Homographic (Homoglyph) Detector — Weekly Report

Date: August 8, 2025

Objective

Build a simple yet effective tool to detect suspicious domain names and URLs that use homoglyphs—Unicode characters that visually resemble ASCII characters—to mimic trusted websites. For instance, a URL like www.arna■zon.com may look similar to www.amazon.com but uses deceptive Unicode characters.

Background (short)

Homoglyph or IDN homograph attacks rely on Unicode characters that appear visually similar to regular Latin letters but differ at the code level. These attacks often bypass standard security checks and deceive users into clicking malicious links. By combining normalization and character-mapping techniques, these threats can be effectively flagged before harm occurs.

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.

Tools & Libraries

• Python 3.x
• unicodedata — Unicode normalization
• idna — Handles IDN and punycode decoding
• tldextract — Splits domain into SLD and TLD
• difflib — Computes similarity ratios
• pytest (optional) — For test validation
• pandas (optional) — For exporting results

Short Confusable Examples (illustrative)

■ → U+0261 → Looks like: g
ο → U+03BF → Looks like: o (Greek omicron)
■ → U+0441 → Looks like: c (Cyrillic es)
■ → U+0430 → Looks like: a (Cyrillic a)
■ → U+0435 → Looks like: e (Cyrillic ie)
■ → U+066B → Looks like: , (Arabic decimal separator)

Minimal Implementation (core functions)

```
import unicodedata
import idna
import tldextract
import difflib

CONFUSABLES = {
    '\u0261': 'g',
    '\u03BF': 'o',
    '\u0441': 'c',
    '\u0430': 'a',
    '\u0435': 'e',
}

def normalize_domain(raw):
    host = raw.split('//')[-1].split('/')[0].split(':')[0]
    try:
        host = idna.decode(host)
    except Exception:
        pass
    return unicodedata.normalize('NFKC', host).lower()

def skeletonize(label):
    result = []
    for ch in label:
        if ch in CONFUSABLES:
            result.append(CONFUSABLES[ch])
        elif ord(ch) < 128:
            result.append(ch)
        else:
            norm = unicodedata.normalize('NFKD', ch)
            ascii_eq = ''.join(c for c in norm if ord(c) < 128)
            result.append(ascii_eq or '?')
    return ''.join(result)

def check_url(url, whitelist, sim_thresh=0.9):
    norm = normalize_domain(url)
    ext = tldextract.extract(norm)
    sld = ext.domain

    if not sld:
        return {'input': url, 'flag': False, 'reasons': ['no domain']}

    nonascii = any(ord(c) > 127 for c in sld)
    skel = skeletonize(sld)
    reasons = []

    if nonascii:
        reasons.append('contains non-ASCII characters')

    if skel in whitelist:
        reasons.append(f'skeleton equals whitelist `{skel}`')
        return {'input': url, 'normalized': norm, 'sld': sld, 'skeleton': skel, 'flag': True, 'r

    if nonascii:
        for w in whitelist:
            if abs(len(w) - len(skel)) > 3:
                continue
```

```
            sim = difflib.SequenceMatcher(None, skel, w).ratio()
            if sim >= sim_thresh:
                reasons.append(f'similarity {sim:.2f} to `{w}`')
                return {'input': url, 'normalized': norm, 'sld': sld, 'skeleton': skel, 'flag':

    return {'input': url, 'normalized': norm, 'sld': sld, 'skeleton': skel, 'flag': False, 'reas
```

Example Output (sample)

```
{ 'input': 'http://www.■oogle.com', 'normalized':
'www.■oogle.com', 'sld': '■oogle', 'skeleton': 'google', 'flag':
True, 'reasons': ['contains non-ASCII characters', 'skeleton
equals whitelist `google`'] }
```

How it Works (short)

- Normalize Unicode domain to a standard form.
- Extract the second-level domain (SLD) from the full URL.
- Replace any homoglyphs using a confusable map.
- Compare the cleaned string with known safe domains using exact and fuzzy match.

Results & Discussion

The tool is lightweight, explainable, and works effectively for initial triage. However, it is limited by the scope of the mapping dictionary. Fine-tuning fuzzy match thresholds can improve results. Adversaries may combine techniques (e.g., use of similar-looking subdomains or alternate TLDs).

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.

Future Improvements

- Expand character mapping using official confusables.txt.
- Add WHOIS lookups and DNS reputation scoring.
- Add rendering-based detection using OCR or screenshots.
- Build real-time monitoring for enterprise use.

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.

Name: Raj Verma
Intern ID: 335