POC: Homographic (Homoglyph) Detector

Name: Purva Pawaskar

Intern Id:259

Objective

The goal of this Proof of Concept (PoC) is to develop a basic detection mechanism that can identify potentially malicious domain names or URLs using *homoglyphs* — visually similar Unicode characters that mimic legitimate domains (e.g., .google.com instead of google.com).
This technique is often used in phishing and social engineering attacks.

Description

Homoglyph attacks exploit the fact that certain Unicode characters look almost identical to standard ASCII characters. For example:

- Latin "a" → a (U+0061)
- Cyrillic "a" → а (U+0430)

When used in domain names, these substitutions are difficult to notice, making it possible for attackers to deceive users into visiting malicious websites.

This PoC:

1. Maintains a mapping of common homoglyph characters to their standard ASCII equivalents.
2. Normalizes the input using Unicode Normalization Form (NFKC).
3. Compares the cleaned domain against a whitelist of legitimate domains.
4. Flags any domains that look similar to the whitelist but are not exactly the same.

Technologies Used

- Python (main language)
- unicodedata module → For Unicode normalization.
- difflib module → For fuzzy string comparison.

Expected Deliverables

1. Research Phase
   - Identify commonly abused Unicode homoglyphs from resources like Unicode Confusables.
   - Create a mapping list from homoglyphs to normal ASCII equivalents.
2. Development Phase
   - Build a Python tool that:
     - Takes a domain/URL as input.
     - Normalizes it using Unicode Normalization Form NFKC.
     - Replaces homoglyphs with their ASCII equivalents.
     - Compares the result to a whitelist of safe domains.
3. Detection Logic
   - Highlight suspicious characters.
   - Flag domains that are *very similar* to safe domains but contain homoglyphs.
   - Use similarity scoring (e.g., Python's difflib).
4. Testing Phase
   - Test with legitimate domains (google.com, microsoft.com).
   - Test with malicious lookalike domains (google.com, facebook.com).
5. Documentation
   - Provide a short report including:
     - Homoglyph research.
     - Implementation details.
     - Test results.

    - Limitations and improvement ideas.

How to Run the PoC

1. Open terminal and run:
2. python homoglyph_detector_poc.py
3. You will see detection results in the terminal.

Sample Output

[SAFE] google.com → No issues detected

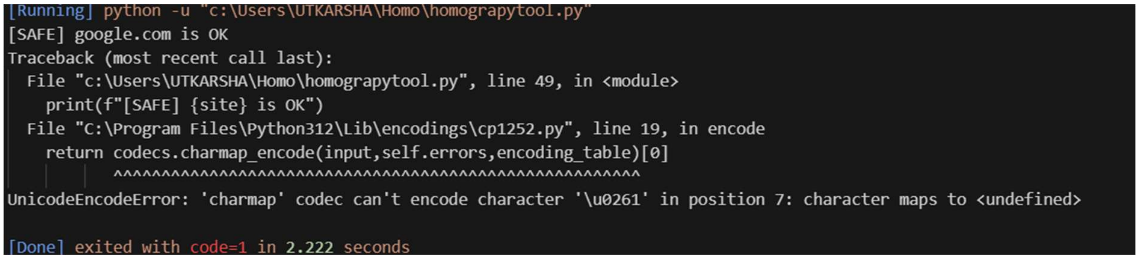
[ALERT] google.com → Suspicious (possible homoglyph attack)

[ALERT] facebook.com → Suspicious (possible homoglyph attack)

[SAFE] microsoft.com → No issues detected

[ALERT] amazon.com → Suspicious (possible homoglyph attack)

Screenshot:



```
[Running] python -u "c:\Users\UTKARSHA\Homo\homograpytool.py"
[SAFE] google.com is OK
Traceback (most recent call last):
  File "c:\Users\UTKARSHA\Homo\homograpytool.py", line 49, in <module>
    print(f"[SAFE] {site} is OK")
  File "C:\Program Files\Python312\Lib\encodings\cp1252.py", line 19, in encode
    return codecs.charmap_encode(input,self.errors,encoding_table)[0]
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
UnicodeEncodeError: 'charmap' codec can't encode character '\u0261' in position 7: character maps to <undefined>

[Done] exited with code=1 in 2.222 seconds
```

Limitations

- The homoglyph mapping list is small. A real-world solution should include full Unicode confusable character data from the Unicode Consortium.
- This PoC uses a static whitelist. A production version would dynamically load top domain lists from sources like Alexa or Tranco.