Homographic (Homoglyph) Detector — Revised Report

Date: August 8, 2025

Objective

Detect potentially malicious domain names and URLs that use *homoglyphs* — visually similar Unicode characters — to impersonate trusted websites (for example, `www.paypal.com` using a Cyrillic `p` to mimic `paypal.com`). The goal is a compact, practical detector that flags suspicious inputs for human review.

Background (short)

An IDN homograph attack takes advantage of Unicode characters that look like ASCII letters but are different code points. These look-alikes are frequently used in phishing campaigns and credential-harvesting sites because they bypass casual visual inspection. Simple normalization alone is not sufficient; we combine normalization with a *skeletonization* mapping and heuristic checks to catch likely spoofs while minimizing false positives.

Detection Approach (summary)

1. **Normalize** incoming domain strings (NFKC) and decode punycode (IDNA) where present.
2. **Detect non-ASCII usage** in the second-level domain (SLD) — presence of non-ASCII characters is a primary signal.
3. **Skeletonize** the SLD by mapping known confusable Unicode characters to ASCII equivalents (based on a mapping table). This produces a comparable ASCII string.
4. **Compare** the skeleton against a whitelist of trusted SLDs (e.g., `google`, `paypal`, `amazon`) using exact match and a fuzzy-similarity check.
5. **Flag** and report domains with suspicious characteristics for human triage.

Tools & Libraries

- **Python 3.9+**
- `unicodedata` — Unicode normalization and decomposition
- `idna` — Punycode / IDN handling
- `tldextract` — split domain labels (SLD/TLD)
- `difflib` — quick similarity checks
- `pandas` (optional) — reporting / CSV export

- `pytest` (optional) — unit testing

Short Confusable Examples (illustrative)

Fake character	Codepoint	Looks like	ASCII target
ɡ	U+0261	ɡ	`g`
ο	U+03BF	ο (Greek omicron)	`o`
с	U+0441	с (Cyrillic es)	`c`
а	U+0430	а (Cyrillic a)	`a`
е	U+0435	е (Cyrillic ie)	`e`
٫	U+066B	٫ (Arabic decimal separator) — used in some obfuscations	`.` (special handling)

> *Note:* This table is a tiny sample. For production-grade detection use the Unicode Consortium's `confusables.txt` or an established confusable library.

Minimal Implementation (core functions)

```python
# Minimal skeletonization + check (run in Python 3.9+)
# pip install tldextract idna

import unicodedata
import idna
import tldextract
import difflib

# small starter mapping (extend from confusables.txt)
CONFUSABLES = {
    '\u0261': 'g',   # ɡ -> g
    '\u03BF': 'o',   # Greek omicron -> o
    '\u0441': 'c',   # Cyrillic es -> c
    '\u0430': 'a',   # Cyrillic a -> a
    '\u0435': 'e',   # Cyrillic ie -> 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):
    out = []
    for ch in label:
        if ch in CONFUSABLES:
            out.append(CONFUSABLES[ch])
            continue
        if ord(ch) < 128:
            out.append(ch)
            continue
        decomp = unicodedata.normalize('NFKD', ch)
        ascii_eq = ''.join(c for c in decomp if ord(c) < 128)
        out.append(ascii_eq or '?')
    return ''.join(out)

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, 'reasons': reasons}

    # fuzzy compare only if non-ASCII present (reduces false positives)
    if nonascii:
        for w in whitelist:
            if abs(len(w)-len(skel)) > 3:
                continue
            if difflib.SequenceMatcher(None, skel, w).ratio() >= sim_thresh:
                reasons.append(f'similarity {difflib.SequenceMatcher(None, skel, w).ratio():.2f} to `{w}`')
                return {'input': url, 'normalized': norm, 'sld': sld, 'skeleton': skel, 'flag': True, 'reasons': reasons}

    return {'input': url, 'normalized': norm, 'sld': sld, 'skeleton': skel,
'flag': False, 'reasons': reasons}

# Example
if __name__ == '__main__':
    wl = {'google', 'paypal', 'amazon'}
    for t in ['http://www.google.com', 'https://paypal.com', 'http://
амазон.com']:
        print(check_url(t, wl))
```

Example Output (sample)

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

How it Works (short)

- **Normalization:** make different Unicode forms uniform (NFKC) and decode `xn--` punycode.
- **Label extraction:** analyze the second-level domain (SLD) which is most often impersonated.
- **Skeletonization:** replace known confusables with ASCII equivalents and strip diacritics.
- **Comparison:** exact skeleton match or high fuzzy similarity against a whitelist + non-ASCII presence triggers a flag.

Results & Discussion

- **Strengths:** Lightweight, explainable, easy to run in batch or integrate into a triage pipeline.
- **Weaknesses:** Small mapping -> missed cases. Fuzzy matching thresholds need tuning to balance false positives vs. misses. Advanced adversaries combine TLD tricks, subdomain tricks, visually similar punctuation, or use homoglyphs in paths/usernames.

What I Learned

- The Unicode space contains many visually similar characters that can subvert naive checks.
- Normalization + skeletonization is a practical first line of defense.

- Human review remains important — automatic flagging is a triage aid, not a final verdict.

Future Improvements

- Load the official `confusables.txt` to construct a comprehensive mapping.
- Add a test suite (pytest) with positive/negative examples to measure FP/FN rates.
- Integrate domain reputation lookups and WHOIS checks to prioritize high-risk detections.
- Consider visual (render-and-compare) approaches for the hardest-to-detect cases.

Conclusion

A compact, explainable detector built on normalization, skeletonization, and simple similarity checks provides a practical early-warning system for homograph-based impersonation. With a larger confusables map and tuning, this approach scales for operational use as a triage layer in email and web protections.

Name: Sanskar Salvi

Intern Id: 334

End of report.