Checklist

1. Information Gathering
    a. Subdomain enumeration
        I. Sublis3tr
    b. Checking live hosts
        I. Httpx
    c. Checking status code
        I. httpstatus.io
    d. Recon
        I. Source code view
        II. Check JS files
        III. Way-back URL (web archive)
        IV. Google Dorking
        V. Bug Bounty Helper (https://dorks.faisalahmed.me/#)
        VI. GitHub Dorking

2. Critical File Vulnerability
    a. Dirb
    b. If got 403, then use 403 bypass (https://github.com/iamj0ker/bypass-403)

3. Host Header Injection

    Chain Bugs:

    a. HHI ----> Password Reset Poisoning
    b. HHI ----> Password Reset Poisoning via dangling markup (Host: target.com:'<a href="//bing.com/?)
    c. HHI ----> Web cache poisoning
        I. Web Cache Poisoning ----> XSS
        II. Web Cache Poisoning ----> Open Redirection
        III. Web Cache Poisoning ----> Open Redirection ----> Dos

    Payloads:

    I. Change the host header

        Host: vulnerable-website.com ----> evil-website.com

    II. Duplicating the host header

        GET /index.php HTTP/1.1
        Host: vulnerable-website.com
        Host: evil-website.com

III. Add host override headers

```
X-Forwarded-For: evil-website.com
X-Forwarded-Host: evil-website.com
X-Client-IP: evil-website.com
X-Remote-IP: evil-website.com
X-Remote-Addr: evil-website.com
X-Host: evil-website.com
```

IV. Add line wrapping

```
GET /index.php HTTP/1.1
 Host: vulnerable-website.com
Host: evil-website.com
```

V. Supply an absolute URL

```
GET https://vulnerable-website.com/ HTTP/1.1
Host: evil-website.com
```

4. Automation
    a. Nuclei

5. Rate limit
    a. Profile details change, login, forgot password, comment, share, report post, tag, sending friend request, register, contact form, Any form, 2fa submission
    b. Try rate limit bypass

6. Account Takeover
    a. Response Manipulation/Status code Manipulation
    b. Brute Force

7. 2FA BYPASS (https://www.youtube.com/watch?v=X2WfhBYQ2fY)
    a. Response Manipulation/Status code Manipulation
    b. Brute Force
    c. Token doesn't expire after usage
    d. Request 2 tokens from account A and B. Use the A's token in B's account
    e. Try to go directly to the dashboard URL without solving the 2FA. If not, success try adding the referral header to the 2FA page URL while going to dashboard
    f. Search the 2FA code in response
    g. Search the 2FA code in JS files
    h. CSRF/Clickjacking to disable 2FA

i. Request Manipulation
j. Enabling 2FA doesn't expire previous sessions

8. WordPress
    a. Automation
        I. WPScan
        II. CMSMap
        III. CMSScan
    b. XMLRPC
        I. SSRF - pingback.ping
            1. Port Scanning
            2. Origin Ip Found
        II. Bruteforce - wp.getUsersBlogs
    c. Application-level DOS via load-scripts.php

9. XSS
    a. Basic Payload
        I. Use Burp spider to find requests having parameters.
        II. Find vulnerable parameter and try payload there.
    b. Manual building
        I. If keyboard input become string response try to give input through mouse. E.g., onmouseover=alert(1);
        II. When we closing the input and it gets filtered out then build a payload which would not closed initially and it supposed to execute initially. E.g., <svg/onload=alert (1); here we have use; to break so no need of closing here.
        III. If input reflects as a plaintext, then use svg vector and if they filter some part like open parathesis or anything then try html entity encoder. E.g., we are giving <svg><script>alert(1)</script> but it reflects as <svg><script>alert1)</script> then try to encode "(" using html entity encoder (can use burp suite html encoder to encode it) after encoding "(" it will become "(" and final payload will be < svg><script>alert (1)</script>

            Summary: when something will filter you can convert that into html code so browser directly execute that

        IV. If input is reflecting in html comment, then close comment, then inject payload. E.g., --!><script>alert(1)</script>
    c. Automation
        I. Use Burp Suite's intruder and XSS payload file.
    d. XSS through host header injection
    e. XSS through file uploading.
    f. XSS through RFI (Whenever websites take URL as an input filed you can try to inject payload through a file)

g. Try to change self XSS to reflected.

10. URL Redirection
    a. Using common parameter list (continue, window, redirect, path, url, to, out, view, show, dir, navigation, domain etc.)
    b. URL Redirection on Path Fragments. Example:
        - Any.com/payloads
        - Any.com/bing.com
        - any.com//bing.com
    c. Use burp's intruder and open redirection payload list to automate open redirection

11. Parameter Tampering
12. Html Injection
13. File Inclusion
    a. LFI

       Possible Parameter ['file', 'document', 'folder', 'root' 'path' pg', 'style' pdf', 'template', p hp_path','doc']

       Automation: LFISuite [https://github.com/D35m0nd142/LFISuite]

    b. RFI

       Possible Parameter ['dest', 'redirect', 'uri', 'path', 'continue', 'url', 'window', 'next', data', 'reference', 'site', 'html', 'val', 'validate', domain', 'callback', return', page', 'feed' 'host', 'port', 'to' out', 'view', dir', 'show' 'navigation', 'open']

14. SPF record
    a. To check SPF record
        I. Go to - http://www.kitterman.com/spf/validate.html
        II. Or go to - https://mxtoolbox.com
    b. Exploitation

       Go to - https://anonymousemail.me/

15. Insecure CORS
    a. Insecure CORS through Response Header
        I. Search in Response: Access-Control-Allow-Origin
    b. Insecure CORS through Request Header
        I. Add header in request: Origin: http://evil.com
        II. Search in Response: Access-Control-Allow-Origin
    c. curl http://any.com -H "Origin: http://hackersera.com" -I

d. Conditions
    - POORLY IMPLEMENTED, BEST CASE FOR ATTACK:
        - Access-Control-Allow-Origin: https://attacker.com
        - Access-Control-Allow-Credentials: true
    - POORLY IMPLEMENTED, EXPLOITABLE:
        - Access-Control-Allow-Origin: null
        - Access-Control-Allow-Credentials: true

16. SSRF
    a. You have to find any parameter that may have some kind of external interaction or they can interact to external domain
    - Read file from server (file:///LFI_payloads)
    - Scan the Internal Network
    - SSRF with RFI

17. Source Code Disclosure
    a. Google dork
        I. Site:example.com index.of.backup
    b. Use intruder with critical file payload

18. CSRF
19. Burp Suite (Actively and Passively scan the host)
20. Intercom