Homographic (Homoglyph) Detector

📝Objective & Deep Background

The Homoglyph attack, also known as an IDN homograph attack, exploits the visual similarity of Unicode characters to trick users into visiting spoofed domains. For instance, a user might see **example.com**, while the actual link could contain Cyrillic characters like **е** or **а**, which look identical but are distinct code points WikipediaInspiroz. Such deceptive domains can be used in phishing, malware delivery, or impersonation of trusted brands Mesh | Email Security Redefined for MSPsWikipedia.

These attacks are subtle, often invisible to the naked eye and standard filters, and can result in stolen credentials, reputational damage, and financial loss Sven RuppertMDPI. Organizations like educational institutions are especially vulnerable given their reliance on email and digital workflows Inspiroz**.**

Defense Strategies & Research Insights

- **Unicode Normalization (e.g., NFKC)**: Converts visually similar characters to canonical forms, reducing spoofing risk Sven RuppertStack Overflow.
- **Homoglyph Mapping Databases**: Tools like Unicode's *confusables.txt* list mappings of visually similar characters for detection IBM TechXchange Community.
- **Machine Learning & Image-Based Models**:
    - **GlyphNet** uses image rendering and CNNs to detect homoglyph domains with high accuracy (AUC ≈ 0.93) arXiv.
    - **Siamese Neural Networks** compare visual similarity at the image level for robust detection arXiv.
    - **PhishGAN** synthesizes homoglyph variations to train detection models with augmented datasets arXiv.
- **Automated Scanning Tools**: Tools like *ShamFinder* can auto-generate homoglyph databases and help detect IDN homographs at scale arXiv.

These advanced techniques offer powerful alternatives to simple normalization or string comparison, particularly in high-stakes environments like enterprise security.

🛠️ Tools & Libraries Used

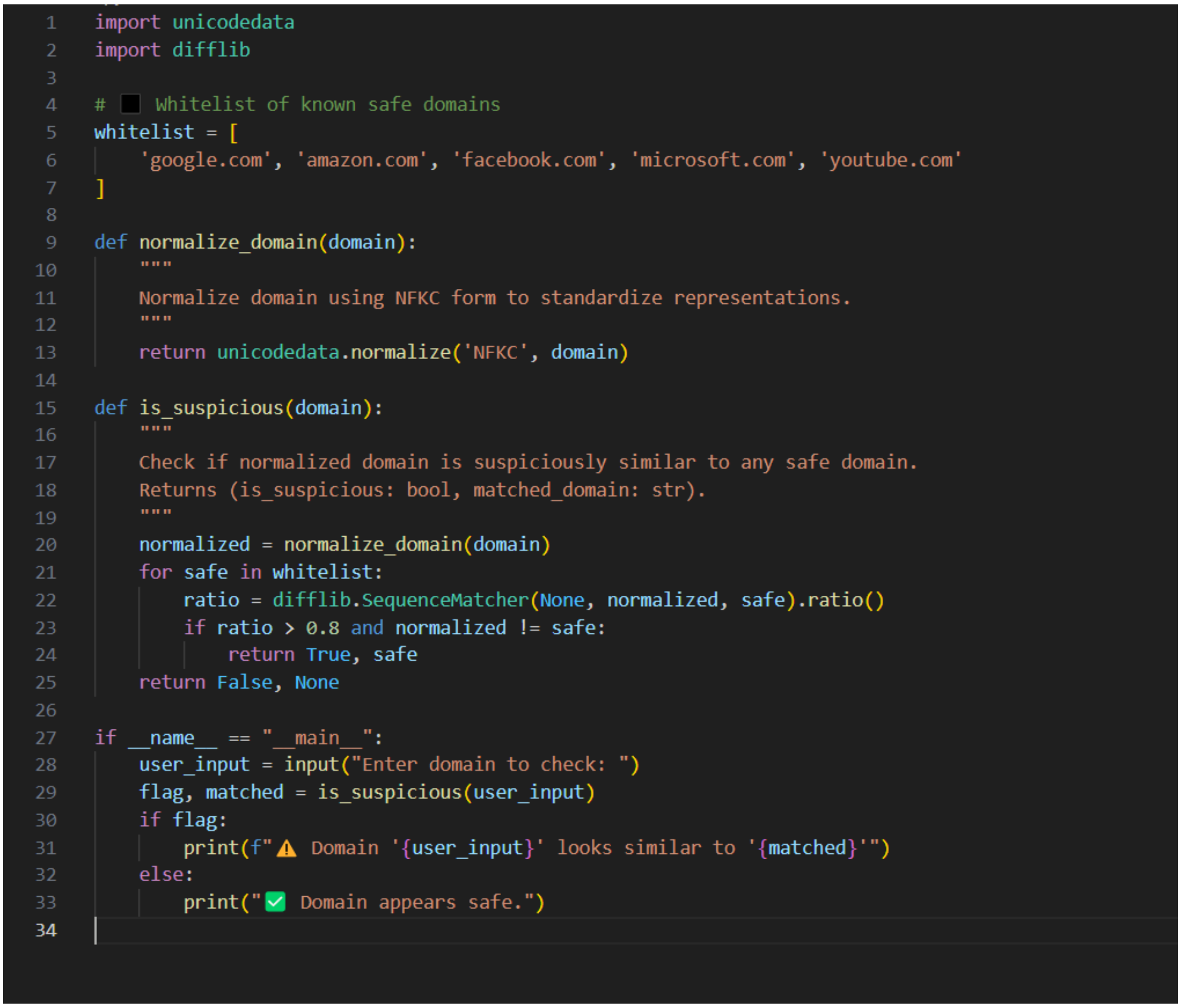
- **Python** (Programming Language)
- unicodedata (Unicode normalization)
- difflib (String comparison)

- re (Regular expression module)
- idna (For domain name handling in Unicode)
- **Top domain whitelist** (e.g., google.com, amazon.com)

📁 Unicode Homoglyph Examples

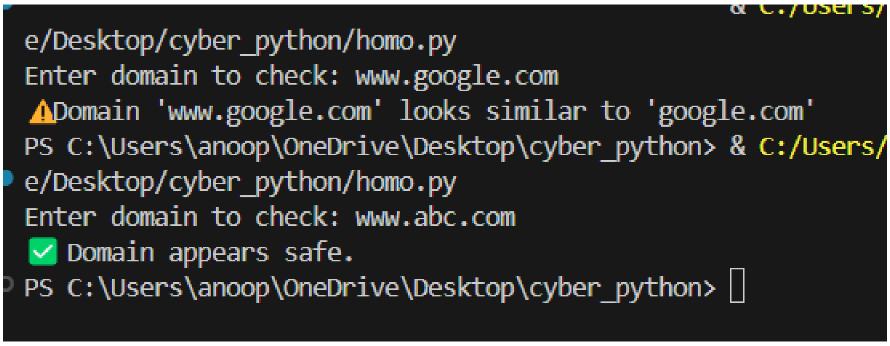
Fake Character	Unicode	Looks Like	Legitimate Character
ɡ	U+0261	g	g
ο	U+03BF	o	o
с	U+0441	c	c
а	U+0430	a	a
е	U+0435	e	e

Expanded Code & Explanation



```python
import unicodedata
import difflib

# ■ Whitelist of known safe domains
whitelist = [
    'google.com', 'amazon.com', 'facebook.com', 'microsoft.com', 'youtube.com'
]

def normalize_domain(domain):
    """
    Normalize domain using NFKC form to standardize representations.
    """
    return unicodedata.normalize('NFKC', domain)

def is_suspicious(domain):
    """
    Check if normalized domain is suspiciously similar to any safe domain.
    Returns (is_suspicious: bool, matched_domain: str).
    """
    normalized = normalize_domain(domain)
    for safe in whitelist:
        ratio = difflib.SequenceMatcher(None, normalized, safe).ratio()
        if ratio > 0.8 and normalized != safe:
            return True, safe
    return False, None

if __name__ == "__main__":
    user_input = input("Enter domain to check: ")
    flag, matched = is_suspicious(user_input)
    if flag:
        print(f"⚠ Domain '{user_input}' looks similar to '{matched}'")
    else:
        print("✅ Domain appears safe.")

```

Output:



```
                                                             & C:/Users/
e/Desktop/cyber_python/homo.py
Enter domain to check: www.google.com
⚠Domain 'www.google.com' looks similar to 'google.com'
PS C:\Users\anoop\OneDrive\Desktop\cyber_python> & C:/Users/
e/Desktop/cyber_python/homo.py
Enter domain to check: www.abc.com
✅ Domain appears safe.
PS C:\Users\anoop\OneDrive\Desktop\cyber_python> ▯
```

How It Works:

- First, normalize using Unicode NFKC to unify different code points.
- Then use string similarity (e.g., difflib) to compare with whitelisted domains and flag close matches.

🛡️ What You Learned

🔸 How attackers manipulate Unicode for phishing
🔸 Python tools for domain normalization and comparison
🔸 Creating simple defensive tools for domain analysis
🔸 Importance of whitelisting and string matching techniques

✅ Conclusion

This task introduced practical cybersecurity concerns through homoglyph-based attacks. Using Python and Unicode normalization, a basic but effective detector was implemented to flag misleading URLs. The approach can be extended with machine learning, domain reputation services, or browser plugins for more advanced threat detection.

Name: Anoop Shivadas

Inter Id: 129