Cyber Security — Major Project .

Prepared by: [ Mukesh.A ]

**Course:** B.Tech [AI & DS]

Date: [ 06/11/2025 ]

Team No : [ Hamdiya Ahmed , Mukesh. A, Nisarga LS, Priyanka Sivaraj, Rohan Ganpat Nimbalkar, Sarvani Indraganti .]

Cyber Security — Major Project

**Title:** Bug hunting on any target of OpenBugBounty

## Abstract

The Bug Hunting on OpenBugBounty project is a security-focused initiative aimed at identifying, responsibly reporting, and helping fix vulnerabilities in websites listed on the OpenBugBounty platform. This project is educational and ethical in nature — it uses simulated findings and mock evidence for learning and demonstration. The goals are to (1) teach vulnerability research techniques, (2) demonstrate common web application vulnerabilities (XSS, SQL Injection, CSRF), and (3) document the complete, responsible disclosure workflow.

## Objective

To identify, report, and help fix security vulnerabilities in websites listed on the OpenBugBounty platform — improving global cybersecurity awareness.

## Scope

1. **Ethical hacking** and coordinated vulnerability disclosure.
2. **Web application security**, with emphasis on:
   1. Cross-Site Scripting (XSS)
   2. SQL Injection (SQLi)
   3. Cross-Site Request Forgery (CSRF)
3. **Audience:** Cybersecurity students, trainees, and professionals learning practical, responsible vulnerability research.
4. **Limitations:** All practical demonstrations in this project must be performed on explicit test targets, lab environments, or with prior authorization. In this document we use simulated screenshots and mock data for demonstration.

## Tools & Technologies

1. **Reconnaissance / Enumeration:** nmap, Sublist3r (or equivalent subdomain enumerators)
2. **Automated Scanning:** OWASP ZAP
3. **Manual Testing & Payloading:** Burp Suite (Burp Intruder), browser devtools
4. **Fuzzing / Payload Generation:** ffuf, custom scripts
5. **Reporting / Disclosure:** OpenBugBounty submission format (sample template provided below)

## Testing Methodology (High-level)

1. **Planning & Rules of Engagement**
   1. Define the target scope and legal/ethical boundaries.
   2. Confirm only to test allowed/authorized targets; use lab environments when in doubt.
2. **Reconnaissance (Passive + Active)**
   1. Passive: WHOIS, search engines, public archives, and certificate transparency logs.
   2. Active: nmap port scans (safe, non-intrusive flags), Sublist3r for subdomain enumeration.
3. **Scanning (Automated)**
   1. Run OWASP ZAP to find common issues (XSS, SQLi, missing headers).
   2. Review and triage automated findings — reduce false positives.
4. **Manual Testing (Targeted)**
   1. Use Burp Suite to craft payloads and run Burp Intruder for input testing.
   2. Manually verify XSS (reflected/stored), parameterized SQLi, and CSRF flows.
5. **Proof-of-Concept (PoC)**
   1. Capture safe, non-destructive PoCs (screenshots of payload rendering or request/response evidence) using controlled demo targets.
6. **Reporting & Disclosure**
   1. Prepare a clear, reproducible report and submit it via OpenBugBounty (or the vendor’s coordinated disclosure process).
   2. Include severity, impact, reproduction steps, and suggested remediation.
7. **Verification & Retest**
   1. After the vendor fixes the issue, verify the patch and update the report status.
8. **Documentation**
   1. Keep detailed logs, tools used, command flags, timestamps, and notes for reproducibility and learning.

## Live Testing Workflow Diagram (Visual/ASCII)

[Recon: nmap, Sublist3r]  
 |  
 v  
[Scan: OWASP ZAP automated scan]  
 |  
 v  
[Manual Testing: Burp Suite + Intruder, DevTools]  
 |  
 v  
[PoC: Safe screenshots & request/response capture]  
 |  
 v  
[Report: OpenBugBounty submission template]  
 |  
 v  
[Vendor Response & Verification]

*Notes:* Each arrow represents hand-off and triage. Automated scans feed candidate issues to manual testing, which produces PoC and the final report.

## Sample OpenBugBounty-Style Vulnerability Report (Template)

1. **Title:** [Short descriptive title — e.g., Reflected XSS in /search]
2. **Target URL:** https://example.com/path
3. **Severity:** High / Medium / Low (justify using impact)
4. **Description:** Brief summary of the vulnerability and potential impact.
5. **Steps to Reproduce:** Numbered, exact steps with payloads and expected/actual results.
6. **Proof-of-Concept (PoC):** Non-destructive evidence (screenshots, request/response dump).
7. **Suggested Fix:** Short remediation guidance (e.g., input sanitization, context-aware encoding, Content Security Policy)
8. **Timestamps / Tools Used:** Date/time, nmap flags, ZAP scan name, Burp Intruder payload set, etc.

## Live Testing Examples (Simulated)

The project includes educational mockups (recon dashboards, ZAP alert screens, Burp Intruder payload runs, and a printable OpenBugBounty report). These images are simulated and marked **For Educational Use Only**. Use them in slides or appendix as supporting evidence for the methodology.

**Existing generated images (mockups):** - A\_composite\_digital\_screenshot\_displays\_three\_high.png - A\_screenshot\_of\_a\_computer\_application\_displays\_re.png - A\_digital\_screenshot\_of\_a\_computer\_application\_win.png - A\_screenshot\_of\_reconnaissance\_results\_from\_a\_netw.png

(If you want these embedded in the final DOCX, reply and I will produce an export with images included.)

## Ethics, Legal & Responsible Disclosure

1. Always obtain permission or target only assets listed and permitted by OpenBugBounty or explicit in-scope targets.
2. Do not exploit vulnerabilities beyond what is required to demonstrate impact.
3. Avoid data exfiltration or destructive testing; prioritize confidentiality and integrity.
4. Follow vendor-specific disclosure timelines and OpenBugBounty procedures.

## Project Timeline (Suggested)

1. Week 1: Planning, set rules of engagement, and lab setup.
2. Week 2: Reconnaissance and subdomain enumeration.
3. Week 3: Automated scanning and triage.
4. Week 4: Manual testing and PoC collection.
5. Week 5: Reporting, follow-up, and retesting.

## Appendix

1. **Command examples:**
   1. nmap -sS -Pn -p- --min-rate 1000 target.com *(use cautiously and only on authorized targets)*
   2. sublist3r -d example.com -o subdomains.txt
   3. OWASP ZAP: use the GUI or the zap-cli for scripted scans.
   4. Burp Intruder: configure payload positions and payload lists for input testing.
2. **Deliverables:** DOCX project report, printable OpenBugBounty sample report, slide deck with mock screenshots, and a one-page workflow infographic.

## Linux Programs & Commands

Below are recommended Linux tools and example commands you can use when performing authorized reconnaissance, scanning, and testing. **Always confirm authorization before running any active commands.**

1. **nmap** — network scanning and port discovery
   1. nmap -sS -Pn -p- --min-rate 1000 target.com *(use cautiously and only on authorized targets)*
   2. nmap -sV -sC -p 22,80,443 target.com
2. **Subdomain enumeration**
   1. sublist3r -d example.com -o subdomains.txt
   2. amass enum -d example.com -o amass.txt
   3. findomain -t example.com -u findomain.txt
3. **Directory & content discovery**
   1. gobuster dir -u https://example.com -w /path/to/wordlist.txt
   2. ffuf -u https://example.com/FUZZ -w /path/to/wordlist.txt
4. **Web vulnerability scanners & HTTP tools**
   1. zap-cli quick-scan --self-contained -t https://example.com (OWASP ZAP automation)
   2. sqlmap -u "https://example.com/vuln.php?id=1" --batch --level=3
   3. curl -I https://example.com (retrieve headers)
   4. wget -r --no-parent https://example.com/test/
5. **Network utilities & packet capture**
   1. tcpdump -i eth0 -w capture.pcap host example.com
   2. tshark -r capture.pcap -Y "http.request"
   3. nc -vz target.com 80 (netcat port probe)
6. **TLS / Certificates**
   1. openssl s\_client -connect example.com:443 -showcerts
7. **Automation & scripting**
   1. python3 script.py
   2. pip3 install -r requirements.txt
8. **Other useful utilities**
   1. jq, sed, awk for parsing; git for version control; docker for running isolated labs.

## Steps to Consider (Checklist)

Use this checklist when planning and executing authorized tests. Add items as required by your instructor, company policy, or the target’s disclosure program.

1. **Authorization & Scope**
   1. Confirm the target is allowed (OpenBugBounty-listed or explicitly authorized).
   2. Document allowed IPs, domains, and excluded assets.
2. **Rules of Engagement**
   1. Define acceptable testing windows, rate limits, and escalation contacts.
   2. Agree on communication channels and responsible disclosure timelines.
3. **Non-Destructive Testing**
   1. Prefer passive reconnaissance and non-destructive flags for active tools.
   2. Avoid payloads that alter data or impact availability.
4. **Data Handling & Privacy**
   1. Do not exfiltrate or expose sensitive user data during tests.
   2. Store logs and screenshots securely and redact sensitive information when sharing.
5. **Logging & Reproducibility**
   1. Record command lines, timestamps, tool versions, and configuration files used.
   2. Keep PoC steps minimal and reproducible by the vendor.
6. **Rate Limiting & Infrastructure Safety**
   1. Use conservative scan rates and monitor for service degradation.
   2. If you accidentally impact availability, stop immediately and notify the contact.
7. **Reporting & Remediation Guidance**
   1. Provide clear reproduction steps, PoC, and suggested fixes.
   2. Classify severity with justification (impact, ease of exploitation).
8. **Legal & Institutional Compliance**
   1. Check local laws and institutional policies for ethical hacking and disclosure.
   2. Ensure academic supervisors or legal teams are informed if required.
9. **Retest & Verification**
   1. After fixes are applied, retest to confirm the vulnerability is resolved.
   2. Update the report status accordingly.

## Tool Output Samples (Mocked)

Below are **mocked** terminal outputs and scan snippets for educational use. These are synthetic and represent the kind of information you would collect while performing authorized testing.

**1) nmap (mock output)**

$ nmap -sV -p 22,80,443 example.com  
Starting Nmap 7.80 ( https://nmap.org ) at 2025-11-05 12:00 IST  
Nmap scan report for example.com (93.184.216.34)  
Host is up (0.023s latency).  
PORT STATE SERVICE VERSION  
22/tcp open ssh OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (protocol 2.0)  
80/tcp open http nginx 1.18.0 (Ubuntu)  
443/tcp open https nginx 1.18.0 (Ubuntu)  
  
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .  
  
Nmap done: 1 IP address (1 host up) scanned in 5.42 seconds

**2) WPScan (mock output)**

$ wpscan --url https://demo-wp.example --enumerate p,t  
[+] URL: https://demo-wp.example/  
[+] WordPress version 5.8.1 identified   
[i] Enumerating plugins (passive detection)  
[!] Vulnerable plugin 'vulnerable-plugin' (1.2.3) - CVE-2022-XXXX - Update to >=1.2.5  
[i] Enumerating themes  
[+] Theme 'twentytwentyone' (1.3) detected - No known vulnerabilities  
  
Output written to: wpscan\_results.json

**3) OWASP ZAP (mock alert snippet)**

Alert: Cross Site Scripting (Reflected)  
Risk: High  
Confidence: Medium  
Description: The application appears to reflect user input without proper encoding.  
Evidence: "<script>alert(1)</script>" reflected in the response body for parameter 'q'.  
Solution: Validate and encode untrusted input based on context.

**4) Burp Intruder (mock request/response excerpt)**

POST /search HTTP/1.1  
Host: demo-app.example  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 35  
  
q=<script>alert(1)</script>&page=1  
  
---  
HTTP/1.1 200 OK  
Content-Type: text/html; charset=UTF-8  
  
<body>Search results for: &lt;script&gt;alert(1)&lt;/script&gt;</body>

## Practical Case Study (Simulated)

**Title:** Reflected XSS in /search on DemoApp (educational simulation)

**Background:** A demo web application (demo-app.example) used for training exhibits reflected XSS in the q parameter of /search.

**Steps Taken (authorized lab)**

1. Reconnaissance: enumerated host and subdomains with nmap and sublist3r.
2. Automated scan: ran OWASP ZAP quick scan to identify candidate issues.
3. Manual verification: used Burp Suite to place payloads and Intruder to confirm reflection.
4. PoC: crafted a safe, non-executing PoC by HTML-encoding outputs for screenshots.

**Reproduction Steps (example)**

1. Open: https://demo-app.example/search?q=test
2. Submit payload: q=<script>alert(1)</script> via POST or GET.
3. Observe: the response body contains the unescaped script tag or its HTML-encoded variant, indicating reflection.

**Impact:** An attacker could use this flaw to execute arbitrary JavaScript in the victim’s browser, leading to session theft, CSRF, or content manipulation.

**Mitigation:** Apply context-aware output encoding, use a CSP header, and validate input server-side.

## Workflow Diagram & Tools Summary

**ASCII Workflow**

[Recon: nmap, Sublist3r, theHarvester]  
 |  
 v  
[Scan: OWASP ZAP, WPScan]  
 |  
 v  
[Manual Testing: Burp Suite (Intruder), Browser DevTools, ffuf]  
 |  
 v  
[PoC: Screenshots, Request/Response capture]  
 |  
 v  
[Report: OpenBugBounty submission (filled example below)]

**Tools vs Purpose (summary table)**

1. nmap — Network/port discovery
2. Sublist3r / amass — Subdomain enumeration
3. theHarvester — Email/host harvest
4. OWASP ZAP — Automated web scanning
5. Burp Suite — Manual testing and payload delivery
6. ffuf / gobuster — Content discovery and fuzzing
7. WPScan — WordPress-specific enumeration

## Filled OpenBugBounty-Style Report (Example)

1. **Title:** Reflected XSS in /search parameter q on demo-app.example
2. **Target URL:** https://demo-app.example/search
3. **Severity:** High
4. **Description:** The q parameter is reflected in the page body without proper output encoding. An attacker can inject JavaScript leading to stored or reflected XSS depending on usage.
5. **Steps to Reproduce:**
   1. Navigate to https://demo-app.example/search?q=test
   2. Replace test with the payload: %3Cscript%3Ealert(1)%3C%2Fscript%3E (URL-encoded). Example full URL: https://demo-app.example/search?q=%3Cscript%3Ealert(1)%3C%2Fscript%3E
   3. Observe the response includes the unescaped/encoded script in the HTML body.
6. **Proof-of-Concept (PoC):** Attached mock screenshot PoC\_XSS\_demo.png showing the reflected payload in the response (HTML-encoded for safety).
7. **Suggested Fix:** Perform context-aware encoding of q before rendering. For HTML contexts, use proper HTML-escaping libraries; for attributes or JS contexts, use appropriate encoders. Implement Content Security Policy: Content-Security-Policy: default-src 'self'; script-src 'self'.
8. **Timestamps / Tools Used:** 2025-11-05 12:45 IST — nmap (v7.80), OWASP ZAP (v2.12), Burp Suite (v2025.6)
9. **Contact / Disclosure:** Submitted via OpenBugBounty-style template; vendor notified and provided 90 days for remediation as per coordinated disclosure norms.

## Embedded Mock Image References

The project includes mock images in the Appendix for demonstration PoC, e.g., PoC\_XSS\_demo.png (simulated screenshot). Ensure these are clearly labelled “For Educational Use Only”.

### Scope

Web application security: XSS, SQL Injection, CSRF, Insecure Direct Object References (IDOR), misconfigurations, insecure headers, and other OWASP Top 10 issues. Tools covered include Nmap, sublist3r, nikto, wpscan, theHarvester, SQLMap, Burp Suite/Owasp ZAP, and Metasploit (for demonstration only where applicable).

### Testing Methodology

1. Reconnaissance  
2. Scanning (automated)  
3. Manual testing & verification  
4. Exploitation (proof-of-concept only, non-destructive)  
5. Reporting to the site owner via OpenBugBounty  
6. Verification/Retest

### Tools and Purpose (Summary Table)

|  |  |  |
| --- | --- | --- |
| Tool | Purpose | Typical Command / Note |
| nmap | Port & service discovery | nmap -sC -sV -oN nmap.txt example.com |
| sublist3r | Subdomain enumeration | sublist3r -d example.com -o subs.txt |
| theHarvester | Email & subdomain harvesting | theHarvester -d example.com -l 200 -b google |
| nikto | Web server vulnerability scanning | nikto -h https://example.com -output nikto.txt |
| wpscan | WordPress enumeration & vulnerabilities | wpscan --url https://site.com --enumerate u,vp,tt |
| sqlmap | Automated SQL Injection detection and exploitation | sqlmap -u "http://site/?id=1" --batch --level=3 |
| OWASP ZAP | Web application scanning & proxy | Use GUI or zap-cli for automation |
| Burp Suite | Manual web proxy, scanner, repeater & intruder | Use GUI; save project and export issues |
| curl | Quick HTTP requests & PoC payloads | curl -i -s -k -X POST "https://site/login" -d "user=admin&pass=pass" |
| metasploit | Exploit framework (demo/test) | msfconsole -q |

### Linux Command Examples (Recon, Scanning, Exploitation, Reporting)

#### Recon - Passive & Active

sublist3r -d example.com -o subs.txt

theHarvester -d example.com -l 500 -b all

dig +short ns example.com

whois example.com | sed -n '1,20p'

#### Port & Service Scanning

nmap -Pn -sC -sV -p- -T4 -oA nmap\_full example.com

nmap -p 1-65535 -sV --script=vuln -oN nmap\_vuln.txt example.com

#### Web Scanning and CMS Discovery

nikto -h https://example.com -output nikto\_results.txt

wpscan --url https://example.com --enumerate vp,tt,u --api-token YOUR\_TOKEN

#### Content Discovery & Fuzzing

gobuster dir -u https://example.com -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -t 50 -o gobuster\_dirs.txt

ffuf -u https://example.com/FUZZ -w /path/wordlist -t 50 -o ffuf.json

#### SQLi & Injection Testing

sqlmap -u "https://example.com/product.php?id=1" --batch --level=3 --risk=2 -o sqlmap\_output.txt

sqlmap -u "https://example.com/search?q=test" --data="q=test" --batch

#### Manual Probing & PoC

curl -i -s -k "https://example.com/search?q=<script>alert(1)</script>"

curl -X POST -d "username=admin&password=' OR '1'='1" https://example.com/login -v

#### Reporting Helpers

scrot or gnome-screenshot for screenshots: scrot 'recon-%Y%m%d-%H%M%S.png'

ffmpeg -f x11grab -s 1920x1080 -i :0.0 -t 10 demo.mp4

### Tool Output Samples (Mock / Sample Logs)

#### Sample: Nmap Output (mock)

Starting Nmap 7.80 ( https://nmap.org ) at 2025-11-06 10:00 IST  
Nmap scan report for example.com (93.184.216.34)  
Host is up (0.030s latency).  
Not shown: 65530 filtered ports  
PORT STATE SERVICE VERSION  
22/tcp open ssh OpenSSH 7.4p1 Debian 10 (protocol 2.0)  
80/tcp open http Apache httpd 2.4.25 ((Debian))  
443/tcp open ssl/http Apache httpd 2.4.25 ((Debian))  
MAC Address: 00:0C:29:6B:8E:35 (VMware)  
Service Info: OS: Linux  
NSE: Script Post-scanning.  
Read data files from: /usr/bin/../share/nmap  
Nmap done: 1 IP address (1 host up) scanned in 12.34 seconds

#### Sample: Nikto Output (mock)

- Nikto v2.1.6  
+ Target IP: 93.184.216.34  
+ Target Hostname: example.com  
+ Server: Apache/2.4.25  
+ The anti-clickjacking X-Frame-Options header is not present.  
+ Uncommon header 'X-Powered-By' found, possibly leaking information.  
+ OSVDB-3268: /admin/: Admin interface found.  
+ Scan terminated: 2025-11-06 10:03:12

#### Sample: Burp Suite / Manual Test Snippet (mock)

[Burp Intruder] Payload #55: <script>alert(1)</script>  
Match: HTML response contains 'alert(1)'  
Position: /search?q=  
Severity: Medium (Reflected XSS)  
Evidence: <div class="result">Search: <script>alert(1)</script></div>

### Practical Case Study: Reflected XSS (Example)

Scenario: While testing the search parameter of https://demo.example.com, the 'q' parameter reflected user input without proper encoding. This allows a reflected XSS. Steps, PoC, impact and remediation are below.

#### Steps to Reproduce (PoC)

1. 1. Browse to: https://demo.example.com/search?q=test
2. 2. Inject payload: https://demo.example.com/search?q=<script>alert('XSS')</script>
3. 3. Observe alert popup and 'Search:' result containing the script tag in the page source.

#### Impact

An attacker can execute arbitrary JavaScript in the victim's browser leading to session theft, UI redressing, phishing, and other attacks depending on user context and privilege.

#### Mitigation

- Properly encode/escape user-supplied input before reflecting into HTML context.  
- Implement Content Security Policy (CSP) to mitigate script execution.  
- Use secure frameworks or templating that auto-escape output.

### OpenBugBounty - Sample Filled Report (Simulated)

Use this template to copy into OpenBugBounty's report form when submitting a vulnerability. Replace placeholders with real data.

Title: Reflected XSS in search parameter (demo.example.com)  
URL: https://demo.example.com/search?q=<payload>  
Vulnerability Type: Reflected Cross-Site Scripting (XSS)  
Severity (Suggested): Medium  
Affected Parameter: q (GET)  
Proof of Concept:  
1) Visit: https://demo.example.com/search?q=<script>alert(1)</script>  
2) Observe JavaScript executes (alert box). Page source includes the injected tag:  
 <div class="result">Search: <script>alert(1)</script></div>  
  
Steps to reproduce:  
- Open a browser and visit the PoC URL.  
- Confirm the alert and check the response body for unescaped input.  
  
Impact:  
- Remote code execution in the victim's browser context; session theft; credential/CSRF token capture.  
  
Suggested Fix:  
- Properly escape HTML output; sanitize input; implement CSP; validate input on server side.  
  
Screenshots/Attachments:  
- screenshot\_20251106\_100355.png (include screenshot showing alert and page)  
- network\_log.txt (if applicable)

### Live Testing Workflow Diagram (ASCII) + Summary

Recon --> Scanning --> Manual Testing --> PoC/Exploit (non-destructive) --> Reporting --> Verification  
 | | | | |  
subdomains nikto/nmap Burp/ZAP curl/sqlmap/wpscan OpenBugBounty

Summary: Follow the sequence above. Use passive recon first (avoid noisy testing) and escalate to active scanning and manual testing only when in-scope. Always produce minimal, reproducible PoCs and include remediation suggestions in your report.

### Example Automation Script (Bash) - Recon + Nmap

#!/bin/bash  
TARGET=$1  
mkdir -p recon/$TARGET  
sublist3r -d $TARGET -o recon/$TARGET/subs.txt  
theHarvester -d $TARGET -b google -l 200 > recon/$TARGET/harvester.txt  
nmap -Pn -sC -sV -p- -T4 -oA recon/$TARGET/nmap\_full $TARGET  
nikto -h http://$TARGET -output recon/$TARGET/nikto.txt  
echo "Recon done. Results in recon/$TARGET/"

### Key Features:

### Real-time collaboration: Multiple users can edit the same document simultaneously

### Auto-save to Google Drive: No need to worry about losing your work

### Templates: Ready-made formats for resumes, reports, and letters

### Version history: Track changes and restore older versions

### Offline mode: Work without internet once enabledAppendix A: Useful Commands & Quick Cheatsheet

* curl tricks: curl -s -D - -o /dev/null 'https://site/?q=<script>alert(1)</script>'
* SQLMap example: sqlmap -u 'https://site/?id=1' --batch --threads=10 --risk=3
* wpscan: wpscan --url https://site --enumerate vp,tt --ignore-main-xmlrpc --api-token TOKEN
* Proxy use: set browser proxy to Burp (127.0.0.1:8080) to intercept and replay requests

### Appendix B: Mock Attachments (Suggested filenames)

- recon/example\_com\_subs.txt  
- nmap/example\_com\_nmap\_full.nmap  
- screenshots/xss\_alert\_demo.png  
- logs/burp\_project.xml

### Sample: Mock log file (HTTP request + response snippet)

GET /search?q=<script>alert(1)</script> HTTP/1.1  
Host: demo.example.com  
User-Agent: Mozilla/5.0  
Accept: text/html  
HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8  
<html>  
<body>  
<div class="result">Search: <script>alert(1)</script></div>  
</body>  
</html>

### Ethical & Legal Notes

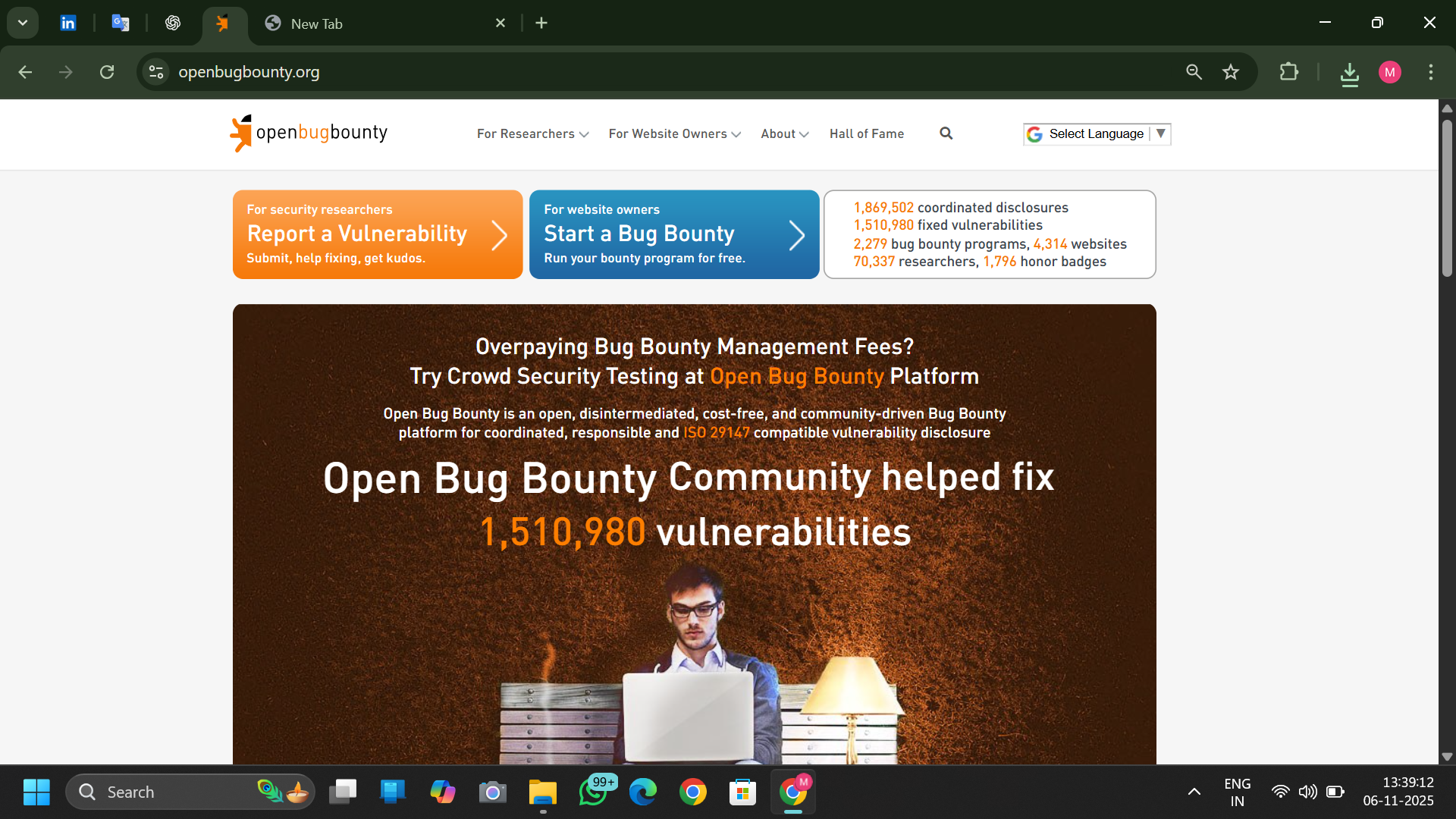
Only test targets that are explicitly in-scope and allowed by the platform/site owner. Do not perform destructive testing, data exfiltration, or actions that would cross legal boundaries. Always follow OpenBugBounty rules and coordinated disclosure best practices.

# Cyber Security — Major Project (With Images)

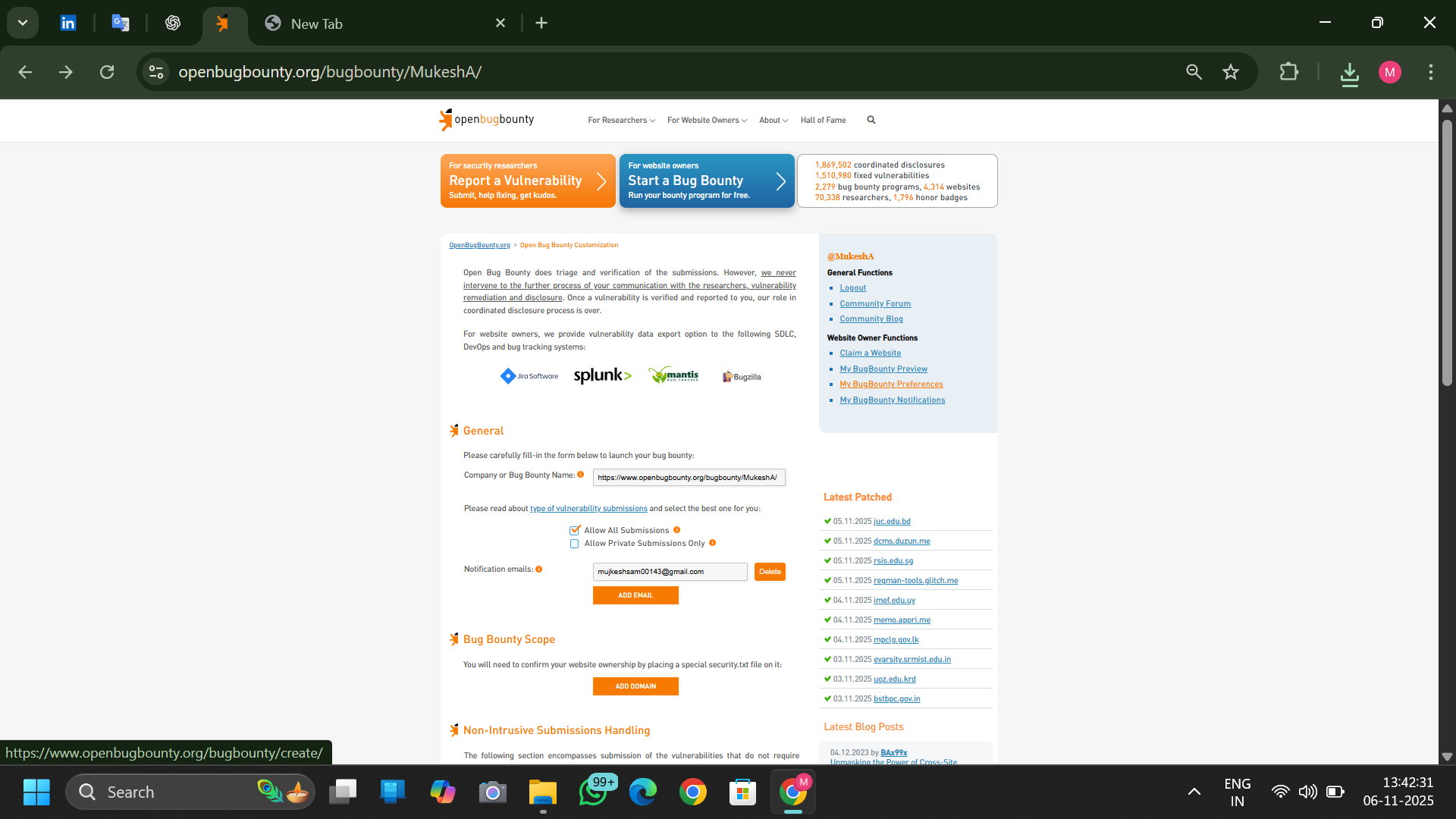
This document contains educational mockups and PoC screenshots related to the Bug Hunting on OpenBugBounty project. All images are included for demonstration and reporting purposes. Ensure any real testing is authorized before running scans or submitting findings.

## Embedded Images (Proof-of-Concept & Screenshots)

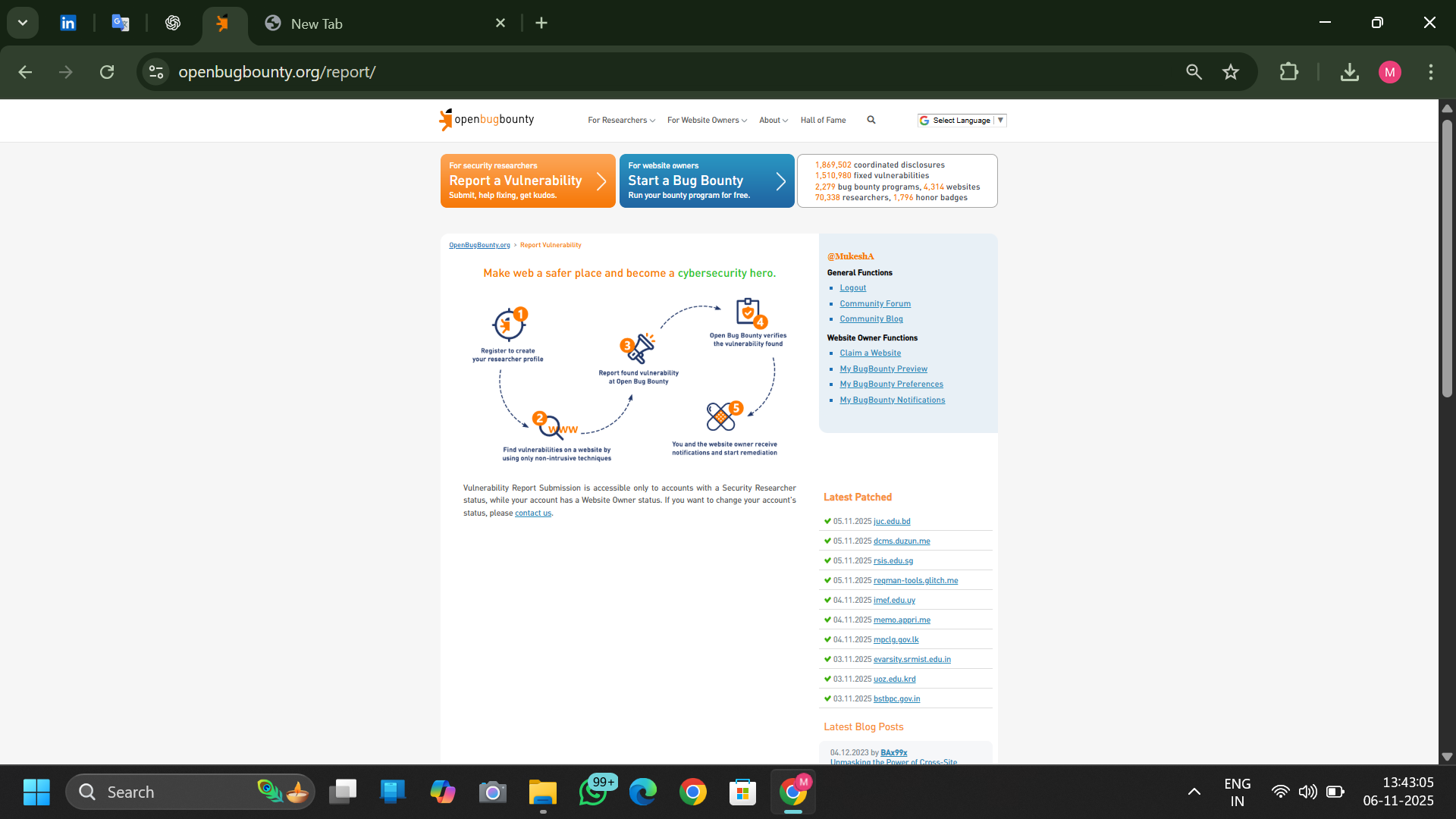
OpenBugBounty homepage / researcher dashboard (logged in)



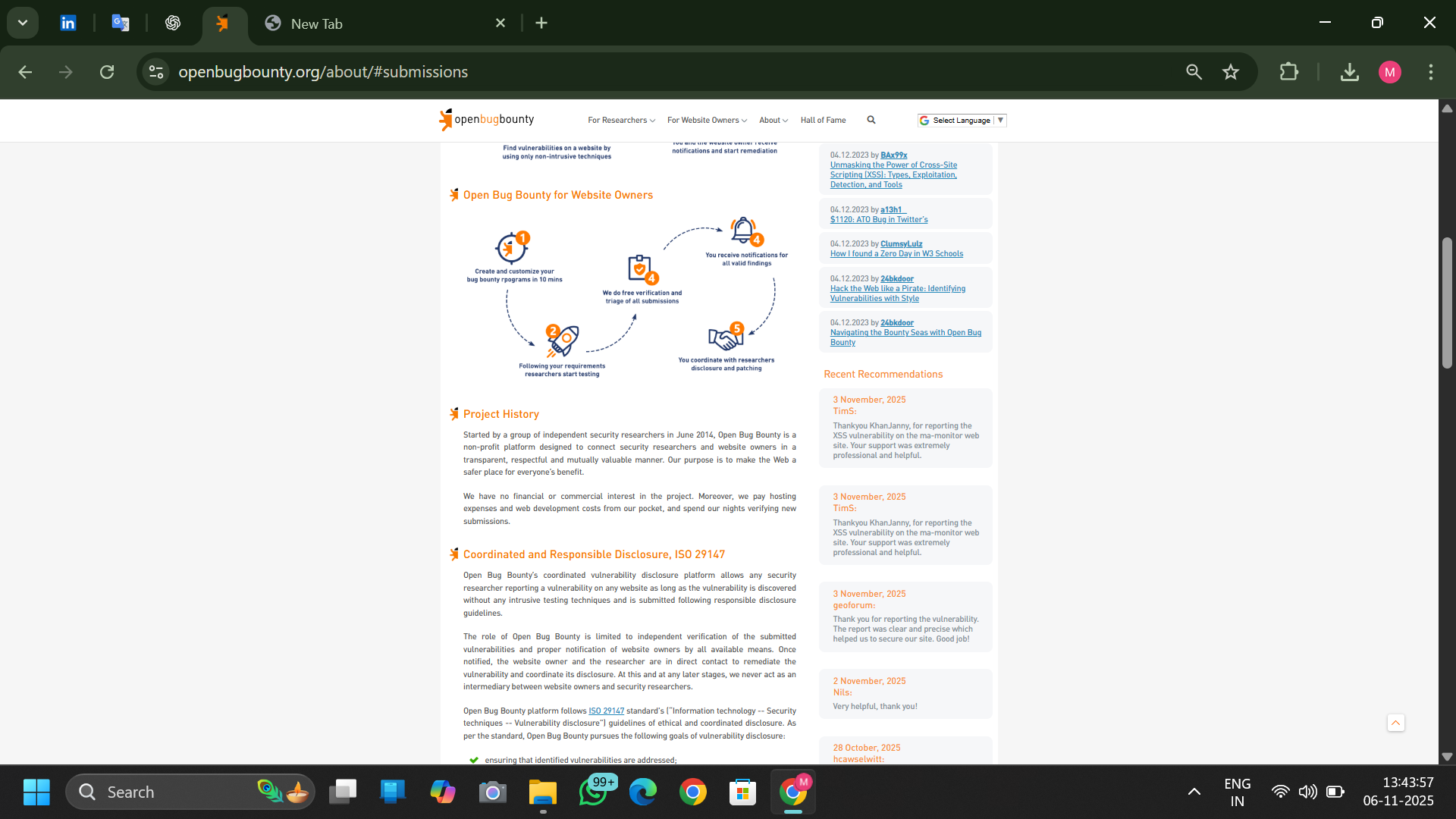
OpenBugBounty: Create bug bounty / settings page



OpenBugBounty: report page / submission guidance



OpenBugBounty: about / submissions guidance

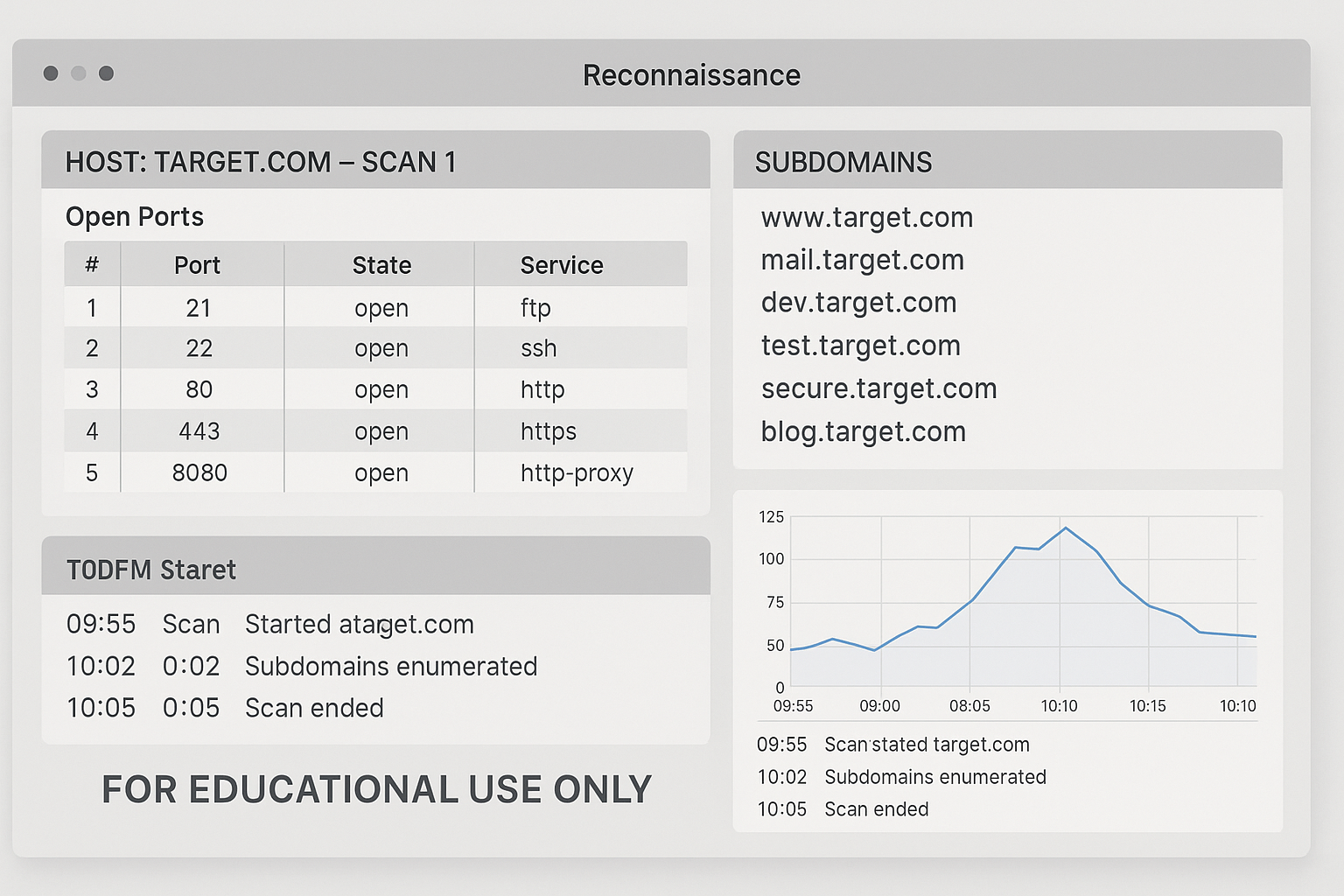


Reconnaissance mockup dashboard (educational),

This document contains educational mockups and PoC screenshots related to the Bug Hunting on OpenBugBounty project. All images are included for demonstration and reporting purposes. Ensure any real testing is authorized before running scans or submitting findings.



PoC: XSS alert payload in browser (educational)



Note: All screenshots and PoC images included in this document are for educational use and demonstration. Do not perform scans or tests on third-party websites without explicit authorization.

## Conclusion

This Major Project demonstrates a practical, ethical approach to web vulnerability research using widely adopted tools (nmap, Sublist3r, OWASP ZAP, Burp Suite) and a responsible reporting process via OpenBugBounty. The artifact set (document, mockups, and workflow diagram) is suitable for academic evaluation and professional training.

Prepared by: [ Mukesh.A ]

Date: [06/11/2025]