### Code 4: dark\_web\_monitor.py

import requests

from bs4 import BeautifulSoup

import spacy

from rapidfuzz import fuzz

import json

import time

import logging

import matplotlib.pyplot as plt

from collections import Counter

# Configure logging

logging.basicConfig(level=logging.INFO, format='%(message)s')

# Load spaCy English model

nlp = spacy.load("en\_core\_web\_sm")

# Tor SOCKS5 proxy for .onion sites (Tor usually listens on 9050 or 9150)

TOR\_SOCKS\_PROXY = 'socks5h://127.0.0.1:9050'

# Set up requests session with Tor proxy

session = requests.Session()

session.proxies = {

'http': TOR\_SOCKS\_PROXY,

'https': TOR\_SOCKS\_PROXY,

}

# Disable SSL warnings (common with .onion sites self-signed certs)

requests.packages.urllib3.disable\_warnings()

# List of URLs to scrape (mix of clearnet and .onion)

URLS = [

'https://ahmia.fi/',

'http://dreadytofatroptsdj6io7l3xptbet6onoyno2yv7jicoxknyazubrad.onion/',

'http://torlib7fmhyvfv2k7s77xigdds3rosio6k6bxnn256xmtzlbgyizduqd.onion/',

]

# Simplified MITRE ATT&CK keyword dictionary with detailed descriptions

MITRE\_ATTACK\_DICT = {

"dread": {

"technique": "T1071",

"name": "Application Layer Protocol",

"description": "Use of application layer protocols to communicate and potentially evade detection."

},

"carding": {

"technique": "T1537",

"name": "Transfer Data to Cloud Account",

"description": "Transfer of stolen credit card data or other payment info to cloud accounts."

},

"drugs": {

"technique": "T1598",

"name": "Phishing",

"description": "Use of phishing techniques often related to illicit drug trade."

},

"darknet": {

"technique": "T1071",

"name": "Application Layer Protocol",

"description": "Communication over darknet protocols or sites."

},

"market": {

"technique": "T1598",

"name": "Phishing",

"description": "Refers to darknet marketplaces where illicit goods/services are traded."

},

"hacking": {

"technique": "T1204",

"name": "User Execution",

"description": "Execution of code by a user, often social-engineered or malicious payloads."

},

"monero": {

"technique": "T1496",

"name": "Resource Hijacking",

"description": "Cryptocurrency mining or theft using resources without consent."

},

"litecoin": {

"technique": "T1496",

"name": "Resource Hijacking",

"description": "Cryptocurrency mining or theft using resources without consent."

},

"deposit": {

"technique": "T1496",

"name": "Resource Hijacking",

"description": "May refer to cryptocurrency deposits related to malicious activity."

},

"porn": {

"technique": "T1598",

"name": "Phishing",

"description": "Phishing or scam related to illicit content or sites."

},

"leaked": {

"technique": "T1537",

"name": "Transfer Data to Cloud Account",

"description": "Transfer or exposure of leaked data."

},

"darktube": {

"technique": "T1598",

"name": "Phishing",

"description": "Illicit streaming or sharing platforms often related to darknet."

},

"scam": {

"technique": "T1598",

"name": "Phishing",

"description": "Social engineering scams, phishing, fraud."

},

"reddit": {

"technique": "T1071",

"name": "Application Layer Protocol",

"description": "Use of social platforms for communication and attack coordination."

},

"forums": {

"technique": "T1071",

"name": "Application Layer Protocol",

"description": "Use of forums for communication or coordination."

}

}

# Threshold for fuzzy matching confidence

MATCH\_THRESHOLD = 80.0

def fetch\_page(url):

"""Fetch the content of a URL via Tor proxy."""

logging.info(f"[+] Visiting: {url}")

try:

response = session.get(url, timeout=60, verify=False)

response.raise\_for\_status()

return response.text

except Exception as e:

logging.error(f"[ERROR] Could not access {url} → {e}")

return None

def extract\_entities(text):

"""Extract named entities using spaCy."""

doc = nlp(text)

entities = [(ent.text, ent.label\_) for ent in doc.ents]

return entities

def match\_mitre\_keywords(text):

"""Match MITRE ATT&CK keywords in text using fuzzy matching."""

found = []

lowered\_text = text.lower()

for keyword, info in MITRE\_ATTACK\_DICT.items():

# Using simple substring check first for performance

if keyword in lowered\_text:

found.append((keyword, info, 100.0))

continue

# Fuzzy match for approximate matches (optional)

ratio = fuzz.partial\_ratio(keyword, lowered\_text)

if ratio >= MATCH\_THRESHOLD:

found.append((keyword, info, ratio))

return found

def save\_json(filename, data):

with open(filename, 'w') as f:

json.dump(data, f, indent=2)

def generate\_summary\_report(mitre\_matches, total\_entities):

lines = []

lines.append(f"Summary Report\n==============\n")

lines.append(f"Total Entities Extracted: {total\_entities}\n")

if not mitre\_matches:

lines.append("No MITRE ATT&CK keyword matches found.\n")

else:

lines.append("MITRE ATT&CK Keyword Matches:\n")

for keyword, info, score in mitre\_matches:

lines.append(f"- Keyword: {keyword}")

lines.append(f" Technique ID: {info['technique']}")

lines.append(f" Name: {info['name']}")

lines.append(f" Match Confidence: {score:.2f}%")

lines.append(f" Description: {info['description']}\n")

return "\n".join(lines)

def plot\_summary(mitre\_matches):

# Count how many times each technique appears

technique\_counts = Counter([info['technique'] for \_, info, \_ in mitre\_matches])

if not technique\_counts:

logging.info("[!] No matches for visualization.")

return

plt.figure(figsize=(10,6))

plt.bar(technique\_counts.keys(), technique\_counts.values(), color='skyblue')

plt.title("MITRE ATT&CK Techniques Matched")

plt.xlabel("Technique ID")

plt.ylabel("Occurrences")

plt.tight\_layout()

plt.savefig("mitre\_summary.png")

plt.show()

def main():

all\_entities = []

all\_mitre\_matches = []

logging.info("[\*] Starting Dark Web Monitor with enhanced MITRE keyword scanning and explanations...\n")

for url in URLS:

html = fetch\_page(url)

if not html:

continue

soup = BeautifulSoup(html, 'html.parser')

text = soup.get\_text(separator=' ', strip=True)

entities = extract\_entities(text)

all\_entities.extend(entities)

logging.info(f" → Extracted {len(entities)} entities from: {url}")

matches = match\_mitre\_keywords(text)

if matches:

logging.info(f" → Found {len(matches)} MITRE ATT&CK keyword matches:")

for keyword, info, score in matches:

logging.info(f" - {keyword} (Technique: {info['technique']} - {info['name']}) [Score: {score:.2f}]")

all\_mitre\_matches.extend(matches)

else:

logging.info(" → No MITRE ATT&CK keywords found in page text.")

# Be polite to servers and avoid rate limiting

time.sleep(3)

# Save extracted entities

save\_json("extracted\_entities.json", all\_entities)

# Prepare MITRE mappings to save (unique by keyword)

mitre\_output = []

seen = set()

for keyword, info, score in all\_mitre\_matches:

if keyword not in seen:

mitre\_output.append({

"keyword": keyword,

"technique": info["technique"],

"name": info["name"],

"description": info["description"],

"score": score

})

seen.add(keyword)

save\_json("mitre\_mappings.json", mitre\_output)

# Generate and save summary report

summary = generate\_summary\_report(all\_mitre\_matches, len(all\_entities))

with open("summary\_report.txt", "w") as f:

f.write(summary)

logging.info("\n[✔] Done!")

logging.info(f"[✔] Extracted {len(all\_entities)} entities total.")

logging.info(f"[✔] Saved entities to extracted\_entities.json")

logging.info(f"[✔] Saved MITRE mappings to mitre\_mappings.json")

logging.info(f"[✔] Saved summary report to summary\_report.txt")

# Generate visualization if matches found

if all\_mitre\_matches:

plot\_summary(all\_mitre\_matches)

else:

logging.info("[!] No MITRE ATT&CK keyword matches found, skipping visualization.")

if \_\_name\_\_ == "\_\_main\_\_":

main()