Name:Kirti Koltharkar Inter ID:246

Homographic (Homoglyph) Detector — Weekly Report

Objective:

- ✓ Homoglyph mapping (homoglyph_map dictionary)
- ✓ Maps visually similar Unicode characters (e.g., А (Cyrillic A) to a, і (Ukrainian i) to i) to their ASCII equivalents.
- ✓ This helps normalize suspicious domains into a standard form.
- ✓ sanitize_domain(domain)
- ✓ Replaces homoglyph characters in a domain with their mapped ASCII versions.
- ✓ Example: "gοοgle.com" (with Cyrillic 'o') becomes "google.com".
- ✓ calculate_similarity(a, b)
- ✓ Uses difflib.SequenceMatcher to calculate how similar two domain strings are (returns a score from 0 to 1).
- ✓ load_file_lines(filepath)
- ✓ Reads a list of domains from a file (used for both safe whitelist and test domains).
- ✓ detect_homoglyph_domains(safe_domains, test_domains, output_path)

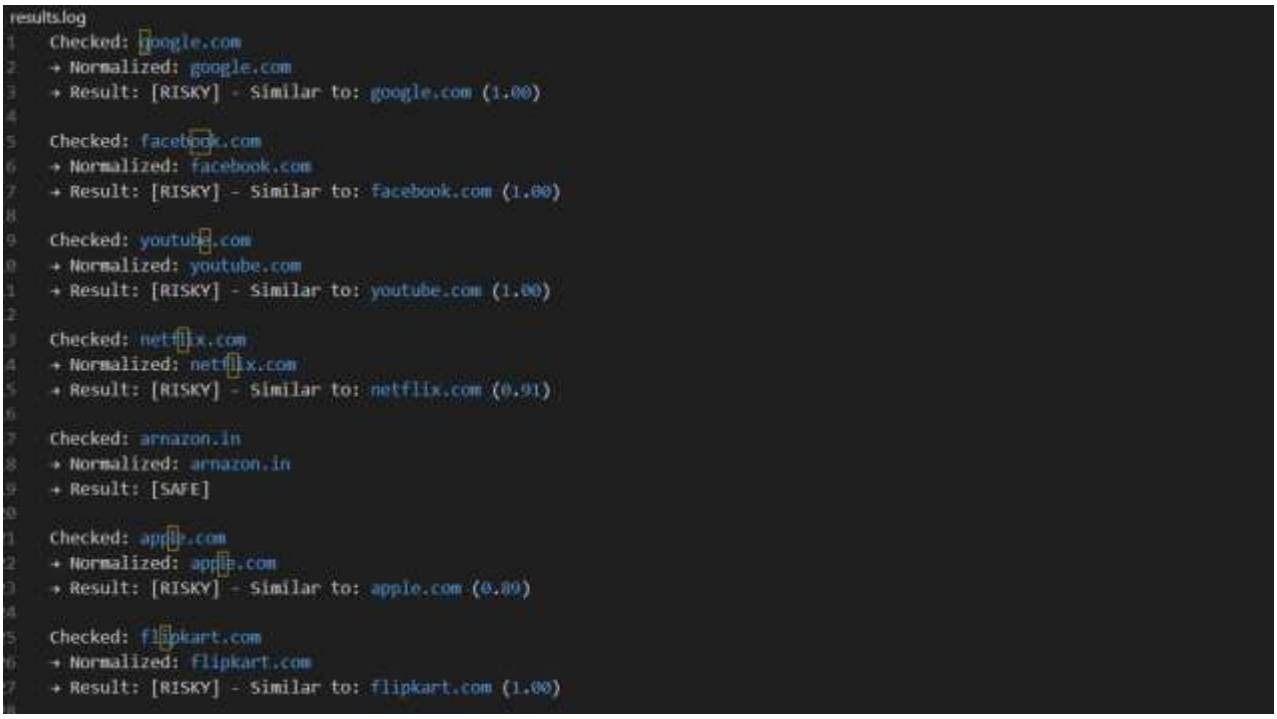
For each test domain:

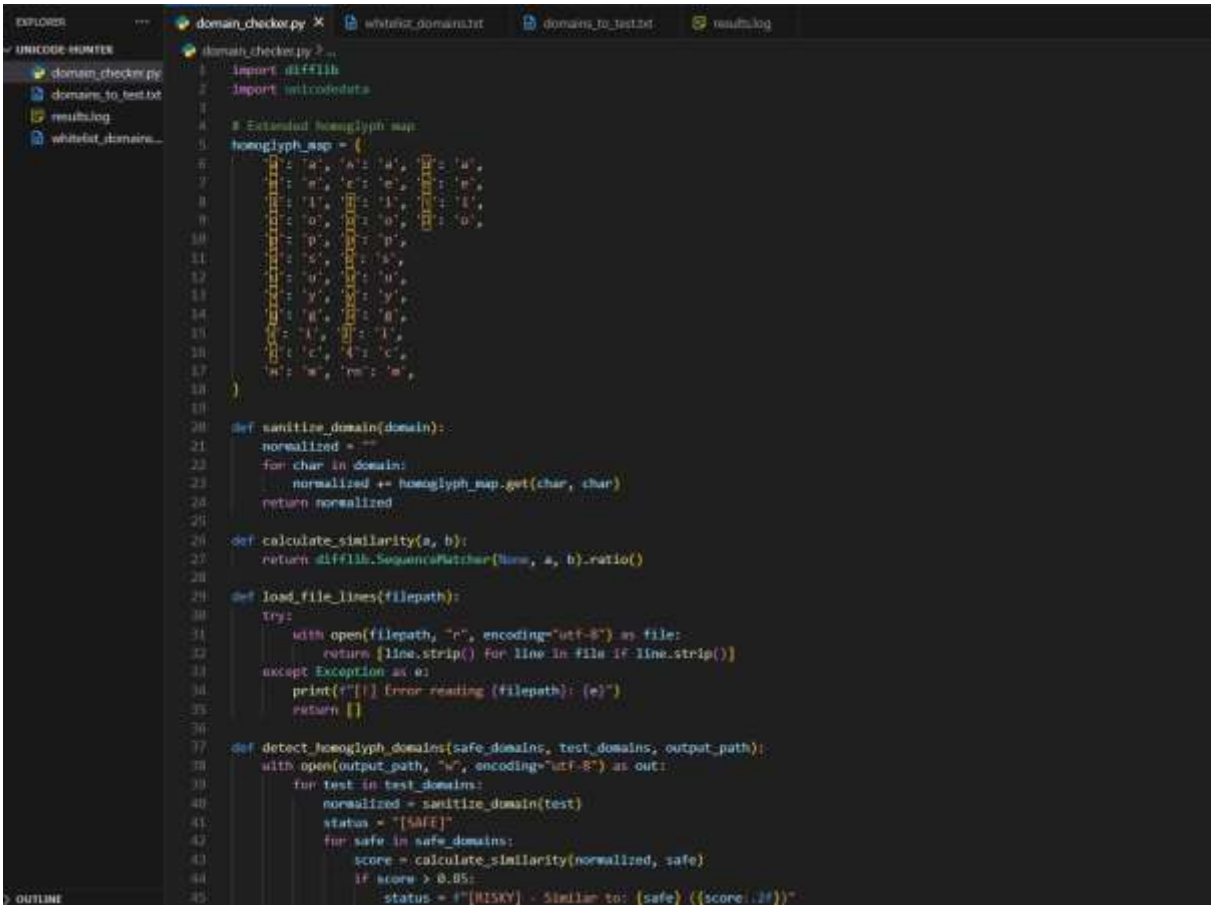
- Normalizes it using the homoglyph map.
- Compares it to each safe (whitelisted) domain.
- If similarity score > 0.85, marks it as [RISKY].
- Otherwise marks it as [SAFE].
- Writes the result to an output file (results.log).

Detection Approach (summary)

- ✓ Normalize domain names using NFKC and decode punycode
- ✓ (IDNA).
- ✓ Identify non-ASCII characters in the domain’s second-level
- ✓ name (SLD).
- ✓ Map confusing characters to standard ASCII equivalents using
- ✓ a confusable dictionary.
- ✓ Compare the transformed result with a list of safe domains.
- ✓ Raise flags on suspicious matches or unusual similarities for review.

Code:





```python
import difflib
import unicodedata

# Extended homoglyph map
homoglyph_map = {
    'а': 'a', 'А': 'a', 'Α': 'a',
    'е': 'e', 'Е': 'e', 'Ε': 'e',
    'і': 'i', 'І': 'i', 'Ι': 'i',
    'о': 'o', 'О': 'o', 'Ο': 'o',
    'р': 'p', 'Р': 'p',
    'с': 's', 'С': 's',
    'ս': 'u', 'Ս': 'u',
    'у': 'y', 'У': 'y',
    'ԛ': 'q', 'Ԛ': 'q',
    'ӏ': 'l', 'Ӏ': 'l',
    'ԁ': 'd', 'Ԁ': 'd',
    'ϲ': 'c', 'Ϲ': 'c',
    'ᴍ': 'm', 'rn': 'm',
}

def sanitize_domain(domain):
    normalized = ""
    for char in domain:
        normalized += homoglyph_map.get(char, char)
    return normalized

def calculate_similarity(a, b):
    return difflib.SequenceMatcher(None, a, b).ratio()

def load_file_lines(filepath):
    try:
        with open(filepath, "r", encoding="utf-8") as file:
            return [line.strip() for line in file if line.strip()]
    except Exception as e:
        print(f"[!] Error reading {filepath}: {e}")
        return []

def detect_homoglyph_domains(safe_domains, test_domains, output_path):
    with open(output_path, "w", encoding="utf-8") as out:
        for test in test_domains:
            normalized = sanitize_domain(test)
            status = "[SAFE]"
            for safe in safe_domains:
                score = calculate_similarity(normalized, safe)
                if score > 0.85:
                    status = f"[RISKY] - Similar to: {safe} ({score:.2f})"
```

Output:

What I Learned:

- ✓ Unicode allows dangerous visual deception.
- ✓ Skeleton mapping and normalization are vital first steps.
- ✓ Even simple tools can catch high-risk spoofing attempts.

- ✓ Manual review is always needed for critical decision-making.

Conclusion:

The implemented solution provides a foundation for identifying suspicious domain names using homoglyph characters. Although lightweight, it is effective for early detection and supports further enhancement using advanced techniques.