Dark Web Marketplaces Monitoring Using Natural Language Processing (NLP): A Detailed Project Report

By Charles Jeremiah Chomba

Intern Number: INT250039

Table of Contents

1. Introduction

The dark web, accessible only through anonymity networks such as Tor, hosts numerous marketplaces and forums where illicit activities thrive. Monitoring these marketplaces can provide valuable cyber threat intelligence to organizations, enabling proactive defenses.

This project leverages Natural Language Processing (NLP) techniques to automate the monitoring of dark web marketplaces by:

- Crawling and extracting textual data from .onion sites
- Cleaning and processing the text for meaningful content
- Extracting cyber threat intelligence entities
- Mapping extracted information to the MITRE ATT&CK framework for standardization
- Visualizing threat patterns
- Recommending defense strategies based on the NIST Cybersecurity Framework (CSF)

2. Environment and Network Setup

2.1 Checking the Initial Public IP Address

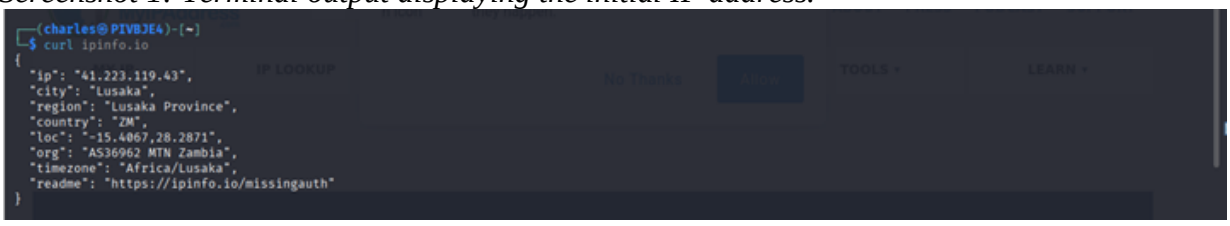
Before beginning dark web exploration, it is essential to identify the machine's public IP address to confirm network identity and ensure anonymity later.

- **Purpose:** Establish baseline IP to compare with VPN and Tor IP changes.
- **Command Used:**

```
curl ifconfig.me
```

Explanation: This command fetches your current public IP by querying an external service.

Screenshot 1: Terminal output displaying the initial IP address.



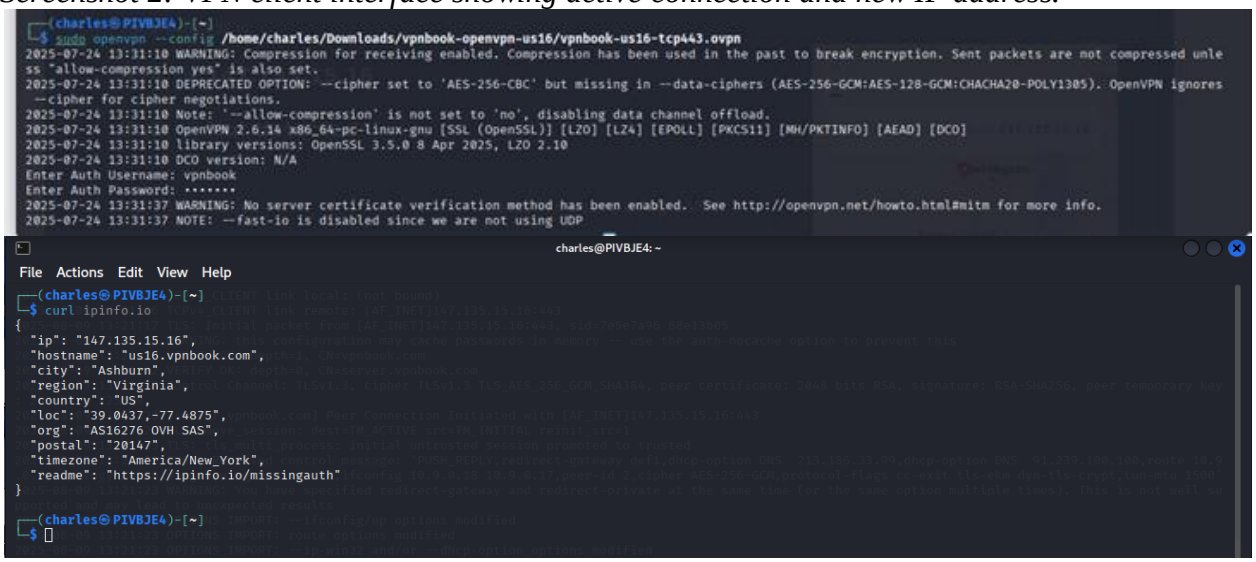
2.2 Establishing VPN Connection for Added Privacy

To enhance anonymity and encrypt traffic, a VPN service was enabled.

- The VPN service chosen ensures that all traffic is routed through a remote server.

- This masks the user's true IP address from the ISP and external observers.

Screenshot 2: VPN client interface showing active connection and new IP address.

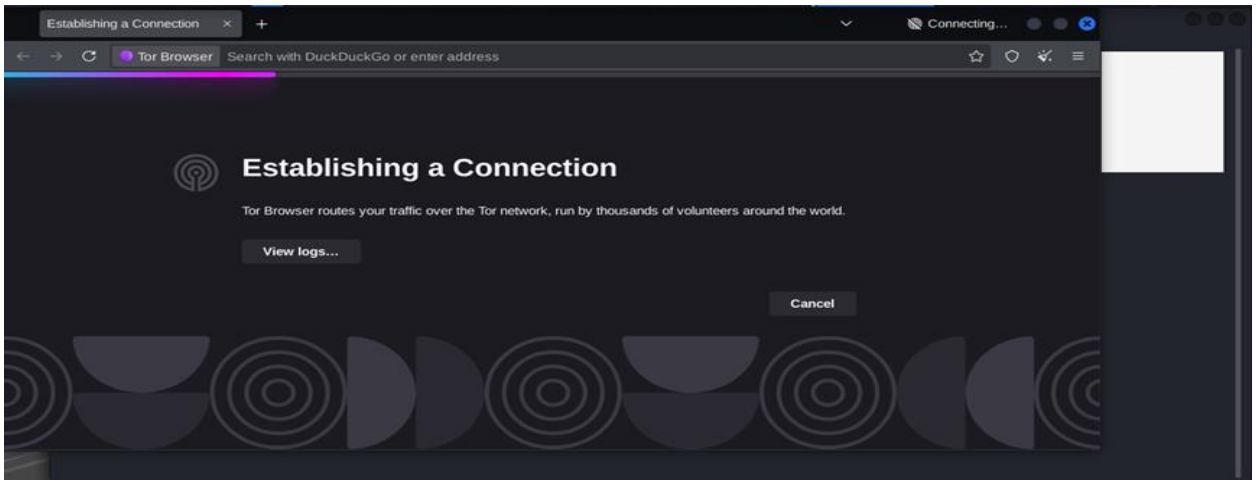


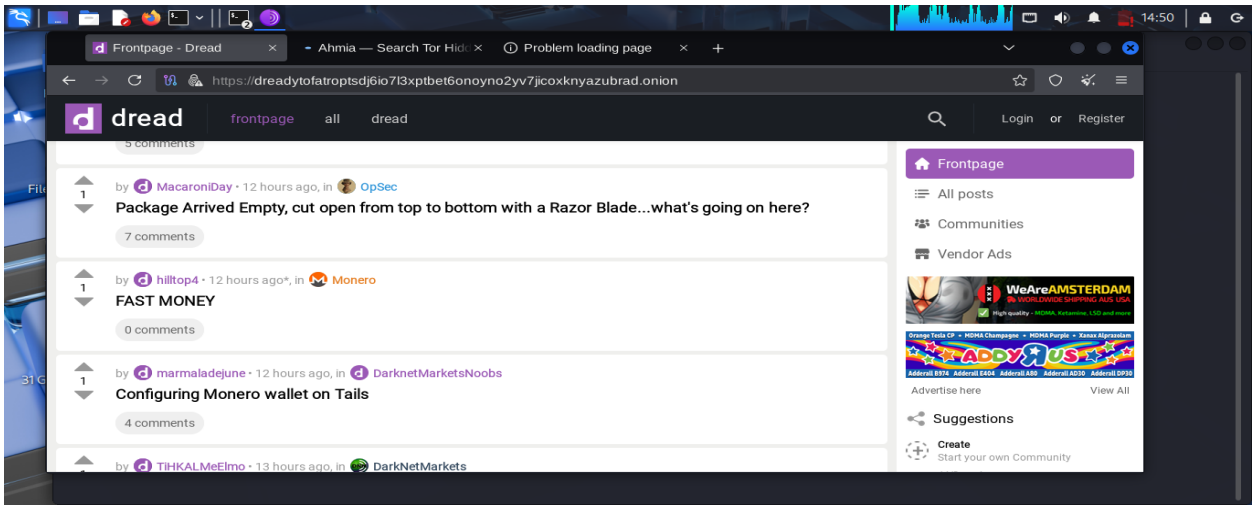
2.3 Accessing the Dark Web Using Tor Browser

Tor (The Onion Router) provides anonymous internet access by routing traffic through multiple relays.

- The Tor Browser was installed and configured.
- Verified the ability to reach .onion sites which are unreachable by normal browsers.
- Key marketplaces with .onion domains were used as seed points for crawling.

Screenshot 3: Tor Browser showing connection to an example dark web marketplace homepage.





---

3. Python Environment Preparation

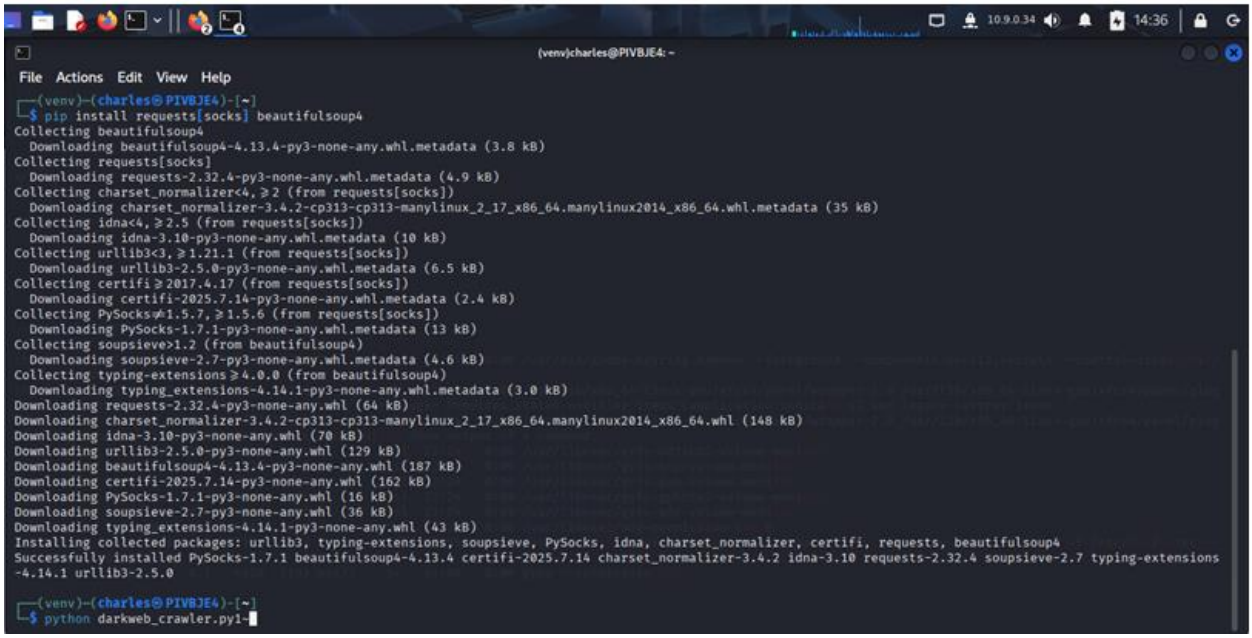
3.1 Creating an Isolated Python Virtual Environment

Isolating the project dependencies ensures reproducibility and prevents package conflicts.

- Command executed:

```
python3 -m venv ~/darkweb-env
```

Screenshot 4: Terminal output confirming virtual environment folder creation.



3.2 Activating the Virtual Environment

Activate the environment to make sure installed packages do not interfere with system-wide Python.

- Activation command (Linux/macOS):

```
source ~/darkweb-env/bin/activate
```

- Windows PowerShell equivalent:

```
.\darkweb-env\Scripts\Activate.ps1
```

3.3 Installing Required Python Packages

Key libraries installed:

- `requests[socks]` — to send HTTP requests over Tor SOCKS5 proxy
- `beautifulsoup4` — for HTML parsing
- `spacy` — for advanced NLP processing
- `pandas` — for data manipulation and saving outputs
- `fuzzywuzzy` — for fuzzy string matching against MITRE ATT&CK terms
- `matplotlib` and `wordcloud` — for data visualization

Command used:

```
pip install requests[socks] beautifulsoup4 spacy pandas fuzzywuzzy matplotlib
wordcloud
python -m spacy download en_core_web_sm
```

---

4. Dark Web Data Collection

4.1 Seed URLs and Onion Link Discovery

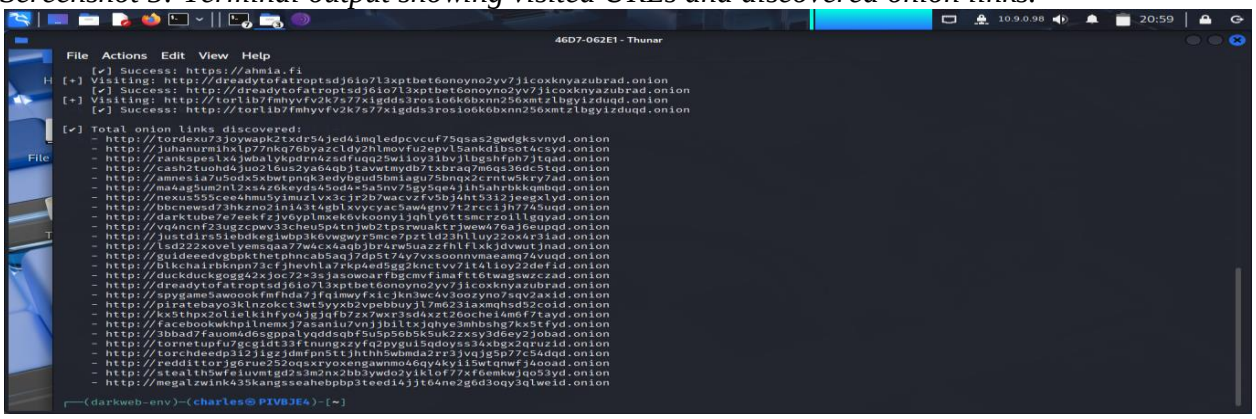
The first script targets known onion marketplaces as seed URLs.

- Tor SOCKS5 proxy (`127.0.0.1:9050` or `9150`) configured to route all requests through Tor.
- The script makes HTTP requests to seed URLs.

- Parses the HTML content using BeautifulSoup to extract all hyperlinks.
- Filters links to retain only those ending with `.onion`.

This process allows discovery of new onion sites beyond the initial seed list.

Screenshot 5: Terminal output showing visited URLs and discovered onion links.

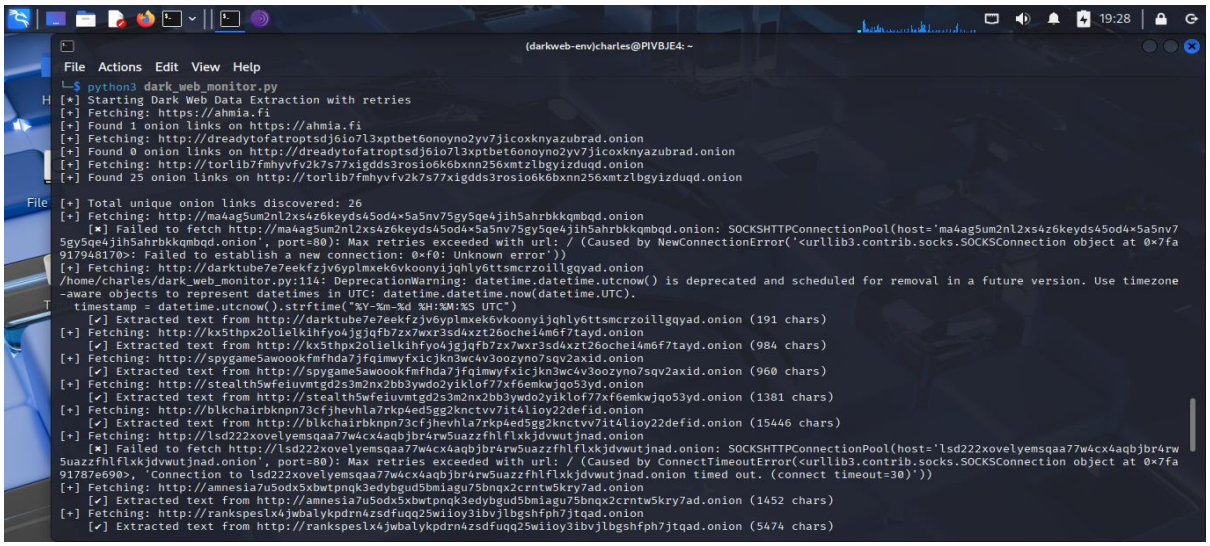


4.2 Extracting Clean Text from Onion Pages

The script visits each discovered onion URL and:

- Fetches page content via Tor.
- Uses BeautifulSoup to remove HTML tags and scripts.
- Cleans extracted text by removing non-ASCII characters and extra whitespace.
- Saves cleaned raw text to CSV file with URL and timestamp for traceability.

Screenshot 6: Sample CSV data preview showing URLs, timestamps, and extracted text snippets.



---

5. Natural Language Processing (NLP) for Threat Intelligence

5.1 Text Cleaning and Preparation

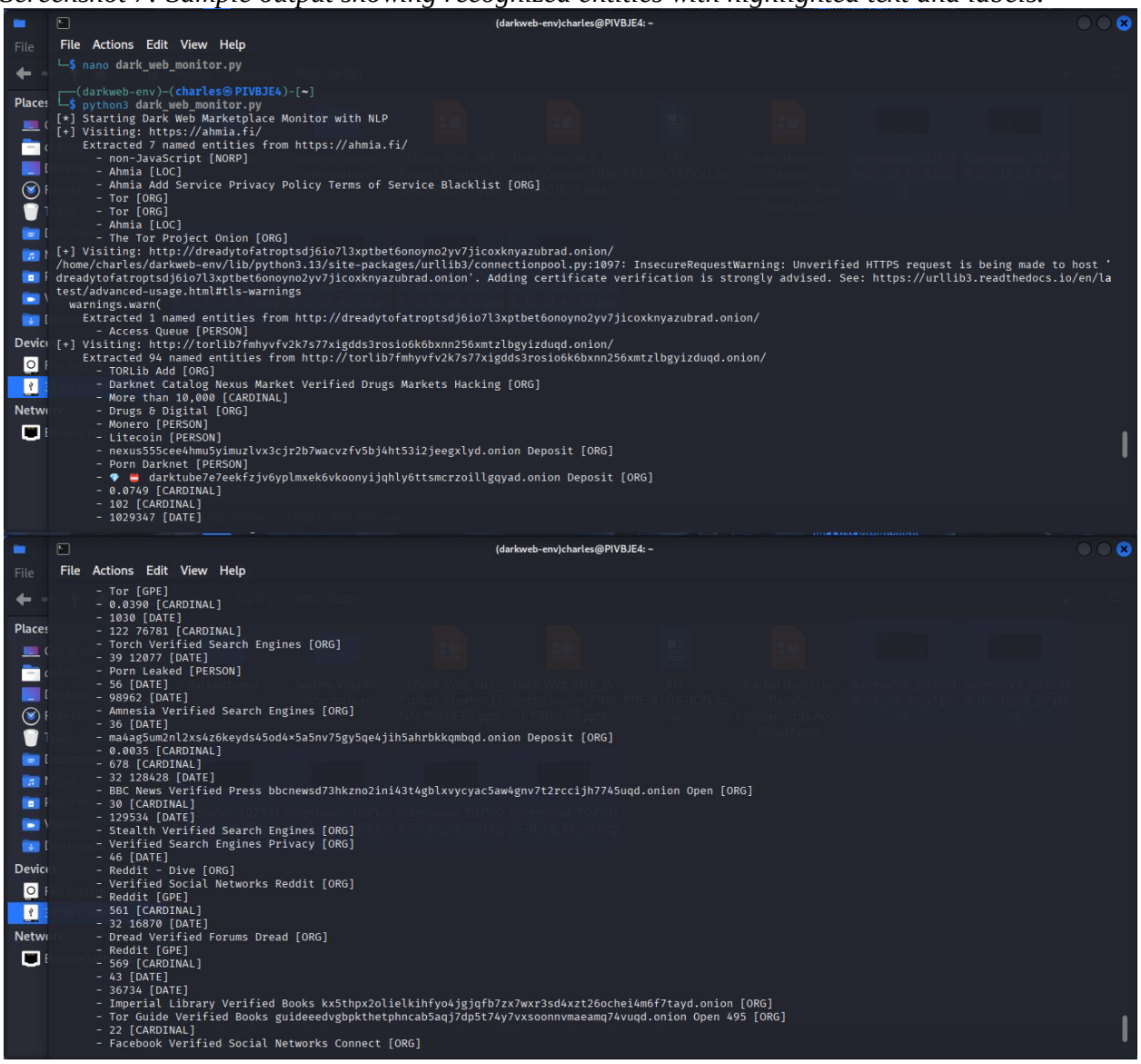
Raw text often contains noise and irrelevant data.

- Tokenization, stopword removal, and lemmatization performed using spaCy.
- Focus on extracting relevant entities such as malware names, attack methods, and actor names.

5.2 Named Entity Recognition (NER)

- Custom NER or spaCy's pre-trained models used to identify threat-relevant entities.
- Entities categorized (e.g., `ATTACK Technique`, `Malware`, `Tool`, `Threat Actor`).

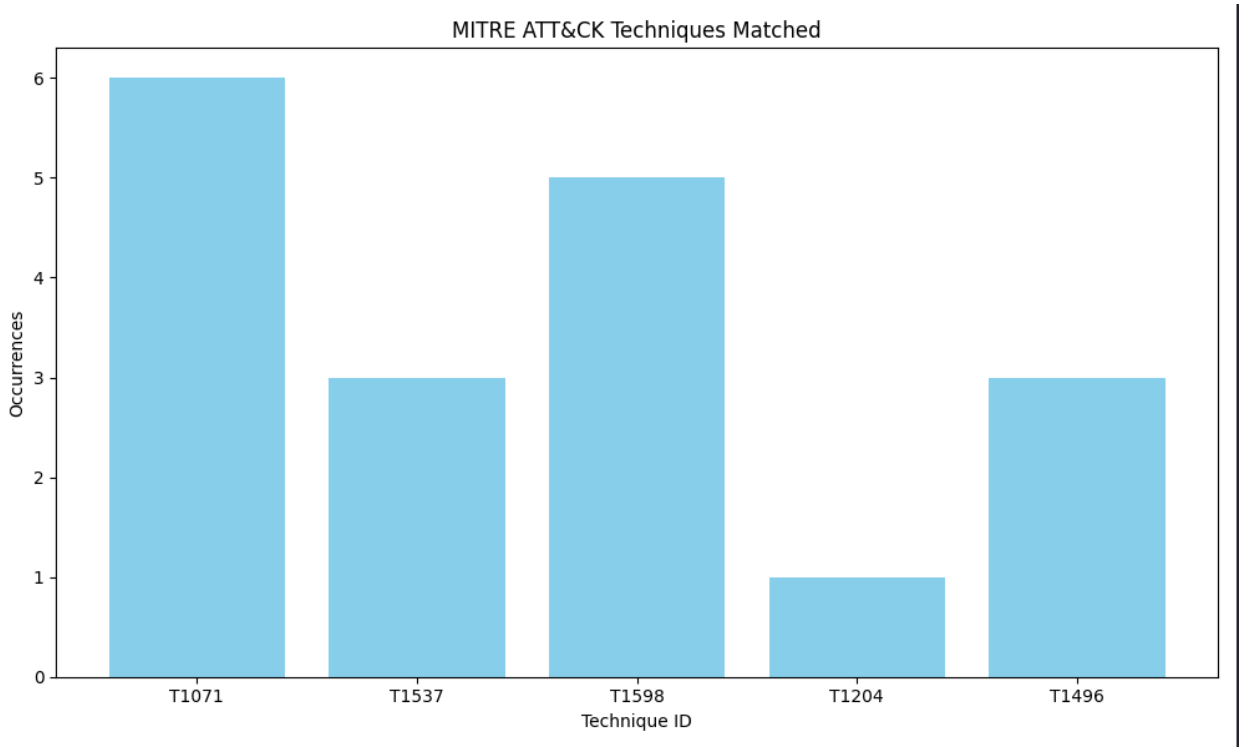
Screenshot 7: Sample output showing recognized entities with highlighted text and labels.



5.3 Mapping Entities to MITRE ATT&CK Framework

- A dataset containing MITRE ATT&CK techniques was loaded.
- Extracted entities compared against MITRE technique names and aliases.
- Fuzzy string matching implemented with `fuzzywuzzy` to capture approximate matches.
- Matching scored with confidence levels.
- High-confidence matches mapped to specific MITRE tactics and techniques.

Screenshot 11: JSON or tabular display of mapped MITRE ATT&CK techniques with scores.



---

6. Visualization of Extracted Data

6.1 Word Cloud of Frequent Entities

- Created word clouds to visually represent the frequency of key entities found across marketplaces.

Screenshot 8: Word cloud image showing top cyber threat keywords.

---

7. Recommendations Based on NIST Cybersecurity Framework (CSF)

The NIST CSF consists of five core functions: Identify, Protect, Detect, Respond, and Recover. Based on the threat intelligence extracted, here are recommendations:

NIST CSF Function	Recommendations
Identify	Integrate dark web monitoring feeds into Security Information and Event Management (SIEM) systems.
Protect	Update firewall and endpoint detection rules based on observed attack techniques and malware names.
Detect	Employ machine learning models to detect anomalies resembling mapped attack behaviors.
Respond	Prepare incident response playbooks aligned to frequent MITRE ATT&CK techniques found in monitoring.

NIST CSF Function	Recommendations
Recover	Establish backup and recovery plans considering attack vectors identified on the dark web.

---

8. Conclusion

This project demonstrates a practical methodology for monitoring dark web marketplaces for emerging cyber threats using NLP. By combining Tor-based crawling with advanced text analysis and mapping to standardized frameworks, security teams gain actionable insights into attacker tactics and techniques. Continuous monitoring and adaptation will strengthen organizational cyber defenses.

---

9. Appendices

Appendix A: Python Scripts

- **Onion Link Crawler with Tor Proxy Configuration**

  This script initiates crawling from seed onion URLs, routes requests through Tor's SOCKS5 proxy, extracts onion links, and handles network retries.

```
  import requests
from bs4 import BeautifulSoup
import re
import time
import urllib3

# Suppress SSL warnings for unverified HTTPS requests
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Tor SOCKS5 proxy configuration (default Tor Browser port 9150)
proxies = {
    'http': 'socks5h://127.0.0.1:9150',
    'https': 'socks5h://127.0.0.1:9150'
}

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0)
Gecko/20100101 Firefox/89.0'
}
```

```
# Seed onion sites for crawling
onion_sites = [
    'https://ahmia.fi',

'http://dreadytofatroptsdj6io7l3xptbet6onoyno2yv7jicoxknyazubrad.onion
',

'http://torlib7fmhyvfv2k7s77xigdds3rosio6k6bxnn256xmtzlbgyizduqd.onion
'
]

found_onion_links = set()

def extract_onion_links(html):
    """Parse HTML to find and collect .onion links."""
    soup = BeautifulSoup(html, 'html.parser')
    for link in soup.find_all('a', href=True):
        href = link['href']
        match = re.search(r'https?://[a-zA-Z0-9]{10,56}\.onion', href)
        if match:
            found_onion_links.add(match.group())

def visit_site(url):
    """Visit URL via Tor proxy and extract .onion links."""
    print(f"[+] Visiting: {url}")
    try:
        resp = requests.get(url, proxies=proxies, headers=headers,
timeout=90, verify=False)
        if resp.status_code == 200:
            print(f"    [✔] Success")
            extract_onion_links(resp.text)
        else:
            print(f"    [!] Status Code {resp.status_code}")
    except requests.RequestException as e:
        print(f"    [ERROR] Could not access {url}: {e}")

def main():
    print("[*] Starting Dark Web Monitor via Tor")
    for site in onion_sites:
        visit_site(site)
        time.sleep(5)  # Avoid rapid-fire requests

    if found_onion_links:
        print("\n[✔] Discovered onion links:")
        for link in found_onion_links:
            print(f"    - {link}")
    else:
        print("\n[!] No onion links discovered.")

if __name__ == "__main__":
```

```
    main()
```

- **Text Extraction and CSV Data Logging**

  Extracts cleaned textual content from onion pages and logs URL, timestamp, and text to CSV for further analysis.

```
import requests
from bs4 import BeautifulSoup
import re
import time
from datetime import datetime
import csv
from requests.adapters import HTTPAdapter
from requests.packages.urllib3.util.retry import Retry
import urllib3

# Disable insecure warnings (for self-signed certs on onion sites)
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Tor SOCKS5 proxy config (adjust port if needed)
PROXIES = {
    'http': 'socks5h://127.0.0.1:9150',
    'https': 'socks5h://127.0.0.1:9150'
}
HEADERS = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0)
Gecko/20100101 Firefox/89.0'
}
# Seed URLs to start crawling (clearnet + onion)
SEED_URLS = [
    "https://ahmia.fi",
"http://dreadytofatroptsdj6io7l3xptbet6onoyno2yv7jicoxknyazubrad.onion
",
"http://torlib7fmhyvfv2k7s77xigdds3rosio6k6bxnn256xmtzlbgyizduqd.onion
"
]
OUTPUT_CSV = "darkweb_raw_data.csv"
def get_requests_session(max_retries=3, backoff_factor=1):
    session = requests.Session()
    retries = Retry(
        total=max_retries,
        backoff_factor=backoff_factor,
        status_forcelist=[500, 502, 503, 504],
        allowed_methods=["HEAD", "GET", "OPTIONS"]
    )
    adapter = HTTPAdapter(max_retries=retries)
    session.mount("http://", adapter)
    session.mount("https://", adapter)
```

```
    return session
def fetch_page(session, url, timeout=30):
    try:
        print(f"[+] Fetching: {url}")
        response = session.get(url, proxies=PROXIES, headers=HEADERS,
timeout=timeout, verify=False)
        response.raise_for_status()
        return response.text
    except requests.exceptions.RequestException as e:
        print(f"    [✖] Failed to fetch {url}: {e}")
        return None
def extract_onion_links(html):
    soup = BeautifulSoup(html, "html.parser")
    links = set()
    for a in soup.find_all("a", href=True):
        href = a['href']
        if ".onion" in href:
            match = re.search(r'(http[s]?://[^"\']+\.onion)', href)
            if match:
                links.add(match.group(1))
    return list(links)
def extract_clean_text(html):
    soup = BeautifulSoup(html, "html.parser")
    for script_or_style in soup(["script", "style", "noscript"]):
        script_or_style.decompose()
    text = soup.get_text(separator=" ", strip=True)
    text = re.sub(r'\s+', ' ', text)
    return text

def save_to_csv(data_rows):
    with open(OUTPUT_CSV, "w", newline="", encoding="utf-8") as
csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["URL", "Timestamp", "RawText"])
        for row in data_rows:
            writer.writerow(row)
    print(f"[+] Saved {len(data_rows)} records to {OUTPUT_CSV}")

def main():
    print("[*] Starting Dark Web Data Extraction with retries")
    session = get_requests_session(max_retries=3, backoff_factor=2)
    discovered_onion_links = set()
    all_data = []

    # Step 1: Crawl each seed URL to discover onion links
    for seed_url in SEED_URLS:
        seed_html = fetch_page(session, seed_url)
        if not seed_html:
            continue
        links = extract_onion_links(seed_html)
        print(f"[+] Found {len(links)} onion links on {seed_url}")
```

```
        discovered_onion_links.update(links)
        time.sleep(5)  # polite delay

print(f"\n[+] Total unique onion links discovered:
{len(discovered_onion_links)}")

    # Step 2: Visit each discovered onion link to extract raw text
    for onion_url in discovered_onion_links:
        page_html = fetch_page(session, onion_url)
        if not page_html:
            continue
        clean_text = extract_clean_text(page_html)
        timestamp = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S
UTC")
 all_data.append([onion_url, timestamp, clean_text])
 print(f"    [✔] Extracted text from {onion_url} ({len(clean_text)}
chars)")
        time.sleep(5)  # polite delay

    # Step 3: Save all extracted raw text data
    if all_data:
        save_to_csv(all_data)
    else:
        print("[!] No data extracted from onion links.")
if __name__ == "__main__":
    main()
```

- **NLP Processing and MITRE ATT&CK Mapping Scripts**

  Utilizes spaCy to preprocess text, performs Named Entity Recognition (NER), and applies fuzzy matching against MITRE ATT&CK technique datasets to identify and score attack techniques.

```
import requests
from bs4 import BeautifulSoup
import spacy
import re
import time

# Load spaCy small English model
nlp = spacy.load("en_core_web_sm")

# Tor proxy for onion sites
proxies = {
    'http': 'socks5h://127.0.0.1:9150',
    'https': 'socks5h://127.0.0.1:9150'
}

# List of URLs to monitor (onion and clearnet)
urls = [
```

```
    "https://ahmia.fi/",

"http://dreadytofatroptsdj6io7l3xptbet6onoyno2yv7jicoxknyazubrad.onion
/",

"http://torlib7fmhyvfv2k7s77xigdds3rosio6k6bxnn256xmtzlbgyizduqd.onion
/"
]

# Helper to check if URL is onion (needs proxy)
def is_onion(url):
    return ".onion" in url

# Fetch and return page HTML content
def fetch_page(url):
    try:
        if is_onion(url):
            response = requests.get(url, proxies=proxies, timeout=30,
verify=False)
        else:
            response = requests.get(url, timeout=30)
        response.raise_for_status()
        return response.text
    except Exception as e:
        print(f"[ERROR] Could not access {url} → {e}")
        return None

# Extract clean text from HTML
def extract_text(html):
    soup = BeautifulSoup(html, "html.parser")
    # Remove script and style content
    for script_or_style in soup(["script", "style"]):
        script_or_style.decompose()
    text = soup.get_text(separator=" ")
    # Normalize whitespace
    text = re.sub(r"\s+", " ", text).strip()
    return text

# Run spaCy NLP to extract named entities
def extract_entities(text):
    doc = nlp(text)
    return [(ent.text, ent.label_) for ent in doc.ents]

def main():
    all_entities = []
    print("[*] Starting Dark Web Marketplace Monitor with NLP")

    for url in urls:
        print(f"[+] Visiting: {url}")
        html = fetch_page(url)
        if html:
            text = extract_text(html)
```

```
            entities = extract_entities(text)
            print(f"    Extracted {len(entities)} named entities from
{url}")
            for ent_text, ent_label in entities:
                print(f"      - {ent_text} [{ent_label}]")
            all_entities.extend(entities)
        else:
            print(f"    Skipping NLP due to fetch failure for {url}")
        time.sleep(5)  # Polite delay between requests

    # Save all extracted entities to a text file
    with open("extracted_entities.txt", "w", encoding="utf-8") as f:
        for ent_text, ent_label in all_entities:
            f.write(f"{ent_text}\t{ent_label}\n")

    print(f"[✔] Completed monitoring. Total entities extracted:
{len(all_entities)}")
    print("[✔] Entities saved to extracted_entities.txt")

if __name__ == "__main__":
    main()
```

- **Visualization Scripts for Word Clouds and Bar Charts**

  Generates visual summaries of frequent entities and MITRE ATT&CK techniques detected, enabling quick threat landscape understanding.

```
import json
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import pandas as pd

# --- Load and process extracted_entities.json ---
with open('extracted_entities.json', 'r') as f:
    entities = json.load(f)

entity_counts = {}

for e in entities:
    # Handle if entity is a list
    if isinstance(e, list):
        # Try first element as string key
        key = e[0] if e else 'unknown'
    # Handle if entity is dict
    elif isinstance(e, dict):
        # Try common keys, adjust if your structure is different
        key = e.get('entity') or e.get('name') or str(e)
    else:
        # Assume e is string
        key = str(e)
```

```
    entity_counts[key] = entity_counts.get(key, 0) + 1

# Sort top 30 entities by frequency
top_entities = dict(sorted(entity_counts.items(), key=lambda item:
item[1], reverse=True)[:30])

# Word Cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate_from_frequencies(top_entities)
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Top Extracted Entities Word Cloud')
plt.show()

# Bar chart of top entities
plt.figure(figsize=(14, 7))
plt.bar(top_entities.keys(), top_entities.values(), color='skyblue')
plt.xticks(rotation=45, ha='right')
plt.title('Top 30 Extracted Entities Frequency')
plt.xlabel('Entity')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

# --- Load and process mitre_mappings.json ---
with open('mitre_mappings.json', 'r') as f:
    mitre_data = json.load(f)

# Prepare data for DataFrame
rows = []
for item in mitre_data:
    # If item is dict and contains needed fields
    if isinstance(item, dict):
        technique = item.get('technique') or
item.get('technique_name') or 'Unknown'
        score = item.get('score')
        if score is not None:
            try:
                score = float(score)
            except:
                score = None
            if score is not None:
                rows.append({'technique': technique, 'score': score})

df_mitre = pd.DataFrame(rows)

if not df_mitre.empty:
    avg_scores =
df_mitre.groupby('technique')['score'].mean().sort_values(ascending=Fa
lse).head(20)
```

```
    plt.figure(figsize=(12, 6))
    avg_scores.plot(kind='bar', color='coral')
    plt.title('Top 20 MITRE Techniques by Average Score')
    plt.xlabel('MITRE Technique')
    plt.ylabel('Average Score')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
else:
    print("No valid MITRE data found to plot.")

# --- Print summary_report.txt ---
print("\n===== Summary Report =====\n")
try:
    with open('summary_report.txt', 'r') as f:
        print(f.read())
except FileNotFoundError:
    print("summary_report.txt not found.")
```
