Proof of Concept – Homograph Attack

Name: Harshali Kasurde

Intern id: 299

**1. Definition**

Homograph attacks are a type of cyberattack where attackers register domain names using visually similar Unicode characters to impersonate legitimate websites.

For example:

• Real: apple.com

• Fake: аррӏе.com (using Cyrillic letters that look like Latin letters)

This attack bypasses human detection and can evade security filters, making it an effective vector for phishing, malware distribution, and credential theft.

**2. Approach**

This script focuses on detecting suspicious characters in domain names or URLs by checking if they:

• Are not part of standard ASCII character sets (letters, numbers, punctuation).

• Have Unicode values and names that differ from safe characters.

• Could be deceptive in visual appearance.

It works by:

• Extracting the domain from a given URL.

• Scanning each character of the domain.

• Identifying and reporting characters that are unfamiliar, non-standard, or invisible Unicode traps.

3. Need and Significance Why This Matters in Cybersecurity:

• Traditional systems may not flag Unicode-based spoofing.

• Attackers exploit this to mimic well-known brands.

• This is widely used in phishing and typosquatting attacks.

• Protecting users from visually deceptive domains is essential in:

Secure browsing

Email filtering

DNS-level blocking

Zero-trust security environments

**4. Logic Behind homograph\_core.py**

Step-by-Step Logic:

1. Define safe characters:

a. Standard ASCII (A–Z, a–z, 0–9, symbols) are marked as safe.

b. A few control characters like newline \n, tab \t is allowed.

2. Scan domain character-by-character:

a. For each character:

If it is not part of the safe set

Fetch its Unicode name and code point (e.g., U+0430).

Report it as a suspicious Unicode character.

3. Extract domain from a URL:

a. If a full URL is given (e.g., http://google.com), extract just google.com for scanning.

**5. Code python**

Python file: homoTool.py

import unicodedata

import string

from urllib.parse import urlparse

*# Set of standard printable ASCII characters (A-Z, 0-9, punctuation, space)*

standard\_chars = set(string.ascii\_letters + string.digits + string.punctuation + " ")

safe\_invisible\_chars = {'\n', '\r', '\t'}

*# Check if character is standard ASCII*

def is\_allowed\_standard\_char(ch):

    return ch in standard\_chars

*# Check if a character is suspicious (non-standard or unsafe control char)*

def is\_suspicious(ch):

    if ch in safe\_invisible\_chars:

        return False

    if ch in standard\_chars:

        return False

    return True

*# Scan text for suspicious characters and return details*

def scan\_text(text):

    results = []

    for ch in text:

        if is\_suspicious(ch):

            try:

                name = unicodedata.name(ch)

            except ValueError:

                name = "Unknown or Non-character"

            codepoint = f"U+{ord(ch):04X}"

            results.append((ch, name, codepoint))

    return results

*# Extract domain part from a full URL (or treat input as a domain)*

def extract\_domain(url):

    parsed = urlparse(url)

    return parsed.netloc or parsed.path

*# Complete scan pipeline for a domain or full URL*

def scan\_domain(domain\_or\_url):

    domain = extract\_domain(domain\_or\_url)

    return scan\_text(domain)

if \_\_name\_\_ == "\_\_main\_\_":

    url = input("Enter a domain or full URL to scan: ").strip()

    suspicious = scan\_domain(url)

    if suspicious:

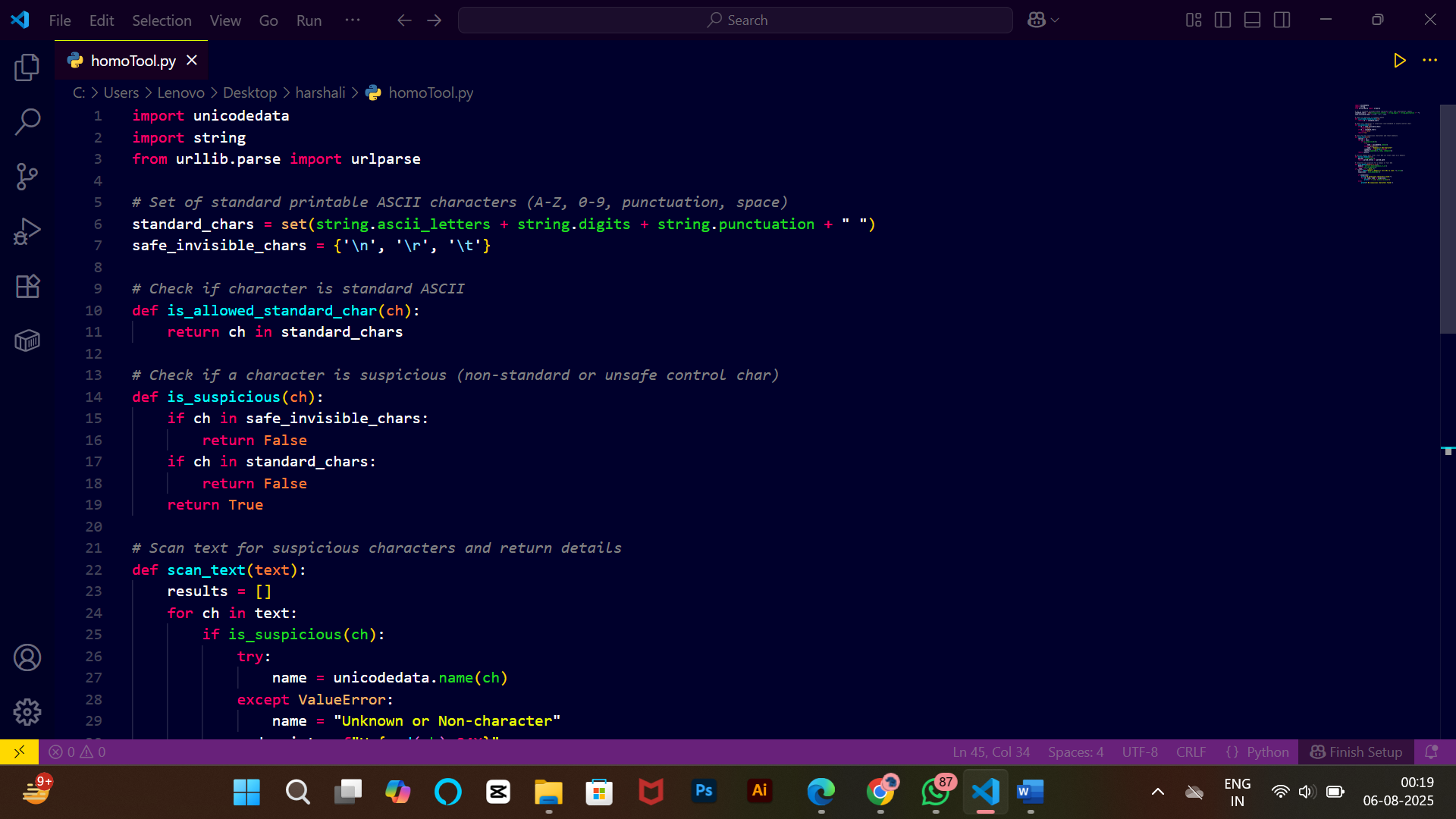
        print("Suspicious characters found:")

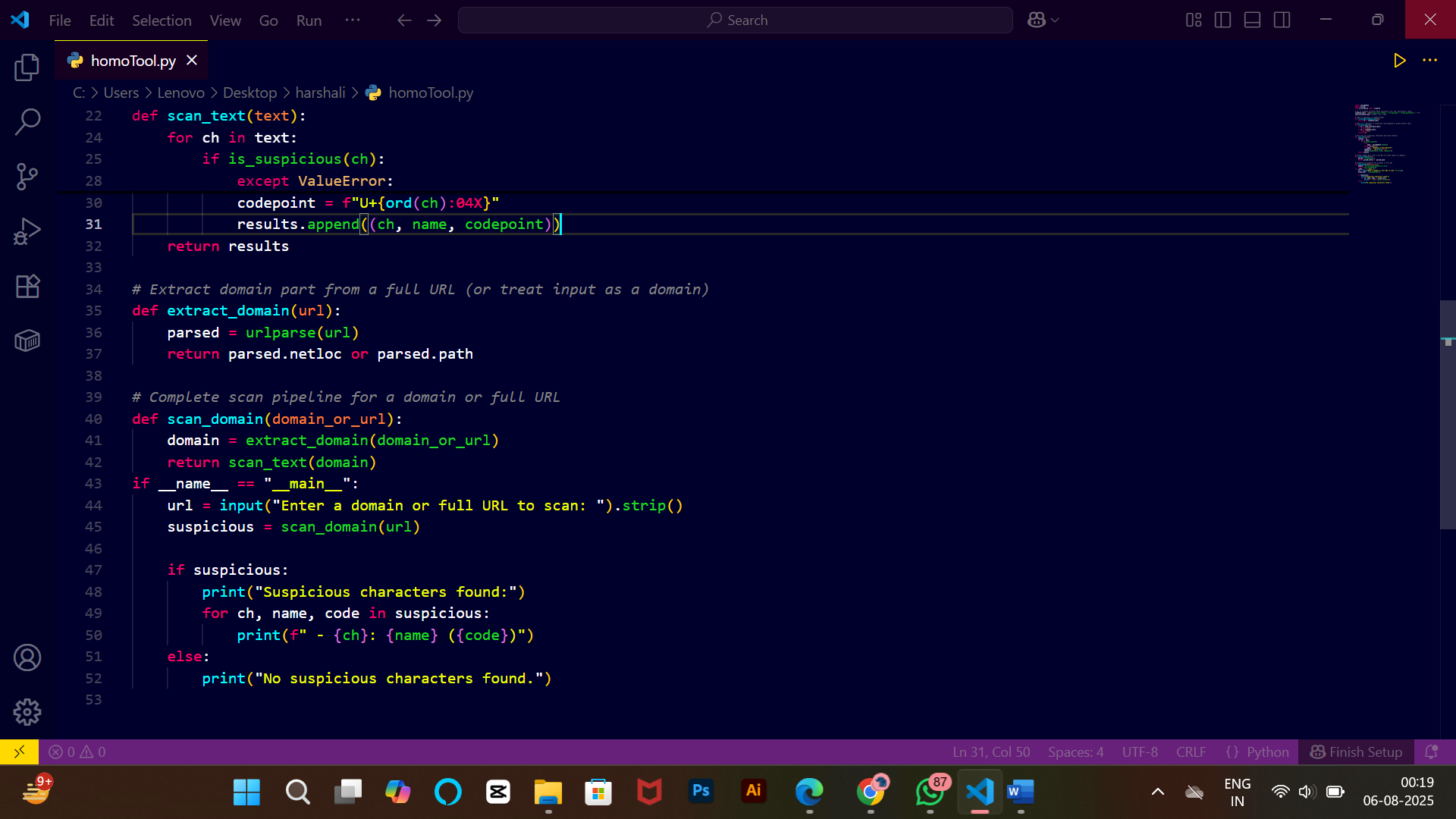
        for ch, name, code in suspicious:

            print(f" - {ch}: {name} ({code})")

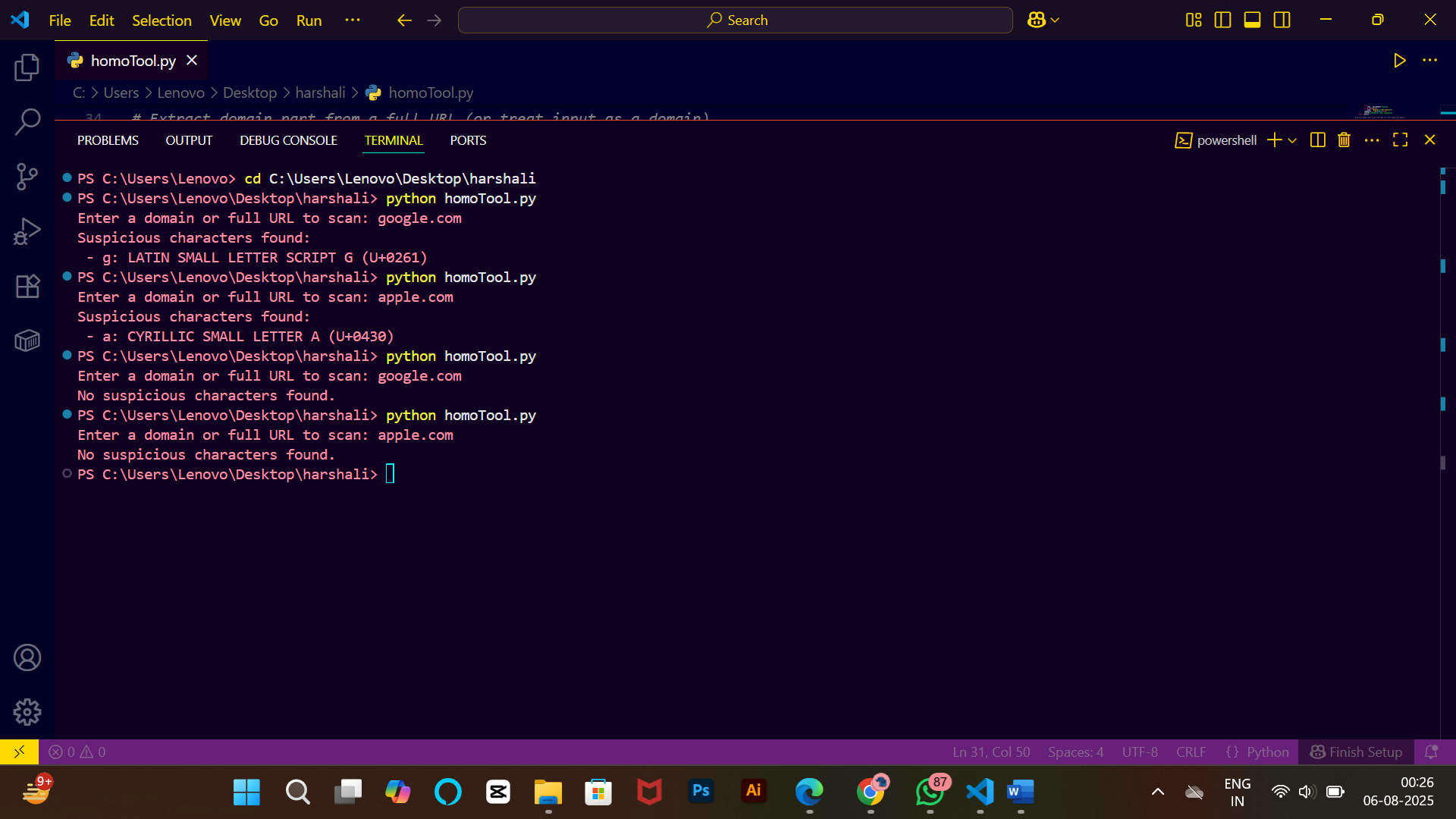
    else:

        print("No suspicious characters found.")





**6.Output**

****