feature vectors
vectorizer = CountVectorizer()
vectors = vectorizer.fit_transform(data[['summary', 'description',
'incidentType']].apply(lambda x: ' '.join(str(x[i]) for i in
range(len(x))), axis=1))

pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)

# Define function to find the most similar incidents
def find_similar_incidents(input_data, num_incidents=10):
    input_vector = vectorizer.transform(input_data[['summary',
'description', 'incidentType']].apply(lambda x: ' '.join(str(x[i]) for i in
range(len(x))), axis=1))
    similarities = cosine_similarity(input_vector, vectors)
    indices = similarities.argsort()[0][::-1][:num_incidents]
    similar_incidents = data.iloc[indices].copy()
    similar_incidents['similarity'] = similarities[0][indices]
    return similar_incidents

# Start an infinite loop to keep asking user for input
while True:
    print('\nEnter new data for finding similar incidents:')
    summary = input('Summary: ')
    description = input('Description: ')
    incident_type = input('Incident type: ')
    input_data = pd.DataFrame({'summary': [summary], 'description':
[description], 'incidentType': [incident_type]})
    input_data.dropna(inplace=True)  # Remove any rows that contain NaN
values
    if summary != "" and description!= "" and incident_type!= "":
        similar_incidents = find_similar_incidents(input_data)
        print('\nSimilar incidents:')
        print('soarId | incidentType | verdict | summary | description')
        print('-------+--------------+---------+---------+-------------')
        for index, row in similar_incidents.iterrows():
            print(f"{row['soarId']} | {row['incidentType']} |
{row['verdict']} | {row['summary']} | {row['description']}")
        if input('Do you want to find similar incidents again? (y/n)
').lower() != 'y':
            break
    else:
        print('\nError invalid input please try again later.')
        break
```

Interactive_ai_with_extra_options:

```
# Import required libraries
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import warnings

# Load dataset
data = pd.read_csv('cegeka dataset.csv', usecols=['summary', 'description',
'incidentType', 'verdict'])

# Drop rows with missing values
data = data.dropna()

# Create a count vectorizer object and fit it on the dataset
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data['summary'] + ' ' + data['description'] +
' ' + data['incidentType'])

# Get the target variable
y = data['verdict']

# Create a logistic regression model object
model = LogisticRegression(max_iter=1000)

# Ignore future warnings
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=FutureWarning)
    # Fit the model on the dataset
    model.fit(X, y)

# Run an infinite loop to keep the model running until stopped
while True:
    # Get input data from the user
    summary = input("Enter a summary: ")
    description = input("Enter a description: ")
    incident_type = input("Enter an incident type: ")
    actual_verdict = input("Enter the actual verdict: ")

    # Create a DataFrame with the input data
    input_data = pd.DataFrame({'summary': [summary], 'description':
[description], 'incidentType': [incident_type], 'verdict':
[actual_verdict]})

    # Vectorize the input data
    X_input = vectorizer.transform(input_data['summary'] + ' ' +
input_data['description'] + ' ' + input_data['incidentType'])

    # Make a prediction
    predicted_verdict = model.predict(X_input)[0]
    print(f"Predicted verdict: {predicted_verdict}")

    # Get the actual verdicts from the dataset
    y_actual = data['verdict']

    # Vectorize all the data
    X_all = vectorizer.transform(data['summary'] + ' ' +
data['description'] + ' ' + data['incidentType'])

    # Split the data into training and test sets
```

# Vragen van het begin van de stage:

## what hebben de analisten nodig voor een beslissing te nemen en hoe kunnen we daarin helpen?

Door de implementatie van automatiseringsoplossingen en het gebruik van ChatGPT kunnen we de beschikbare informatie voor analisten verbeteren, waardoor besluitvormingsprocessen gemakkelijker en beter onderbouwd worden.

## welke machine learning technologieën kunnen samenwerken omtrent het kader van het project?

Het is mogelijk om deze technologieën te integreren, maar ze hebben elk hun eigen sterke punten. Het is aangeraden  om deze technologieën te benutten op plaatsen waar hun specifieke sterke punten het best tot uiting komen.

## welke machine learning technologieën zijn geschikt voor integratie met cegeka playbooks?

Momenteel is python scikit de enige ml technologie die werkt met cegeka playbooks, chatgpt kan mogelijk in de toekomst in ee communicatie playbook worden opgenomen waar de sterkte van chatgpt is. Xsoar ml is naar de toekomst toe meer geschikt om te werken als extra informatie voor de analisten

## welke situatie zijn geschikt voor machine learning integraties?

Momenteel is er voor python scikit en xsoar ml niet genoeg data om op volle kracht te presteren, hoewel we een goede voorspelling kunnen maken van wat er mogelijk is en op basis van deze voorspelling zouden de beste situaties zijn om de data te gebruiken om dezelfde soorten aanvallen en incidenten te automatiseren, terwijl chatgpt zijn sterktes liggen eerder in communicatie

## playbooks gebruiken voor beslissing te maken voor analyst + hoe accuraat?

Playbooks kunnen geen beslissingen maken zonder een ja/nee logica, deze kan voor een deel verplaatst worden met ai maar niet volledig, bijvoorbeeld mijn script voor phishing emails kan vrij accuraat zijn in geval van overduidelijke phishing emails, maar indien er een phishing email komt die gebruik maakt van iets dat nog niet wereldwijd bekent is dan zou chatgpt deze kunnen classificeren als een echte email waar niks mis mee is.