---------- Start ----------

- This Is a collection of **Notes** i use as a reference in bug hunting .
- It is collected form many resources in the internet [Videos , Write-ups , Bug hunter Methodology's , and others] .
- The Main Recourse Is ChatGPT .
- Any one could share it or use it .
- Sorry For My Bad English .

==========

==========

[+] BBP_Checklist :

- [ ] Subdomain Enumeration
- [ ] Internet Search Engine Discovery (Shodan, Censys, etc.)
- [ ] Google Dorking
- [ ] Internals Subdomains / virtual hosts
- [ ] IP enumeration
- [ ] Port Scanning
- [ ] Wayback History
- [ ] Technology Fingerprinting
- [ ] Directory Enumeration
- [ ] Web Crawler & URL Discovery
- [ ] Endpoints / Parameter Fuzzing
- [ ] Hardcoded Information in JavaScript
- [ ] Search for API Endpoints
- [ ] Heat Mapping
- [ ] Manual Search

==========

[+] BBP_Recon

==========

General Notes

==========

```
RPS:
     5r/s >> delay 200ms(0.2sec) >> thread 1
```

```
# Burp Scope :
    .*\.example\.com$

# Zap Scope :
    https?://[a-zA-Z0-9_~.-]{1,}.example.com.*
```

```
---> Subdomains To Start With

info / api /  dev  / admin / stage / dashboard

uat / jira / prod / stg / alph / beta / corp / sandbox /...

- cat live-subs.txt | grep -E
"portal|login|admin|signin|dashboard|test|internal|dev|developer|signup|r
egister|panel"
```

```
Endpoint → Path →  /api/users/123
```

==========

Domain Enumeration

==========

Acquisitions :

- www.crunchbase.com (Find Acquisitions)

Reverse WHOIS :

```
whois example.com
```

```
amass intel -whois -d example.com
```

ASNs :

- Manual:
  - http://bgp.he.net
  - https://bgpview.io
  - https://ipinfo.io

```
    # Validate ASN Ownership**
    whois AS<number>

    # Get IP Prefixes Owned by the ASN
    whois -h whois.cymru.com " -v AS<number>"
    curl ipinfo.io/AS<number>

    # Active Scanning of Live Hosts
    masscan -p1-65535 <ip_range> --rate=1000
    naabu -iL ip_ranges.txt -top-ports 1000 -o naabu_common.json
```

```
echo ASXXXXX | asnmap | naabu -p 443 | httpx
```

```
amass intel -active -asn <asn-no>
```

```
echo X.0.84.0/24 | /root/go/bin/tlsx -san -cn -silent -resp-only |
/root/go/bin/dnsx -silent | /root/go/bin/httpx

echo X.X.X.0/24 | /root/go/bin/tlsx -san -cn -silent -resp-only |
/root/go/bin/dnsx -silent | /root/go/bin/httpx -td -sc -fr -title -ip -rl
10 -t 2 -r /root/resolvers.txt -random-agent -o alive-subs-ASN.txt
```

Shodan Dorking :

```
org:"Intigriti"

asn:AS1234
```

```
net:192.168.43/24, 192.168.40/24

http.status:200 org:"Intigriti"

ssl.cert.subject.CN:"intigriti.com"

http.favicon.hash:<favicon_hash>

org:<company> http.component:php

org:<company> http.title:Login,Log in,Register,Signin, Sign in, Sign up

org:<company> http.title:"Index of"

org:<company> http.status:200,404 -port:80 -port:443 -port:8080 -
port:8443

org:<company> product:jenkins
```

Google Dorking :

```
site:   filetype:jsp
site:   filetype:php
site:   filetype:asp

site:*.example.com intitle:"index of /"

site:   inurl:&

site:   inurl:login  -www

site:   inurl:register  -www

site:   inurl:signup  -www

site:  inurl:admin -www

site:  -www  inurl:api
```

```
site:  inurl:v1 -www

site:  inurl:/app -www
```

==========

Subdomain Enumeration

==========

Crt.sh:

```
curl -s https://crt.sh/\?q\=\domain.com\&output\=json | jq -r '.
[].name_value' | grep -Po '(\w+\.\w+\.\w+)$' | tee subdomains.txt
```

Brute-forcing:

```
shuffledns -mode bruteforce -d <domain> -w <wordlist> -r <resolvers> -o
<output>
```

Alteration:

```
# used with large orgs
shuffledns -d <domain> -w <permutations-wordlist> -r <resolvers> -o
<output>
```

ASNs :

- Manual:
  - http://bgp.he.net

- https://bgpview.io
- https://ipinfo.io

```
# Validate ASN Ownership**
 whois AS<number>

# Get IP Prefixes Owned by the ASN
whois -h whois.cymru.com " -v AS<number>"
curl ipinfo.io/AS<number>

# Active Scanning of Live Hosts
masscan -p1-65535 <ip_range> --rate=1000
naabu -iL ip_ranges.txt -top-ports 1000 -o naabu_common.json
```

- Automated:

```
echo ASZZZZ | asnmap | naabu -p 443 | httpx
```

```
amass intel -active -asn <asn-no>
```

```
echo x.x.x.x/24 | /root/go/bin/tlsx -san -cn -silent -resp-only |
/root/go/bin/dnsx -silent | /root/go/bin/httpx
```

Subfinder :

```
/root/go/bin/subfinder -rl 50 -all -r /root/resolvers.txt -d $domain  |
tee -a subs.txt
```

Amass :

```
/root/go/bin/amass enum --passive -r /root/resolvers.txt -d $domain | tee
-a subs.txt

/root/go/bin/amass enum -active -r /root/resolvers.txt -norecursive -
noalts -d $domain | tee -a subs.txt
```

Internals Subdomains/virtual hosts :

- locate internals subdomains [internal-api's , devops , tools , ..]

```
cat subs.txt | xargs -I{} host {} | tee host-out.txt

dirsearch -u -e -H 'X-Forwarded-For: 127.0.0.1'

curl -H 'Host: 127.0.0.1' url
```

Shodan :

- look for different ASNs from httpx output and use shodan
  to look for that subnet .
- Ex:

```
    Search > X.X.X.X/24
    or
    Search > asn:ASNZZZZ
    or
    earch > asn:ASNZZZZ "title:login"
        // look for apps on those ips
```

Censys :

```
copy page  > \b[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}
nano r-ips.txt
cat r-ips.txt | httpx -sc -mc 200
```

Live Subdomain Validation (DNS Resolution) :

```
/root/go/bin/httpx -td -sc -fr -asn -title -ip -rl 10 -t 2 -r
/root/resolvers.txt -p 8000,8080,8443,443,80,8008,3000,5000,9090,900 -
random-agent -H "Referer: https ://domain.com" -o alive-subs-HTTPX.txt
```

```
-l

/root/go/bin/httpx -l live-subs.txt -p
8080,8443,8000,8888,8081,8181,3306,5432,6379,27017,15672,10000,9090,5900
-threads 80 -title -sc -cl -server -ip -o services-ports.txt
```

```
dnsx -l active_subs.txt -resp -t 10 -o resolved_subs.txt
dnsx -resp-only -silent
```

IP enumeration

- Internet scanning :
  - https://en.fofa.info/
  - censys.com
  - shodan >> hostname:grab.com
  - HTTPX:

```
cat alive-subs-HTTPX.txt  | grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" |
sort -u | tee ips-online.txt
```

```
- Massdns:
```

```
massdns -r /root/resolvers.txt -t A -o S -w massdns-list.out subs.txt

cat massdns-list.out | awk '{print $3}' | sort -u | grep -oE "\b([0-9]
{1,3}\.){3}[0-9]{1,3}\b" > ips-online.txt
```

Ports Enumeration/Scanning :

- `masscan`

```
masscan -iL ips-online.txt -p0-65535 --rate 10  --randomize-hosts  -oL
masscan.out
```

- `naabu`

```
/root/go/bin/naabu -host x.x.x.x/24

# Scan entire internal subnet and save results
naabu -host x.x.x.x/24 -o internal_ports.txt
# Scan specific port ranges
naabu -host x.x.x.x/24 -p 80,443,8000-9000 -o internal_ports.txt

/root/go/bin/naabu  -c 2 -rate 10 -exclude-ports 25  -top-ports 1000 -r
/root/resolvers.txt -l ips-online.txt -o naabu-ports-scan.txt

/root/go/bin/naabu  -c 2 -rate 10 -top-ports 1000 -r /root/resolvers.txt
 -nmap-cli 'nmap -sV -sC' -o naabu_result.txt

/root/go/bin/naabu -c 2 -rate 10 -exclude-ports 25 -top-ports 1000 -r
/root/resolvers.txt -host X.X.X.X/24 -o naabu-ports-scan.txt
```

- nmap

```
nmap -A -sV -sC -Pn --max-rate 50
nmap -sS -A -PN -p- --script=http-title site.com
nmap --script vuln -sV -sC --top-ports 1000 -T2 -iL
nmap -iL ips.txt -sSV -A -O -Pn -F -oX nmap.xml
nmap -sV --top-ports 1000
nmap --data-length 30
nmap --script ssl-cert -p 443 <subdomain/ip>
```

JavaScript Analysis

```
/root/go/bin/katana -jsl -jc -rl 10 -p 1 -rd 1 -c 2 -u

/root/go/bin/katana  -rl 10 -c 1 -f qurl -u

cat f-urls.txt | sort -u | grep -a ".js$" |/root/go/bin/httpx  -mc 200 -r
/root/resolvers.txt  -random-agent -threads 2 -rl 10  | tee  live-js-
urls.txt

/root/go/bin/nuclei  -t /root/nuclei-templates/http/exposures/tokens -rl
10 -bs 2 -c 2 -l live-js-urls.txt -o nuclei_exposures.txt
```

Archive Enumeration

Archive URL Collection

```
gau --subs target.com | anew archived_urls.txt
gau --threads 2 --blacklist ttf,woff,svg,png,jpg,gif,mp4,jpeg
http://url.com/ --o urls.txt

cat domain.txt | uro | grep -E '\.
(xml|json|pdf|sql|txt|zip|tar|gz|tgz|bak|7z|rar|log|cache|secret|db|backu
p|yml|gz|config|csv|yaml|md|md5|exe|dll|bin|ini|bat|sh|tar|deb|rpm|env|dm
g|tmp|crt|pem|key|pub|asc)'
```

```
waybackurls target.com | anew wayback_urls.txt
```

```
/root/go/bin/urlfinder -f
woff,css,png,svg,jpeg,gif,jpg,woff2,swf,ico,tif,tiff,pdf,svg,webp,avif,ap
k  -all -rl 10  -d mars.com -o mars.com.txt
```

```
waymore -i <domain> -mode U -mc 200 --no-subs -oU
waymore -i http://vulnweb.com -mode R --no-subs -oR  waymore-Respons.txt
```

Content Discovery

Feroxbuster

```
feroxbuster -u https://target.com -w /usr/share/wordlists/common.txt -r -
t 20 -o recursive_results.txt

feroxbuster --random-agent -k -E -B --scan-limit 2 -t 1 --rate-limit 10 -
d 2 -w -u
```

FFuF

```
ffuf -u https://target.com/FUZZ -w
/usr/share/wordlists/content_discovery.txt -mc 200,403 -t 1 -p "0.1-0.3"
-r -rate 10 -recursion-depth 3 -o ffuf_results.txt
```

```
ffuf -u http://127.0.0.1:42000/api/FUZZ  -t 1 -p "0.1-0.3" -r -rate 10 -w
/root/without_dots.txt
```

Gospider

```
/root/go/bin/gospider -s "http://testphp.vulnweb.com/" -c 2 -d 5 --
blacklist ".(jpg|jpeg|gif|css|tif|tiff|png|ttf|woff|woff2|ico)"

/root/go/bin/gospider -s "http://127.0.0.1:3000/" -c 2 -K -d 5 --
blacklist .(ico|css|svg) -o go-s-out.txt
```

Dirsearch

```
dirsearch --random-agent --max-rate=10 -t 2 -e
php,asp,aspx,jsp,html,zip,jar,sql,log,db,backup,json,gz,rar,conf -u

dirsearch -i 200 -e
php,bak,old,zip,tar.gz,txt,log,conf,json,asp,jsp,aspx,yml,yaml,rar

dirsearch --random-agent --max-rate=10 -t 2 -H 'X-Forwarded-For:
127.0.0.1' -u

dirsearch -e xml,json,sql,db,log,yml,yaml,bak,txt,tar,gz,zip -x
403,404,500,400,502,503,429 --random-agent -u
```

Web Crawler & URL Discovery

Katana

```
katana -jsl -jc -rl 10 -p 1 -rd 1 -c 2 -u

katana -u subdomains-alive.txt -jsl -jc -rl 5 -p 1 -rd 1 -c 2 -d 5  -kf
-fx -ef woff,css,png,svg,jpeg,gif,jpg,woff2 -o  allurls.txt
```

Gospider

```
gospider -s "https://target.com" -d 2 -o gospider_output/
```

```
gospider -s "https://target.com" -c 2 -K -d 5 --blacklist .(ico|css|svg)
-o gospider_output/
```

Hakrawler

```
echo "https://target.example.com" | hakrawler -depth 2 -plain -js -out
hakrawler_results.txt
```

Parameter Discovery

Arjun Parameter Discovery

```
arjun -u "https://target.example.com" -m GET,POST --stable -w /burp-
params -o params.json
```

Katana

```
katana -d 5 -rl 10 -c 2  -f qurl -u
```

FFuF Parameter Bruteforce

```
ffuf -u https://target.com/page.php?FUZZ=test -w
/usr/share/wordlists/params.txt -o parameter_results.txt
```

Arjun Parameter Discovery

```
arjun -u "https://target.example.com" -m GET,POST --stable -o params.json

arjun -m GET,POST -d 1 --rate-limit 10 --stable  -u $domain  -oT arjun-
params.txt
```

ParamSpider Web Parameters

```
paramspider.py --domain https://cpcalendars.cartscity.com --exclude
woff,css,js,png,svg,php,jpg --output g.txt
```

Testing with Header Manipulation

---

1. Inspect Server's Expected Headers

```
    # Command to Fetch All Headers:
    curl -k -I https://x.x.x.x/

   Look for headers indicating:
    - Authentication requirements (WWW-Authenticate, Authorization).
    - Application-specific headers like x-*
    - Security policies (Content-Security-Policy,Strict-Transport-
Security).
```

2. Common Header Manipulation Techniques

```
A. Test Origin and Referrer Headers
```

```
    # Some servers use Origin or Referer headers for access control.
Altering these can bypass 403 restrictions.
    curl -k -H "Origin: https://x.x.x.x" -H "Referer:
https://x.x.x.x/upload" https://x.x.x.x/

    - Origin: Mimics requests from a whitelisted domain.
    - Referer: Indicates the originating page
```

```
B. Try Host Header Injection
```

```
    # Tests for virtual host misconfigurations, which may redirect you to
hidden admin panels or alternative services.

    curl -k -H "Host: admin.x.x.x.x" https://x.x.x.x/
```

3. Test Authentication-Bypass Scenarios

```
    #  A. Add Authentication Headers
    curl -k -H "Authorization: Bearer example-token" https://x.x.x.x/
    curl -k -u "admin:password" https://x.x.x.x/

    #  B. Manipulate Cookies
    curl -k -H "Cookie: sessionid=test; auth=admin" https://x.x.x.x/
```

4. Bypass IP Restrictions with Proxy or Forwarding Headers

```
curl -k -H "X-Forwarded-For: 127.0.0.1" -H "X-Real-IP: 127.0.0.1"
https://x.x.x.x/
```

5. Test for Debug or Development Headers

```
    # Debug headers might trigger verbose error messages or expose
sensitive information.

    curl -k -H "X-Debug: true" -H "X-Dev-Mode: 1" https://x.x.x.x/
```

6. Automate Header Testing with Tools

```
ffuf -u https://x.x.x.x/ -H "FUZZ: value" -w
/usr/share/wordlists/headers/common.txt
```

WAFs

---

waf00f tool

Automation Scanning Tools

Nuclei :

```
/root/go/bin/nuclei -t /root/nuclei-templates/http/exposures/tokens -rl
10 -bs 2 -c 2 -l live-js-urls.txt -o nuclei_exposures.txt

/root/go/bin/nuclei -rl 10 -bs 2 -c 2 -dast -list live-kat-params.txt -o
nuclei_output_vulnerabilities.txt

cat  urls/param-urls.txt | /root/go/bin/gf sqli | /root/go/bin/nuclei -rl
10 -bs 2 -c 2 -dast -tags sqli -o nuclei_sqli.txt

/root/go/bin/nuclei -tags xss,lfi,sqli -u

/root/go/bin/nuclei -rl 10 -bs 2 -c 1 -dast -tags xss,sqli,lfi,cmdi,ssrf
-l

/root/go/bin/nuclei -rl 10 -bs 2 -c 2 -as critical,high,medium -u

# With specific templates
```

```
nuclei -u http://internal-target -t misconfiguration/

tags cves
tags exposure,disclosure,misconfig
tags default-login
tags wordpress

http/exposures/tokens/ js
```

IIS :

```
shortscan -F -V uri

sns -u $host
```

Cloud Enumeration :

```
cloud_enum -k target.com -b buckets.txt -o cloud_enum_results.txt
```

S3 Bucket Access Test

```
aws s3 ls s3://<bucket_name> --no-sign-request
```

S3 Bucket Content Dump

```
python3 AWSBucketDump.py -b target-bucket -o dumped_data/
```

Dalfox :

```
/root/go/bin/dalfox --waf-evasion  url  -p

/root/go/bin/dalfox  -w 10 --skip-grepping --skip-headless --skip-mining-
all --skip-xss-scanning file param-urls.txt

cat valid-Urls.txt | /root/go/bin/gf xss | sed 's/=.*/=/' | sort -u |
```

```
/root/go/bin/dalfox  pipe
```

SQLi :

```
sqlmap --banner --random-agent --batch --level=3 --
tamper="between,randomcase,space2comment" -u

sqlmap -m sqli.txt  --banner --random-agent --batch --level=3 --
tamper="between,randomcase,space2comment"

sqlmap -r req.txt --string "string when true "

sqlmap -r req.txt -p login --level 3 --dbs --batch

sqlmap -u --data="foo=bar&foo2=bar2"

sqlmap -u --dbs --forms --crawl=2
```

Subdomain Takeover :

```
/root/go/bin/subjack -w 404-list.txt -ssl -t 2 -timeout 30 -o 404-
results.txt -v -c /root/go/pkg/mod/github.com/haccer/subjack@v0.0.0-
20201112041112-49c51e57deab/fingerprints.json
```

==========

[+] BBP_After_Recon

[*] Based On Status Code :

1. 200 status code subdomains

1. **Service identification** :
    1. port scanning
    2. Wappalyzer
2. **Enumerate Directories and Files :**

[/admin, /config, /backup.zip, or /db.sql.]

3. **Enumerate endpoints, parameters**
4. **Questions to ask** :
    1. How does the app handle special chars <>"/
    2. How does the site reference a user
    3. Are there multiple user roles
5. **App Feature analysis :**
    - File Handling [File Upload, File Inclusion]
    - Broken **Access Control**
    - Broken **User Authentication** [OAuth / SSO / 2FA]
    - Injection (SQL, No-SQL, Command)
    - Security Miss-configuration
    - Business logic flaws
    - Other

---

2. 404 status code subdomains

1. Test for Subdomain Takeover [Subjack or Subzy]
2. Enumerate Hidden Directories and Files [custom 404 pages]

---

3. 403 status code subdomains

1. Enumerate Directory and File Access (e.g., /admin, /backup.zip)
2. Check for HTTP Header-Based Restrictions
   [Test bypasses by modifying headers like :]
   1. User-Agent
   2. Referer
   3. Origin
3. Test for Method-Based Access
4. Look for Path Traversal Vulnerabilities
5. Bypass Using Alternate Encoding

==========

[*] Heat Mapping :

1. **Content Types** :
    - Look for multipart-forms [Shell, injections, ++]

- Look for content type XML [XXE]
- Look for content type json [API vulns]

2. **APIs** :
   [Broken Auth ,BAC , mass assignment ,SQLi , SSRF , File Upload, Data Exposure ,Command / Template Injection]
3. **Account Section** :
   - Profile [stord XSS]
   - App Custom Fields [stord XSS, SSTI]
   - Integrations [SSRF, XSS]
4. **Paths or URLs passed as values** : [SSRF, Redirs]
5. **Uploads Functions** :
   - Self Uploads
     - XML Based (Docs/PDF) [SSRF,XSS,XXE]
     - Image [XSS,Shell] {name, Binary_header , Metadata}
   - Where is data stored? [s3 perms]
6. **File download** :
   [Path traversal, info disclosure]
7. **Search bars** :
   [Reflected XSS, sql injection]

==========

[*] Mapping Web Application :

1. Information Gathering
   1. Understand Role and Permission Hierarchies:
      - Identify user roles (e.g., guest, user, admin).
      - Note actions and resources available to each role. [CRUD]
   2. Identify Entry Points:
      - APIs, endpoints, and URL patterns.
      - Admin panels, dashboards, or restricted resources.
2. File Access Vulnerabilities
   1. Directory Traversal
   2. Exposed File
3. CORS Misconfigurations
4. API-Level Testing
   1. Verify Access Controls on API Endpoints
   2. Reuse or modify JWT tokens to test for improper validation
5. Session and Cookie Testing
   1. Session Fixation

    2. Cookie Manipulation [change roles or bypass restrictions (isAdmin=false → isAdmin=true)]
    3. Replay Attack [Resend valid session cookies to test for persistent access]
6. Testing Unauthenticated Access
    1. Direct Resource Access [Attempt accessing restricted URLs without logging in.]
    2. IDOR (Insecure Direct Object References)
    3. Force Browsing
7. Role and Privilege Escalation Testing
    1. Horizontal Privilege Escalation
    2. Vertical Privilege Escalation

==========

[*] Mapping SPA Application :

SPA frameworks ?

> A **SPA framework** refers to a framework used to build **Single Page Applications (SPA)**. In a SPA, the entire application runs on a single HTML page and dynamically updates content without reloading the whole page. This leads to faster, smoother user experiences similar to desktop apps.

Key Characteristics of SPAs :

- Only one HTML page is loaded initially.
- Navigation and content updates happen dynamically via JavaScript.
- Often use AJAX or Fetch API to communicate with the backend.
- Uses client-side routing to change views without full-page reloads.

How To Approach:

1. For SPA framework (React, Vue, EmberJS, AngularJS)
2. Open web browser then >> **Inspecting the Network Tab in Developer Tools**
3. Search for interested endpoints [admin , api , ..]
4. If the target is promising then Choose one option :

A. All at once :

```
1. Download Js Files
2. Analyze_JS-files
3. params , fuzz
4. check for securtity bugs
```

B. Only interested ones :

```
- Network > Fetch/XHR > if interested req >> copy as curl

1. [ curl ] + ' -x http://127.0.0.1:8080 -k '
2. then try to test the api from the proxy.
3. Alter the path by remove folders backward then check the response
4. Alter the HTTP Methods
5. check the Dev Tools again if any new ajax req.
6. Test again and remove unnecessary headers if u
    get 400 bad req as stats code & change the Method also
7. Use FFuf to Fuzz the API for params or endpoints

-  waybackmachine website >> to find enpoints and params
```

==========

[*] JavaScript Enumeration :

[+] GOALS

1. **Discover Hidden APIs**
2. **Find Admin Panels & Private Routes**
3. **Authentication tokens**
4. Endpoints
5. parameters
6. hardcoded credentials
7. expired domain names
8. postmessage misconfigurations

[+] Download JS Files

```
1.
gau target.com | grep -E "\.js$" | tee js_files.txt

for url in $(cat js_files.txt); do wget "$url"; done

2.
echo "example.com" | waybackurls | grep ".js"
```

```
    3.
    wget -r -l 1 -H -D example.com -A .js https://example.com
        ///
        - `-r`: Recursive download
        - `-l 1`: Limit recursion depth to 1 level
        - `-H`: Enable downloading from external domains
        - `-D example.com`: Restrict downloads to the target domain
        - `-A .js`: Only download files with `.js` extension
        ///


    gospider -s "http://testphp.vulnweb.com/" -d 10 --blacklist ".
(jpg|jpeg|gif|css|tif|tiff|png|ttf|woff|woff2|ico)"


    katana -jsl -jc -rl 10 -d 5 -u https://example.com

    Linkfinder.py
```

[+] Check for hard-coded tokens:

```
    ## Check for hardcoded tokens:
    grep -Ei "token|api[_-]?key|auth|secret" *.js

    cat js-files.txt | gf apikey > secrets.txt

    httpx -ms 'apikey' -l all_js_links.txt -t 100
```

(access_key|access_token|admin_pass|admin_user|algolia_admin_key|algolia_api_key|alias_pass|alicloud_access_key|amazon_secret_access_key|amazonaws|ansible_vault_password|aos_key|api_key|api_key_secret|api_key_sid|api_secret|api.googlemaps AIza|apidocs|apikey|apiSecret|app_debug|app_id|app_key|app_log_level|app_secret|appkey|appkeysecret|application_key|appsecret|appspot|auth_token|authorizationToken|authsecret|aws_access|aws_access_key_id|aws_bucket|aws_key|aws_secret|aws_secret_key|aws_token|AWSSecretKey|b2_app_key|bashrc password|bintray_apikey|bintray_gpg_password|bintray_key|bintraykey|bluemix_api_key|bluemix_pass|browserstack_access_key|bucket_password|bucketeer_aws_access_key_id|bucketeer_aws_secret_access_key|built_branch_deploy_key|bx_password|cache_driver|cache_s3_secret_key|cattle_access_key|cattle_secret_key|certificate_password|ci_deploy_password|client_secret|client_zpk_secret_key|clojars_password|cloud_api_key|cloud_watch_aws_access_key|cloudant_password|cloudflare_api_key|cloudflare_auth_key|cloudinary_api_secret|cloudinary_name|codecov_token|config|conn.login|connectionstring|consumer_key|consumer_secret|credentials|cypress_record_key|database_password|database_schema_test|datadog_api_key|datadog_app_key|db_password|db_server|db_username|dbpasswd|dbpassw

ord|dbuser|deploy_password|digitalocean_ssh_key_body|digitalocean_ssh_key_ids|docker_hub_password|docker_key|docker_pass|docker_passwd|docker_password|apikey|dockerhub_password|dockerhubpassword|dot-files|dotfiles|droplet_travis_password|dynamoaccesskeyid|dynamosecretaccesskey|elastica_host|elastica_port|elasticsearch_password|encryption_key|encryption_password|heroku_api_key|sonatype_password|awssecretkey)

[+] Search for API Endpoints

Look for :
- Parameters
- Admin panels(`/admin`, `/dashboard` , `/internal` , `/manage`,`/debug`)
- `api.target.com`
- `/v1/users`
- `/admin/login`
- `/api/private/`
- GraphQL endpoints like :
`fetch("https://target.com/graphql", {...})`
`query {, mutation {, introspectionQuery`

```
# 1. From hostes:
grep -i ngix hosts  | cut -d ' ' -f1 | httpx -ms '.chunk.' -title

grep -i ngix hosts  | cut -d ' ' -f1 | httpx -ms 'main.js' -title

# 2. If Downloaded Js Files:
grep -Eo "https?:\/\/[a-zA-Z0-9./?=_-]*" *.js | sort -u | tee
api_urls.txt

grep -E "Route|path|navigate|redirect|push|location.href" *.js

grep -iE 'v[1-9]|api|admin|chunk' *.js

cat all-gospider | grep -E '\.js' | tr '[] ' '\n' | grep -E '\.js' | anew
all-js-links

grep -rnaiE 'chunk\.' go-reslt | awk '/.js$/{print $NF}' | grep -E
'^http'

cat hosts | grep -vE site.com  | grep -vE '.com|www' | sort -u | httpx -
path '/api' -content-type | grep -i json

grep -oE '"/rest/[^"]+"|/api/[^"]+' js.beautify --color
```

```
grep -rnaiEo '.{1,20}'await.{,40}\.(get|post|put|delete|patch)(.{1,50}'
*/ |grep -E '\.(get|post|put|delete|patch)' | tr ' ' '\n' | grep '"/' |
cut -d '"' -f2 | sed 's#^#url/s#g'
| httpx -sc -title -cl -proxy http://127.0.0.1:8080 -t 40 -x POST -body
'{}'


# 3. From The DevTools
```

==========

[*] Mapping API :

Methodology :

1. Enumerating API endpoints.
2. Finding parameters and HTTP methods.
3. Exploiting an API Vulnerability :
    - **Mass Assignment**
    - **Broken Authentication**
    - **Broken Access Control (BAC)**
    - **SQL Injection (SQLi)**
    - **File Upload Vulnerabilities**
    - **Command Injection / Template Injection via Parameters**
    - **Server-Side Request Forgery (SSRF)**
    - **Excessive Data Exposure** [API returns full user object with passwords or tokens.]

Summary :

```
 Steps :

1. /root/go/bin/katana -u http://127.0.0.1:42000 -d 5 -jc -jsl

2. signup if posibole then login

3. use Authorization: Bearer

4. Fuzz and look for params in js for endpoints
     - ffuf -u http://127.0.0.1:42000/api/FUZZ  -t 1 -p "0.1-0.3" -r -
rate -w /root/without_dots.txt
```

==========

General Notes :

```
- URL Query
https://test.com/channels/2257/widgets/9861
https://test.com/?channels=2257&widgets=9861
```

```
# Creat Wordlist suitable for api
    grep -vE '^\.' /usr/share/seclists/Discovery/Web-Content/raft-small-words-lowercase.txt > without_dots.txt
```

```
# sometimes the response were cached so remove this header
    If-None-Match: aaaaaaaaaaaaaaa
```

```
echo $USER1_TOKEN | cut -d '.' -f2 | base64 -d
```

```
Admin panels :

 (e.g., `/admin`, `/dashboard` , `/internal` , `/manage` ,`/debug`)
```

==========

Authentication Mechanisms

1. API Key Authentication

```
GET /api/data HTTP/1.1
Host: api.example.com
Authorization: Api-Key abc123xyz456

GET /api/data?api_key=abc123xyz456
```

2. Basic Authentication

```
GET /api/data HTTP/1.1
Host: api.example.com
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

3. **Bearer Token

```
GET /api/userinfo HTTP/1.1
Host: api.example.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

4. OAuth 2.0 Authorization Code Flow

Use Case: Web apps that access APIs on behalf of a user (e.g., Google, Facebook logins).

Flow:

1. Redirect user to provider for login.
2. User logs in and authorizes.
3. Provider redirects back with a code.
4. App exchanges code for a token.

```
POST /oauth/token  HTTP/1.1
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=abc123&redirect_uri=https://yourapp.com/callback
```

How To Approach :

1. Register a New User via `curl`

when you're working with **web APIs like Juice Shop**, figuring out what an endpoint like `http://localhost:3000/api/Users` requires (e.g., fields, formats) can be done in a few key ways:

```
curl -X POST http://localhost:3000/api/Users \ -H "Content-Type: application/json" \ -d '{   "email": "user1@example.com",   "password": "Test1234",   "passwordRepeat": "Test1234",   "securityQuestion": { "id": 1,     "question": "Your eldest sibling’s middle name?", "createdAt": "2020-01-01",     "updatedAt": "2020-01-01"   },  "securityAnswer": "test" }'
```

1. Use Burp Suite (Proxy or Repeater)

1. Register a user in the browser with Burp running.

2. Intercept the registration request.
3. Examine:
    - Method (POST)
    - Headers (like `Content-Type`)
    - JSON body (required fields)

This shows you **exactly what the endpoint expects**, and you can copy that to use in your `curl`.

2. Check Browser Dev Tools

1. Open your browser's **Developer Tools** → **Network Tab**.
2. Register a user in the UI.
3. Look at the `POST` request to `/api/Users`.

You'll see:

- All required fields (`email`, `password`, `securityQuestion`, etc.)
- Data format

3. Use Trial and Error with `curl`

Start with minimal required fields:

```
curl -X POST http://localhost:3000/api/Users \ -H "Content-Type: application/json" \ -d '{"email":"test@example.com","password":"Test1234","passwordRepeat":"Test1234"}'
```

If it fails, you'll get a helpful error like:

```
{   "error": "Missing required property: securityQuestion" }
```

`Note:`
Juice Shop gives **verbose error messages** — perfect for learning

4. Check API Docs or Source

- You can browse source code on GitHub.
- Look in `routes/user.js` or the models to see validation rules.

---

2. Login via `curl`

```
curl -s -X POST http://localhost:42000/rest/user/login -H "Content-Type: application/json" -d '{"email":"user01@example.com","password":"12345"}' -v
```

---

3. Enumerating API endpoints :

```
feroxbuster -u http://url/api/v1/admin/  -m GET,POST,PUT,PATCH --scan-limit 2 -t 1 --rate-limit 10 -d 2 --random-agent
```

```
ffuf -w raft-med-lowerca -H 'user-agent: zzzzz-zzzz' -u http://url/api/FUZZ
```

```
output :
            login,signin,signup,....
```

4. Finding param's and HTTP methods :

```
ffuf -t 1 -p "0.1-0.3" -rate 10 -u https://url/api/PATH -X METHOD -w /path/to/wordlist:PATH -w /path/to/wordlist:METHOD
```

Use arjun

```
arjun.py -u https://url/api/users --post -o result.json -x http://127.0.0.1:8080
```

Examples:

```
# Examples:

curl -X POST https://url/v1/events/1/registr -d ' '
seq 1 100 | ffuf -X POST -w -:FUZZ -u https://url/v1/events/FUZZ/registr
```

```
ffuf -x POST -w raft-med-lowerca -H 'user-agent: zzzzz-zzzz' -u
https://url/backend/gestes?FUZZ -r -d '{}'

ffuf -w routes:ROUTES  -x POST -w raft-med-lowerca -H 'user-agent: zzzzz-
zzzz' -u https://url/ROUTES/gestes -r -d 'FUZZ=1' -H 'Content-type:
application/x-www-form-urlencoded' -x http://127.0.0.1:8080

ffuf -w raft-med-lowerca -H 'user-agent: zzzzz-zzzz' -u
https://url/admin/v1/FUZZ -X POST -d '{}' -H 'Content-type:
applicaton/json'
```

5.Make API Requests via DevTools Console

GET request:

```
fetch('http://127.0.0.1:42000/api/products').then(response =>
response.json()).then(data => console.log(data)).catch(error =>
console.error('Error:', error));
```

``

POST request:

```
fetch('https://example.com/api/data', {   method: 'POST',   headers: {
'Content-Type': 'application/json',     'Authorization': 'Bearer
YOUR_TOKEN_HERE'   },   body: JSON.stringify({ key1: 'value1', key2:
'value2' }) }) .then(response => response.json()) .then(data =>
console.log(data)) .catch(error => console.error('Error:', error));
```

How To Test :

Bypass authentication

- Try accessing endpoints with no token, invalid tokens, and tokens from other users.
- Replay, modify, or forge tokens (e.g., JWT, API keys).

Broken Access Control

- Broken **Access Control** [Access someone else's basket]
- Broken Object **Level Authorization** [changing the ID in Repeater to another number (e.g., `GET /rest/basket/2`)]
- **Function-Level** BAC flaw [bypass Functionality that for Admin Access only.]
- Parameter Pollution: Send duplicate parameters in API calls to bypass restrictions.
- Mass Assignment: Object Property Level Authorization (OPLA)

Business Logic Testing

- Can a user submit a payment of $0.01 instead of $100?
- Can a user modify the `user_id` in the request body to get other users' data?

Injection vulnerabilities (SQL, NoSQL, Command)

- Fuzz API parameters for injection vulnerabilities

Old API versions (Less secure)

API Debug Endpoints

Debug endpoints in APIs are routes that developers use during the development phase to inspect application behavior, view logs, test features, or simulate errors. These are usually not intended for production environments because they can reveal sensitive information or provide access to internal application mechanics.

How Debug Endpoints Typically Look

They can appear in various forms depending on the framework, but common patterns include:

- `/debug`
- `/api/debug`
- `/admin/debug`
- `/status` or `/healthcheck` (sometimes debug-like)

- `/__debug__` (Python/Django style)
- `/console`, `/inspect`, `/trace`, `/dump`

They might expose:

- Application logs
- Configuration variables
- Environment variables
- Internal state (e.g., cache contents, memory usage)
- Endpoint tracing
- Mock user sessions

How to Test for Debug Endpoints

1. **Enumerate Routes:** Use tools like `ffuf`, `dirsearch`, or `gobuster`:

```
ffuf -u http://target.com/FUZZ -w /usr/share/wordlists/dirb/common.txt -e .php,.json,.html
```

2. **Look for Suspicious or Developer-focused Routes:** Watch for:
   - Routes returning detailed errors
   - Routes allowing config/state inspection
   - Any unauthenticated access to sensitive data
3. **Send Requests with Verbose Headers:** Use `curl` or `Burp Suite`:

```
curl -i http://target.com/debug
```

4. **Fuzz for Parameters or Secrets:** Try common query strings:

```
?debug=true
?test=1
?env=dev
?verbose=1
/api/configuration?_debug=true
```

5. **Examine Headers and Responses for Clues:** Look for headers like `X-Debug-Token`, or verbose error messages.
6. **Use Burp Intruder to Fuzz for Common Debug Triggers:**

```
console
inspect
trace
dump`
status
healthcheck
__debug__
_debug
debugtrace
verbose
dev
test
staging
admin
```

Security Risks of Debug Endpoints

- **Information Disclosure:** Exposing stack traces, source code, or configuration.
- **Authentication Bypass:** Sometimes debug endpoints lack proper access control.
- **Privilege Escalation:** May allow simulating or assuming other user roles.

Mass Assignment :

Modern frameworks encourage developers to use functions that automatically bind input from the client into code variables and internal objects. Attackers can use this methodology to update or overwrite sensitive **object's properties** that the developers never intended to expose.

Object Property Level Authorization : Even if the user owns the object, they can manipulate **fields they shouldn't**.

Example Attack Scenarios

Scenario #1

A ride sharing application provides a user the option to edit basic information for their profile. During this process, an API call is sent to `PUT /api/v1/users/me` with the following legitimate JSON object:

```
{"user_name":"inons","age":24}
```

The request `GET /api/v1/users/me` includes an additional credit_balance property:

```
{"user_name":"inons","age":24,"credit_balance":10}
```

The attacker replays the first request with the following payload:

```
{"user_name":"attacker","age":60,"credit_balance":99999}
```

Since the endpoint is vulnerable to mass assignment, the attacker receives credits without paying.

Scenario #2

A video sharing portal allows users to upload content and download content in different formats. An attacker who explores the API found that the endpoint `GET /api/v1/videos/{video_id}/meta_data` returns a JSON object with the video's properties. One of the properties is `"mp4_conversion_params":"-v codec h264"`, which indicates that the application uses a shell command to convert the video.

The attacker also found the endpoint `POST /api/v1/videos/new` is vulnerable to mass assignment and allows the client to set any property of the video object. The attacker sets a malicious value as follows: `"mp4_conversion_params":"-v codec h264 && format C:/"`. This value will cause a shell command injection once the attacker downloads the video as MP4.

How To Prevent

- If possible, avoid using functions that automatically bind a client's input into code variables or internal objects.
- Whitelist only the properties that should be updated by the client.
- Use built-in features to blacklist properties that should not be accessed by clients.
- If applicable, explicitly define and enforce schemas for the input data payloads.

JWT

- Structure
  HEADER . Body . Sign
- Attacks:

1. Crack Password

```
crackdone.py -k jwt -t "ey.ey.sign" -w rockyou.txt
```

2. None Alg attack [Not Common in Real Scenarios]

```
1. We don't know the password so change
    {"alg":"HS256","typ":"JWT"}
    {"alg":"None","typ":"JWT"}

2. Edit the Body
3. Delet the sign but live the .
    > ey.ey.
```

----------The End----------