Proof of Concept – Homograph Attacks Detection

Name : Sandhya Singh

Intern ID : 233

1 . Definition

Homograph attack detection is the process of analyzing text (especially URLs, domain names, usernames, or file names) to identify characters that look visually similar to legitimate ones but are actually different Unicode code points. These characters are used by attackers to create spoofed identities, tricking users into visiting malicious websites or trusting fake applications.

- **Real :** facebook.com
- **Fake :** faceboTk.com (Here, the attacker uses **Cyrillic capital letter "I"** (U+0406) instead of the Latin "I".

2. Approach: How the Detection Works:

The detection system scans URLs or domain names character by character to identify **suspicious characters**.

1) The script checks:

- If characters are **not part of standard ASCII** (A-Z, a-z, 0-9, punctuation).
- If they are **Unicode characters that look deceptive**.
- If they are **invisible** or **non-printable traps**.

2) Detection Process:

- **Extract domain** from the full URL (e.g., from http://facebook.com/page, extract facebook.com).
- **Scan each character** in the domain.
- **Identify suspicious characters** that are not standard or safe

3 . Logic Behind the Detection Script (homograph_core.py)

This section explains the **algorithm** used in the detection script:

Step-by-Step Logic:

- **Define Safe Characters**:
    - Standard ASCII: A-Z, a-z, 0–9, punctuation marks.
    - Safe invisible characters like \n (newline), \t (tab) are allowed.
- **Scan Each Character** in the domain:
    - If it's **not safe**, fetch:
        - Its **Unicode name** (e.g., "CYRILLIC CAPITAL LETTER I")
        - Its **code point** (e.g., U+0406)
- **Extract Domain** from URL:
    - From https://facebook.com/profile, it extracts facebook.com for analysis

4 : Explaination of the code :

- **Imports :**

```
import unicodedata

import string

from urllib.parse import urlparse
```

unicodedata: Used to get the Unicode name of characters.

Provides predefined sets of characters (letters, digits, etc.).

Extracts parts of a URL (like domain name).

- **Safe Character Sets :**

```
standard_chars = set(string.ascii_letters + string.digits + string.punctuation + " ")
safe_invisible_chars = {'\n', '\r', '\t'}
allowed_chars = standard_chars.union(safe_invisible_chars)
```

- These are characters considered **safe/standard** (ASCII).
- Everything outside of this set will be treated as **suspicious**.

- **Suspicious Character Check :**

```
def is_suspicious(ch):
    return ch not in allowed_chars
```

- Checks if a character is not in the allowed list.

- **Scan Suspicious Characters :**

```
def scan_text(text):
    suspicious_chars = filter(is_suspicious, text)

    def get_char_info(ch):
        try:
            name = unicodedata.name(ch)
        except ValueError:
            name = "Unknown or Non-character"
        codepoint = f"U+{ord(ch):04X}"
        return (ch, name, codepoint)

    return list(map(get_char_info, suspicious_chars))
```

  - Scans a string for suspicious characters and returns:
    - The **character**
    - Its **Unicode name**
    - Its **Unicode codepoint**

- **Extract Domain from URL :**

```
def scan_domain(domain_or_url):

    domain = extract_domain(domain_or_url)

    return scan_text(domain)
```

  - **Combines all the above:**
    - **Extracts domain**
    - **Scans for suspicious characters**

- **Example Input and Output :**

```
print(scan_domain("https://www.facebook.com"))
```

5 : Conclusion

The project successfully shows that **even simple character-level analysis** can help in **detecting Unicode-based homograph attacks**.