Homoglyph Domain Detector

Ghanshyam singh

Intern id:255

Purpose

This project introduces a basic Python tool to detect homoglyph-based domain spoofing attacks. These attacks exploit Unicode characters that look identical to trusted ones (e.g., Cyrillic “a” vs Latin “a”) to trick users into visiting fake websites.

Technologies Used

- **Python**: Core programming language

- **unicodedata**: For Unicode normalization

- **difflib**: For string similarity comparison

Trusted Domain List

```python
whitelist = [
    'google.com',
    'amazon.com',
    'facebook.com',
    'microsoft.com',
    'youtube.com'
]
```

Unicode Homoglyph Reference

Spoof Character	Unicode	Mimics	Legitimate
g	U+0261	g	g
o	U+03BF	o	o
c	U+0441	c	c
a	U+0430	a	a
e	U+0435	e	e

Detection Logic

```python
import unicodedata
import difflib

def normalize_domain(domain):
    """Standardize Unicode characters using NFKC normalization."""
    return unicodedata.normalize('NFKC', domain)
```

```python
def is_suspicious(domain):
    """Check if domain closely resembles a trusted one."""
    normalized = normalize_domain(domain)
    for safe in whitelist:
        similarity = difflib.SequenceMatcher(None, normalized, safe).ratio()
        if similarity > 0.8 and normalized != safe:
            return True, safe
    return False, None
```

Running the Tool

```python
if __name__ == "__main__":
    domain = input("Enter domain to verify: ")
    flagged, match = is_suspicious(domain)
    if flagged:
        print(f" Warning: '{domain}' is visually similar to '{match}'")
    else:
        print(" Domain looks safe.")
```

Example Output

```
Enter domain to verify: www.google.com
 Warning: 'www.google.com' is visually similar to 'google.com'

Enter domain to verify: www.learnhub.edu
 Domain looks safe.
```

How It Works

- **Normalization**: Converts deceptive Unicode characters to their canonical form.

- **Similarity Check**: Compares normalized domains against a whitelist using string similarity.

- **Threshold**: Flags domains with >80% similarity that aren’t exact matches.

Key Takeaways

- Unicode spoofing is a subtle but dangerous phishing tactic.

- Python’s built-in libraries can help detect visual deception.

- Whitelisting and similarity scoring are effective first-line defenses.

Final Thoughts

This project offers a practical introduction to defending against homoglyph attacks. While simple, it highlights how attackers manipulate Unicode to bypass filters and fool users. Future improvements could include dynamic homoglyph databases, punycode decoding, or even machine learning models for visual similarity detection. It’s a great starting point for anyone exploring cybersecurity and Python-based threat analysis.

Would you like me to add a custom title page or your friend’s name to personalize it further before exporting?