Chapter 4

Mapping the Application:

Application mapping is the first step in analysing or attacking an application. It involves identifying all visible and hidden features to understand how the application works. After this, the application's behaviour, security controls, and used technologies are examined to find possible weak points. This helps determine the main attack surface and where to focus further testing for vulnerabilities.

Enumerating Content and Functionality:

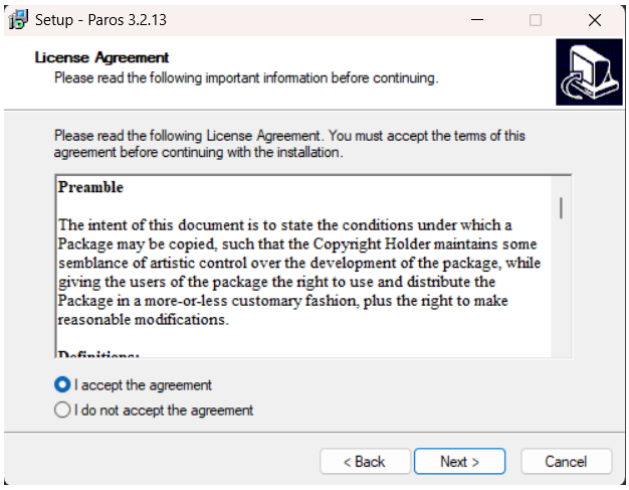
In most applications, most content and features can be found by manually browsing from the main page and following all links and multi-step processes like registration or password reset. A site map, if available, helps in listing the content. However, to fully and accurately inspect and record everything in the application, more advanced techniques are needed beyond simple manual browsing.

Web Spidering:

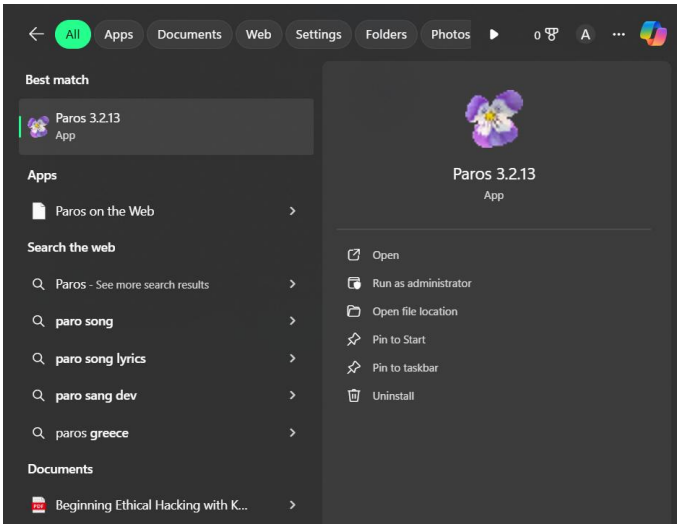
Web spidering uses automated tools to crawl a web application by following links, submitting forms, and sometimes analysing JavaScript to discover content and functionality. Tools like Paros, Burp Spider, and WebScarab help map applications more thoroughly than manual browsing and may even use URLs from robots.txt. However, automated spidering has limitations: it can miss complex JavaScript navigation, fail at multistep forms due to validation, misunderstand form-based navigation, loop endlessly on dynamic URLs, and struggle with authentication and session handling. In some cases, spidering can be dangerous, as it may trigger sensitive or destructive functions if proper access controls are missing.

Web Spidering using Paros:

Install the Paros:



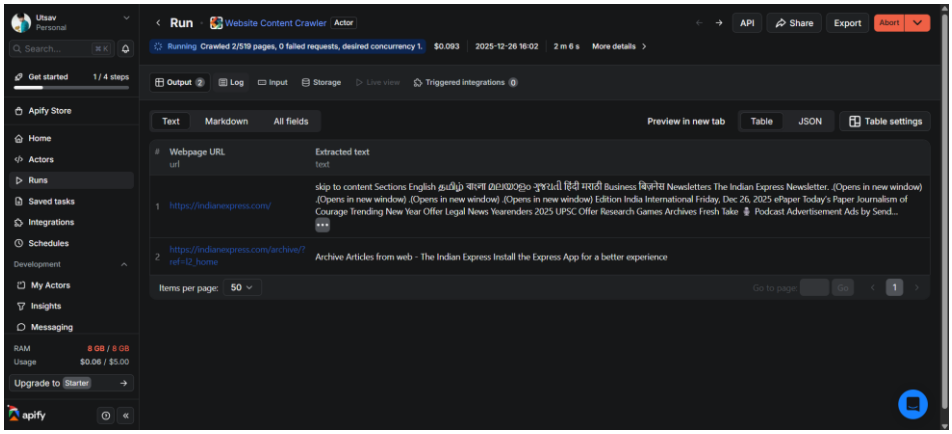
Open the app:



Following screen will appear: click on the 'accept' button.

And proceed accordingly…! (It didn't open on my device)

Similarly, we can use:



How to stop Web Spidering:

In cybersecurity / web application security, you can limit or stop web spidering (crawling) using these controls (brief & practical):

1. Robots.txt (soft control)

- Add disallow rules to /robots.txt
- Only stops compliant crawlers (not attackers)

2. Authentication & Authorization

- Require login (session, OAuth, mTLS)
- Enforce role-based access control
- Prevent unauthenticated crawling

3. Rate Limiting

- Limit requests per IP / user / token
- Thwarts automated enumeration and scraping

4. Web Application Firewall (WAF)

- Detect bot signatures and abnormal crawl patterns
- Block:
    - High request rates
    - Directory brute-force
    - Known bad user agents

5. CAPTCHA / Bot Challenges

- Use CAPTCHA or JS challenges on sensitive endpoints
- Effective against non-human automation

6. Disable Directory Listing

- Prevent exposure of file structures
- Common spider target

7. User-Agent & Behavior Analysis

- Block suspicious or empty user agents
- Monitor traversal patterns (e.g., sequential URL probing)

8. IP Reputation & Geo Blocking

- Block known bot networks, proxies, TOR
- Apply geo-fencing where appropriate

9. Obfuscation (limited value)

- Non-predictable URLs
- Not security by itself, only a minor deterrent

User-Directed Spidering:

User-directed spidering is a controlled way of mapping a web application where a human uses a normal browser to navigate the site, while a security tool quietly watches and records everything in the background. As the user clicks through pages, submits forms, and logs in, the proxy/spider tool captures requests and responses and builds a detailed map of the application. This approach works better than fully automated spidering because the user can handle complex navigation, meet input validation rules, stay logged in, and safely avoid triggering dangerous actions—while still allowing the tool to discover all available pages and functions.

Widely Used Tools

- **Burp Suite**
    - Proxy + crawler + site map
    - Industry standard for web app security testing
    - Supports authenticated and complex workflows
- **OWASP ZAP (Zed Attack Proxy)**
    - Open-source alternative to Burp
    - Manual browsing + spidering via proxy
    - Good for beginners and labs
- **WebScarab**
    - Older OWASP tool
    - Proxy-based spidering during manual navigation
    - Less commonly used today but conceptually relevant

Discovering Hidden Content:

Many applications contain hidden content or features that are not directly visible or linked from the main pages. This can include old or backup files, testing features, configuration files, or functions meant for other user roles like administrators. Attackers who find this hidden content may use it to gain extra access or exploit security weaknesses. Finding such content usually needs both automated tools and manual checking, and sometimes just luck.

Some techniques are:

- Brute-Force Techniques: Brute-force techniques automatically guess common page and directory names and analyse server responses to find hidden or restricted content in a web application.
- Inference from Published Content
- Use of Public Information
- Leveraging the Web Server

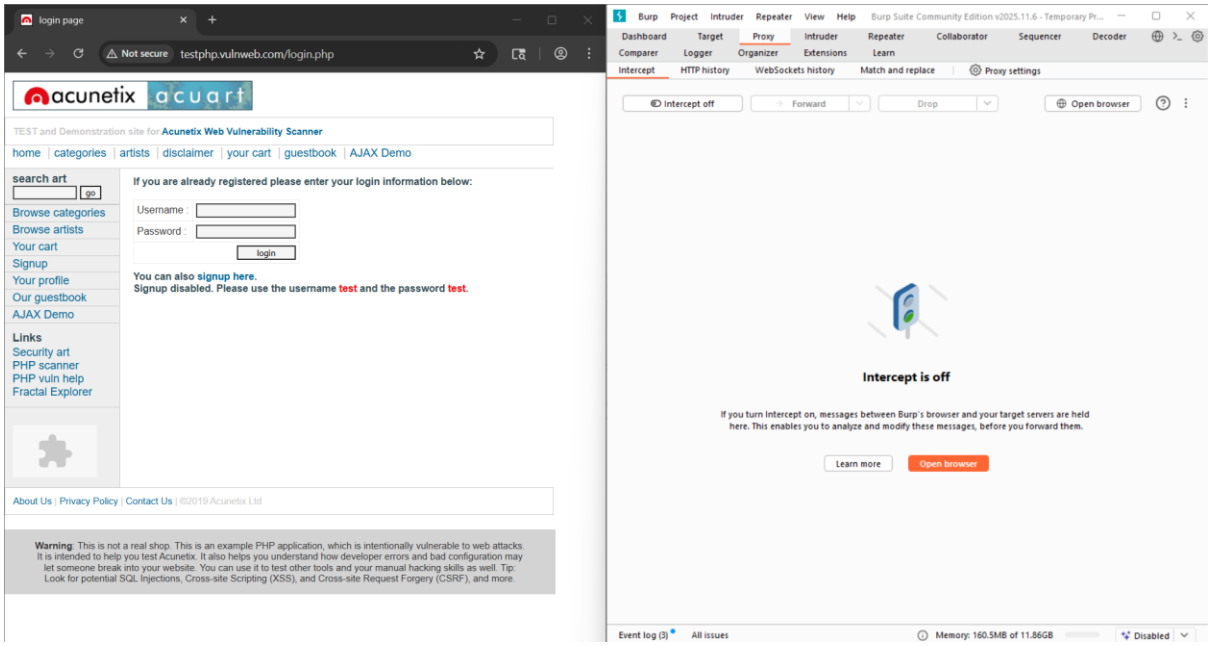
Brute-Force Techniques:

Brute-force techniques use automated tools to discover hidden pages and directories in a web application by guessing common names and sending large numbers of requests to the server. Since applications respond differently to valid and invalid requests, attackers must analyze response codes and patterns rather than relying only on “page found” or “not found” messages. Effective discovery

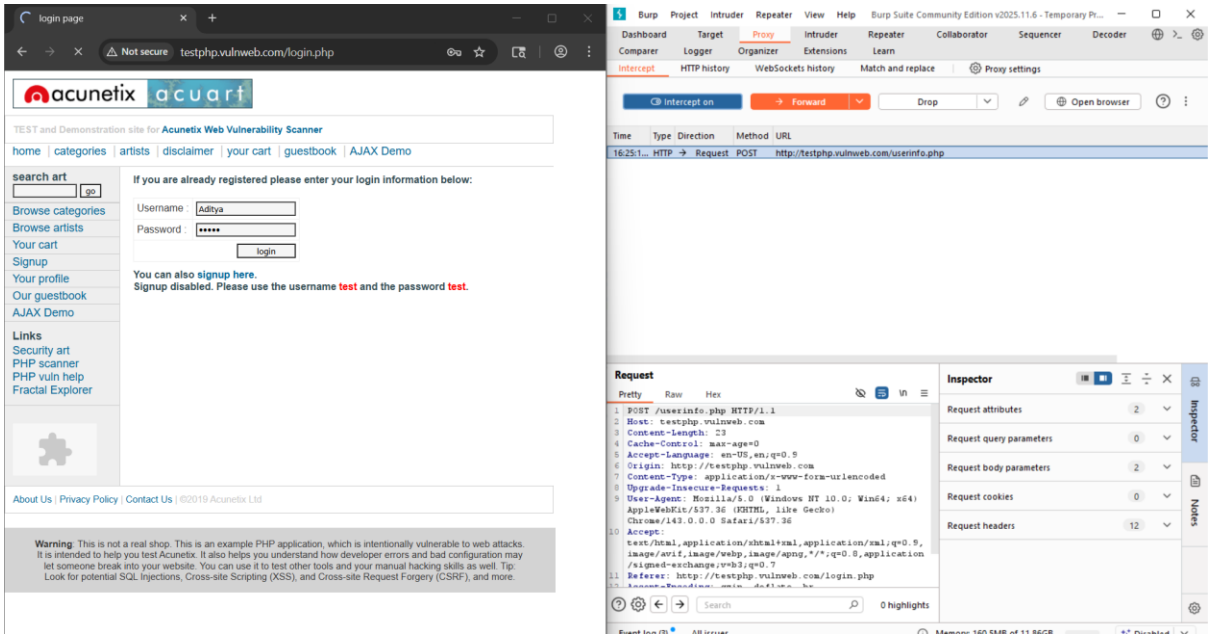
combines automation with manual review of server responses to identify hidden or restricted content.

Brute forcing password using Burp Suite:

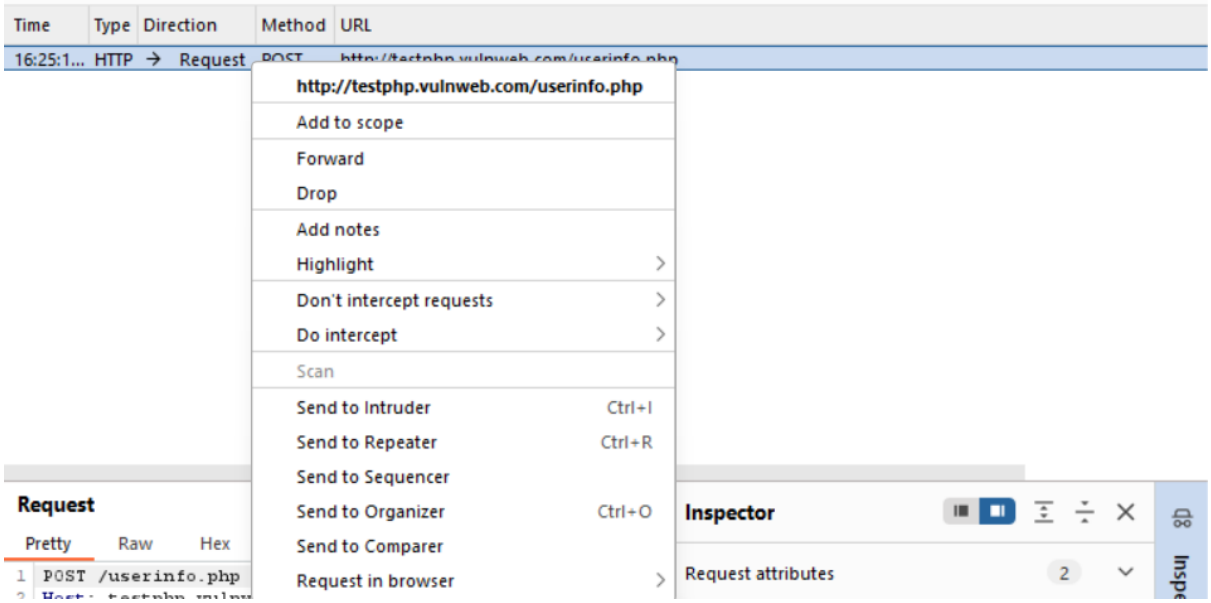
Step1: Open burp suite and the login page (keep intercept off).



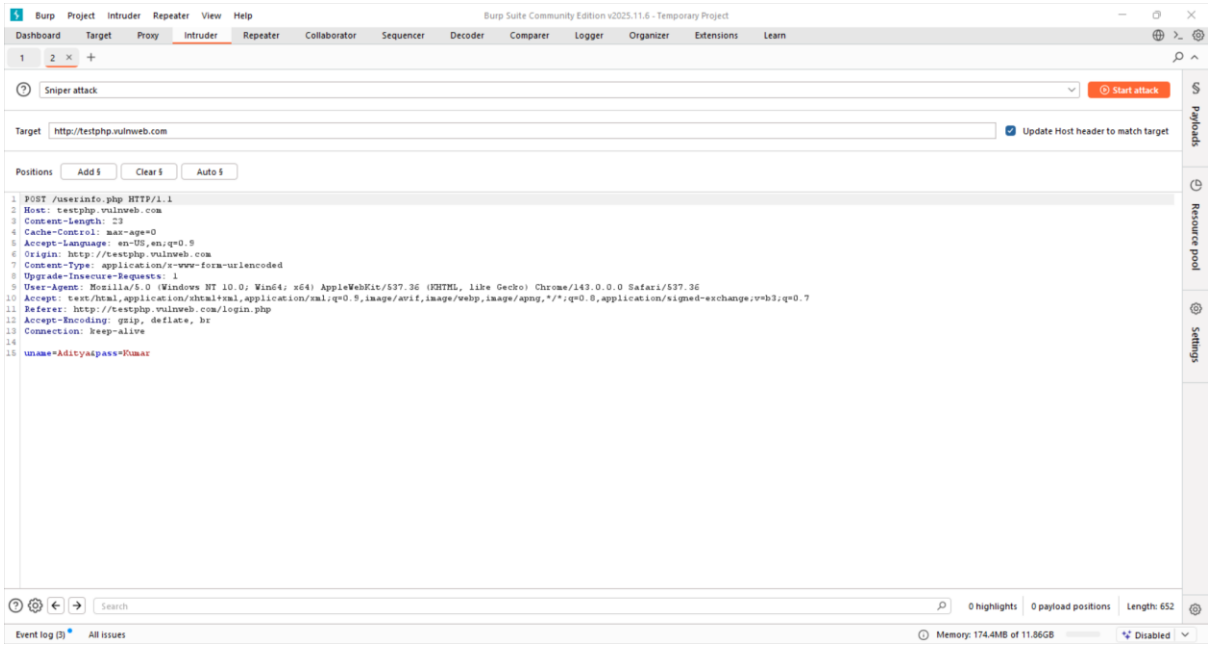
Step2: Fill the details while keeping the intercept off. Following thing can be seen in the “Request”.



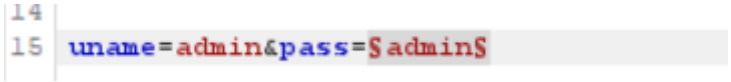
Step3: Click on the request, select the “Send to intruder”.



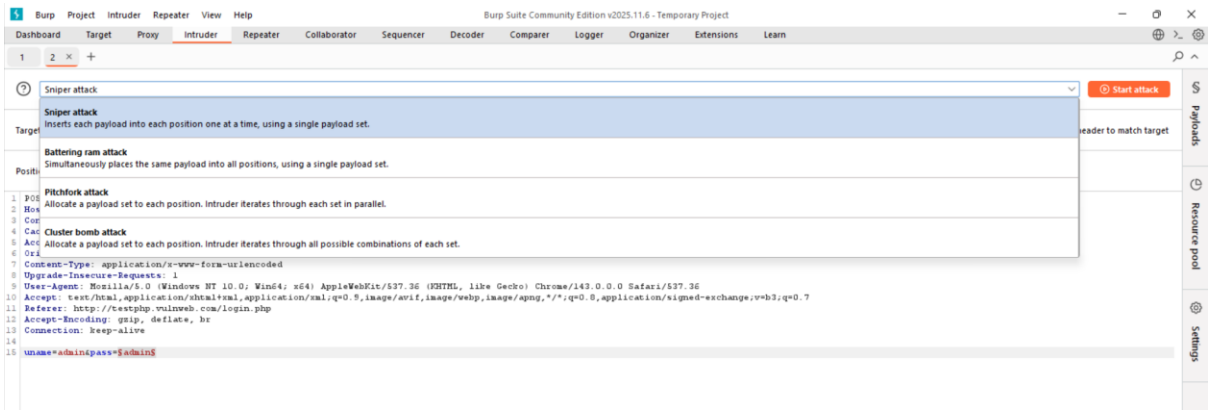
Step4: click on the “intruder” tab. The request can be seen here. Click on the “Clear$” button to clear the $ signs.



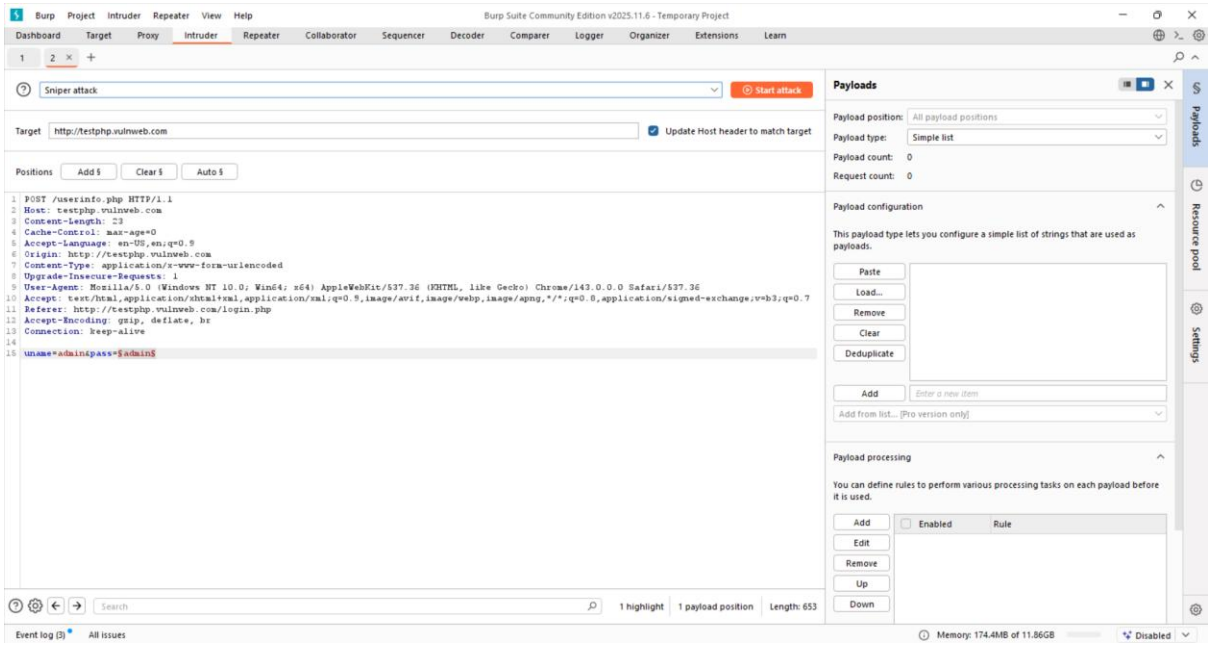
Step5: Now, specifically select the ‘admin’ and click on the “Add$” to add the $ sign.



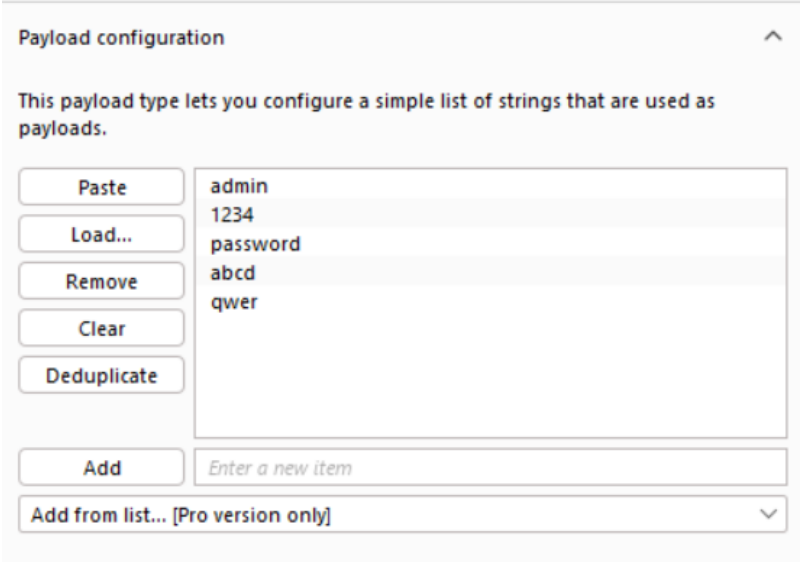
Step6: Select the type of attack from the top. We select with “sniper attack”.



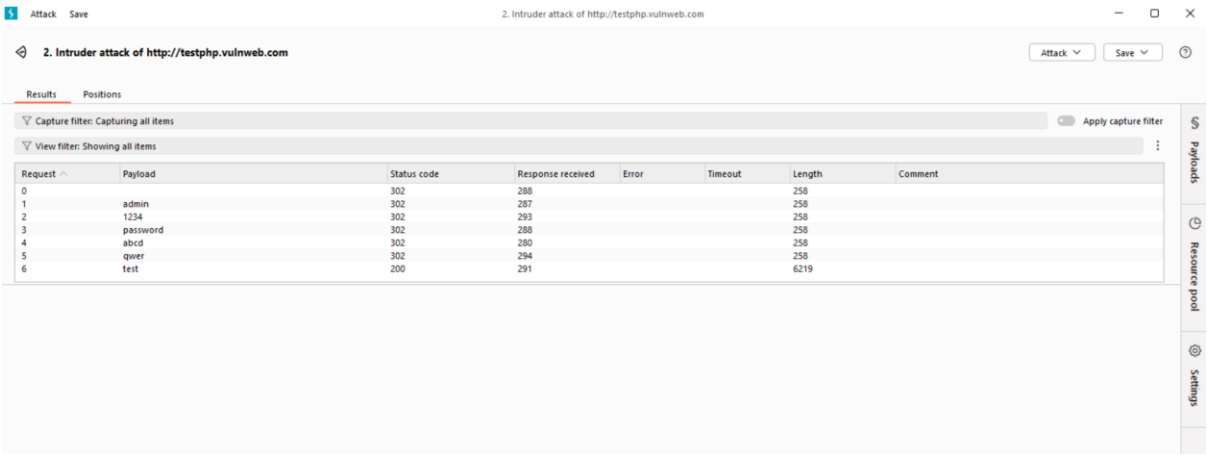
Step7: Click on the “Payloads” section from the right menu. Following will appear.



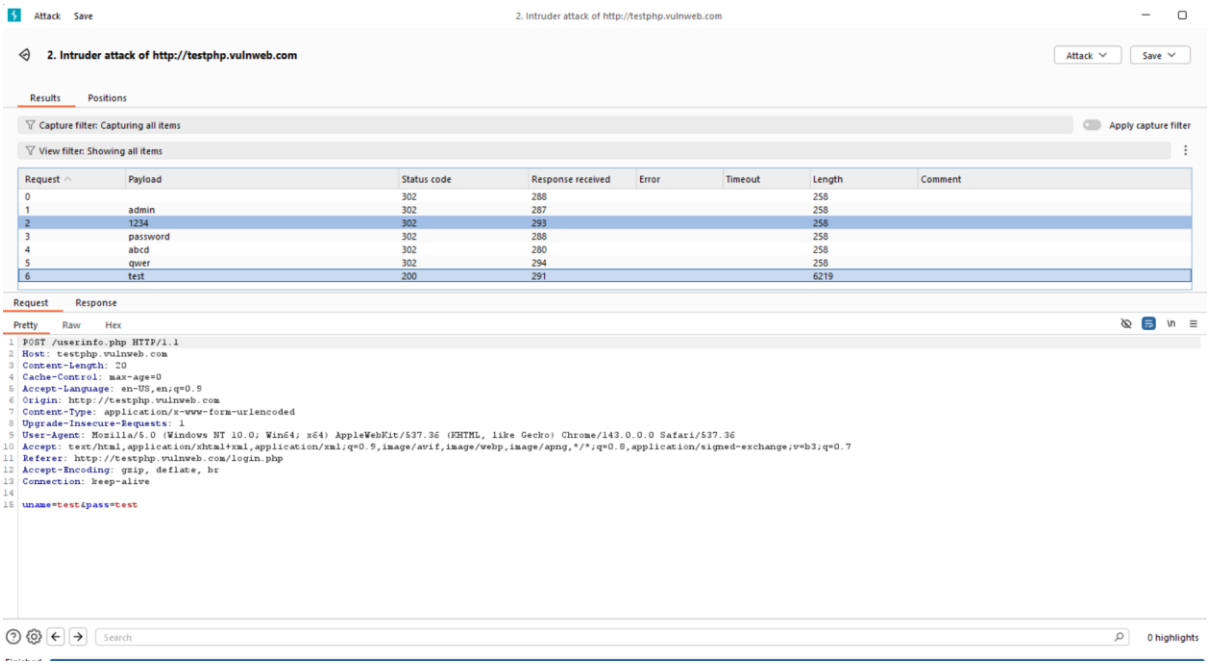
Step8: Add some passwords.



Step9: Click on “start attack” button to start the attack.

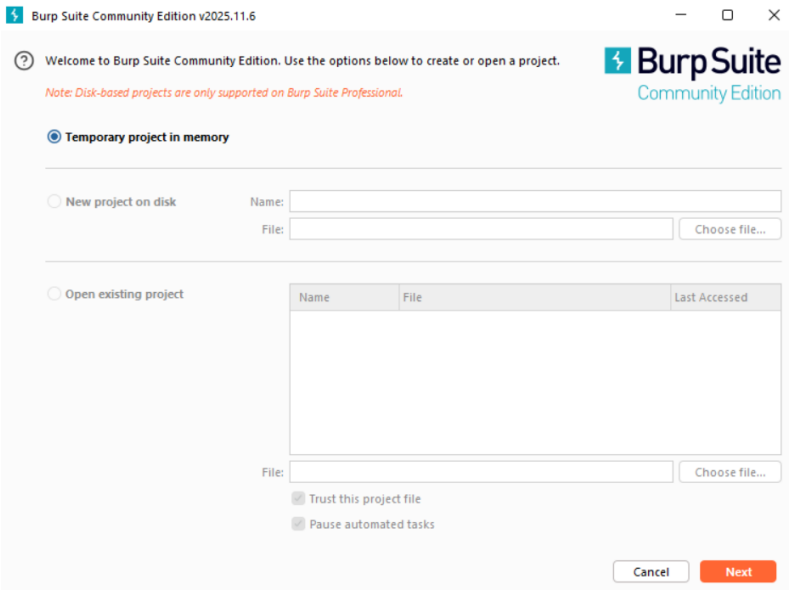


Step10: see for each the length is same, except for one where the payload is “test”, so basically it means that brute force gave the password for user name “test” as “test”.

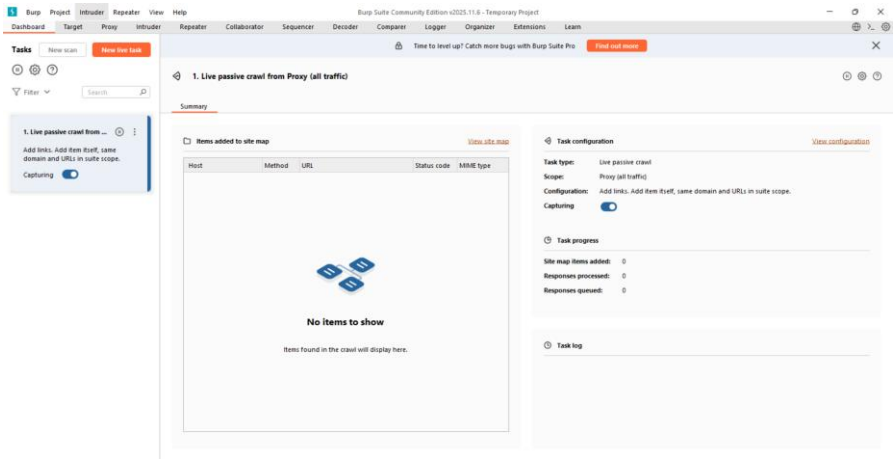


Brute spidering using Burp Suite:

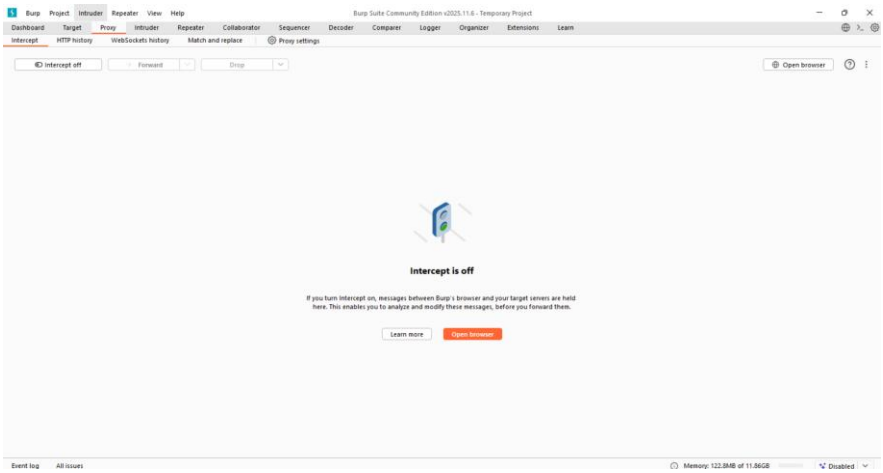
Step1: open the Burp suite



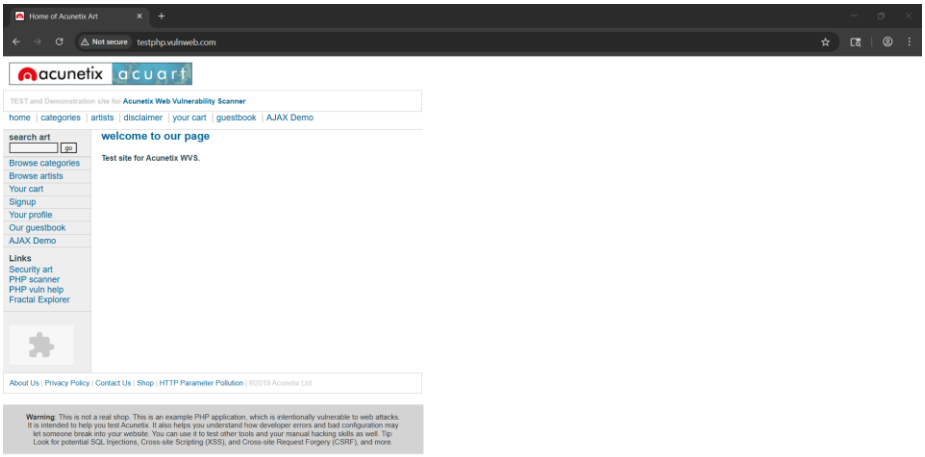
Step2: Following screen will appear.



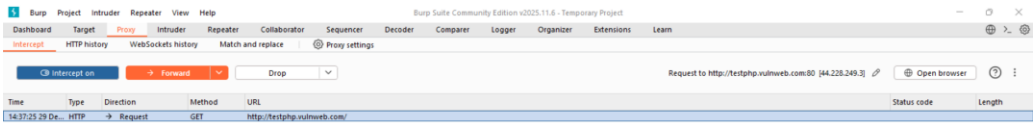
Step3: Go to the 'proxy' section, and click on the "open browser" button.



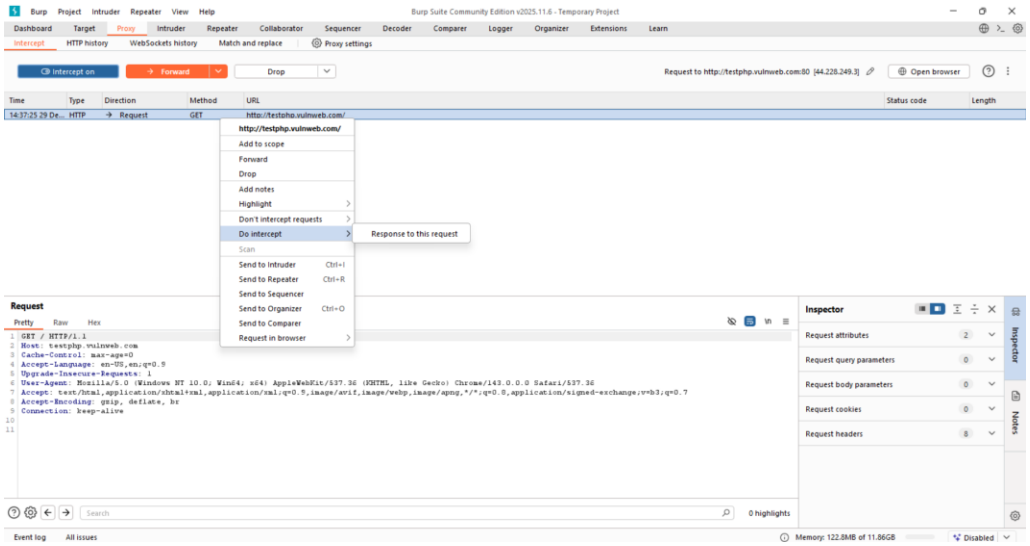
Step4: In the opened chromium browser, visit the website (say: http://testphp.vulnweb.com/)



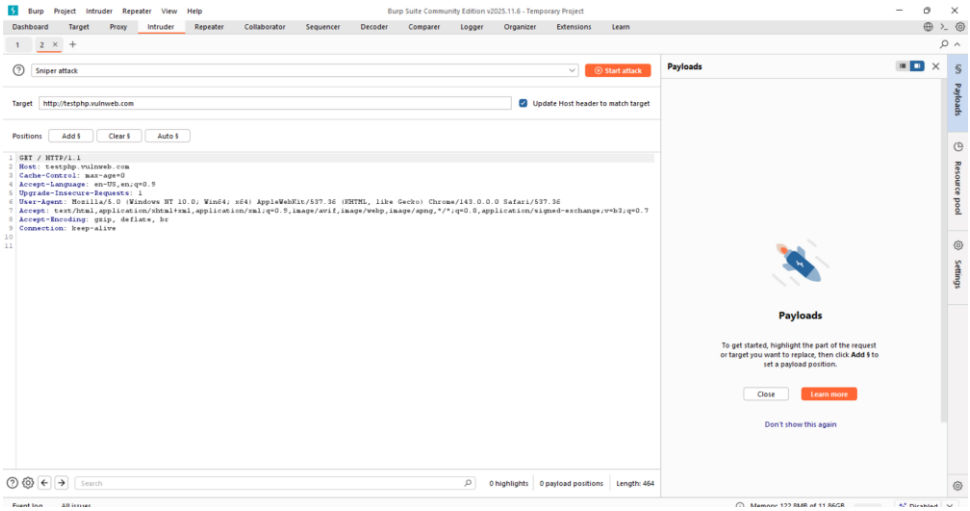
Step5: make sure to keep the intercept on and then refresh the website again. The request will get captured like this.



Step6: Right click on the request and click on “send to intruder”.



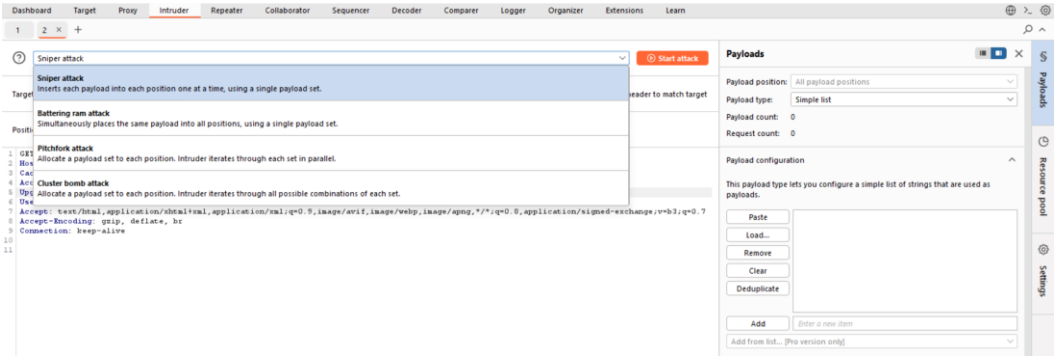
Step7: open the intruder and see that it is here.



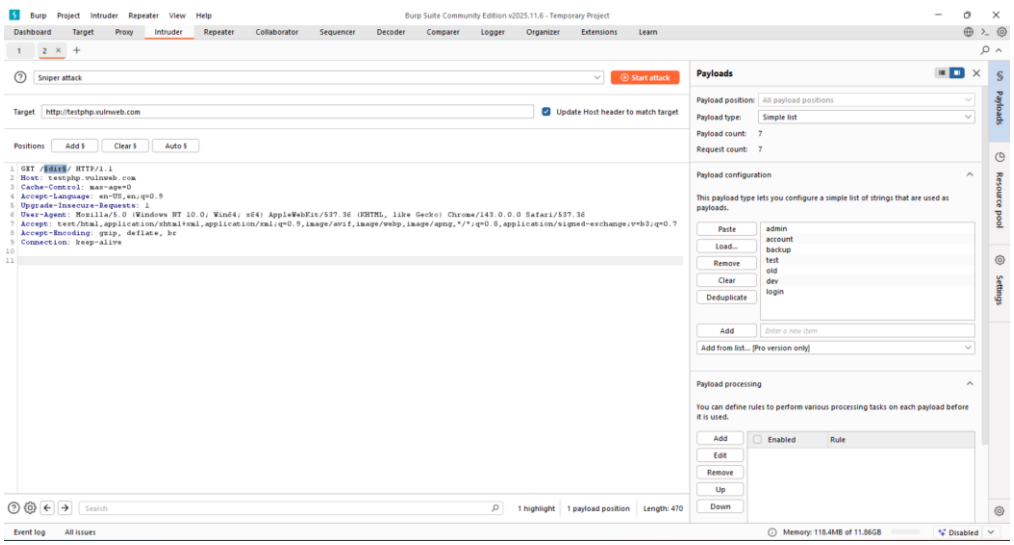
Step8: edit the request like this. This tells burp: Replace §dir§ with values from my wordlist.

```
GET /§dir§/ HTTP/1.1
Host: testphp.vulnweb.com
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

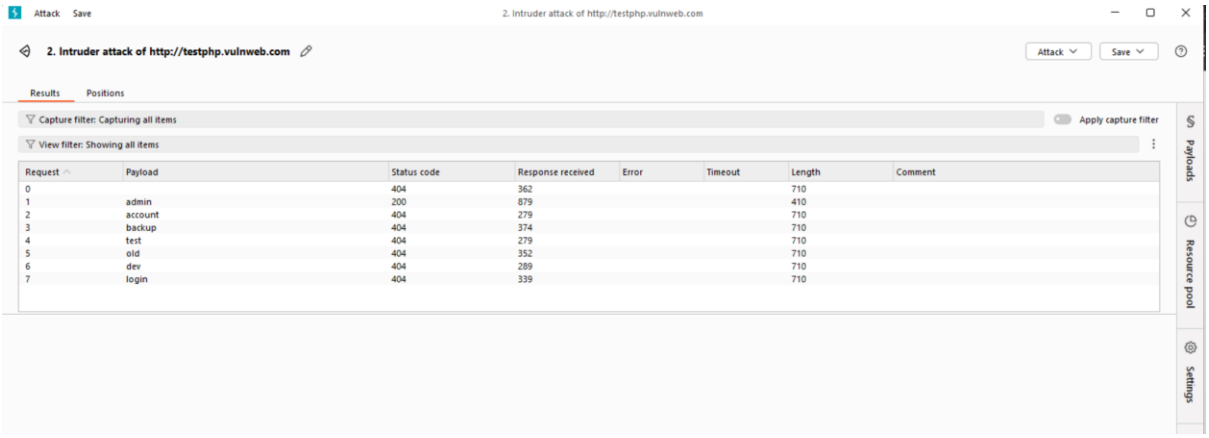
Step9: make sure to select the “sniper” in the attack type.



Step10: Now in the right side, in the payload section, make sure to add the word list. Like this:



Step11: Click on the “start attack” button to start the mapping. Following screen will appear.



Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		404	362			710	
1	admin	200	879			410	
2	account	404	279			710	
3	backup	404	374			710	
4	test	404	279			710	
5	old	404	352			710	
6	dev	404	289			710	
7	login	404	339			710	

Clearly, This is the site’s standard ‘not found’ response (those who have status code ‘404’). Only the one with ‘200’ is valid.

- Burp is not saying: “This is vulnerable”
- Burp is saying: “This response behaves differently from the others.”

Also, we can do recursive discovery as shown below: we just changed the captured request in intruder like this. And change the payload accordingly.

```
GET /admin/§file§ HTTP/1.1
Host: testphp.vulnweb.com
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Inference from published content:

Web applications usually use predictable naming schemes, and by analyzing the names of known resources, you can intelligently guess and automate searches for similar hidden content, increasing the chances of discovering unlisted pages or files.

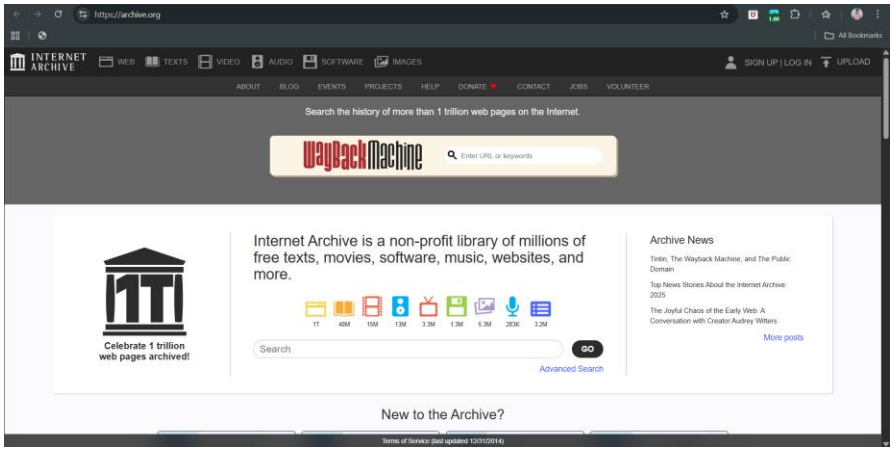
Use of Public Information:

Public sources like search engines and web archives can reveal hidden or unlinked application content by preserving old links, cached pages, and third-party references, helping uncover functionality that is no longer visible on the live site.

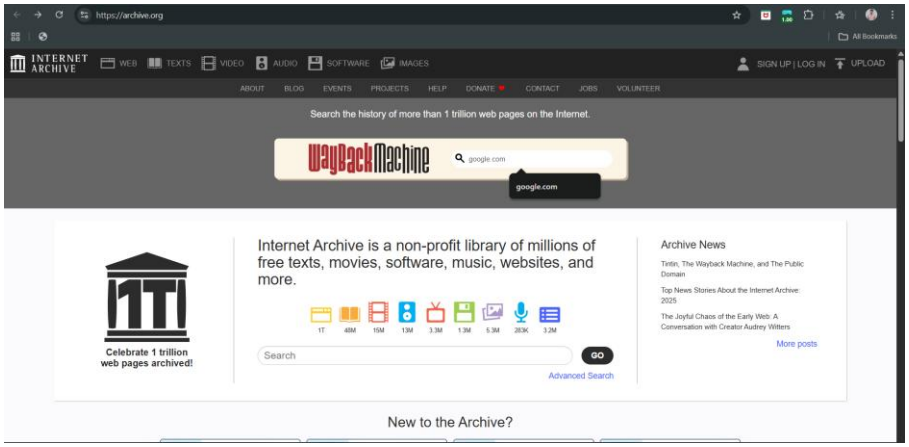
Using way back machines to get an idea of website:

Step1: Open the website https://archive.org/

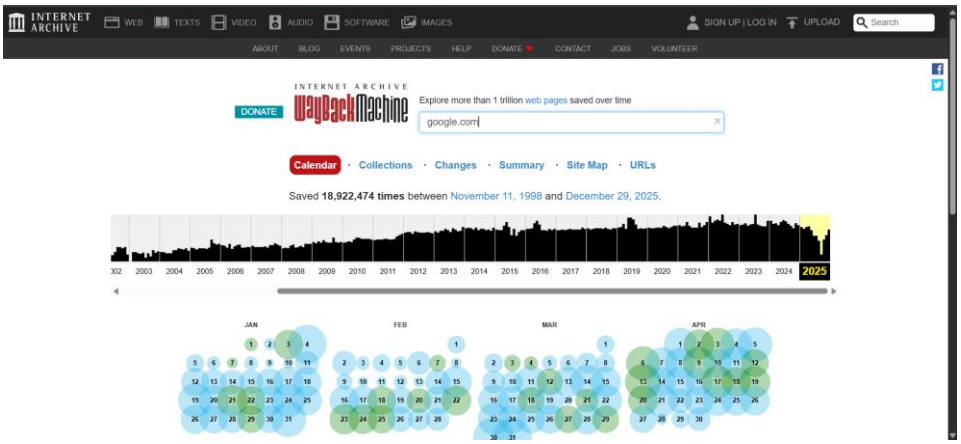
Step2: In the appeared window, fill the details.



After filling details:



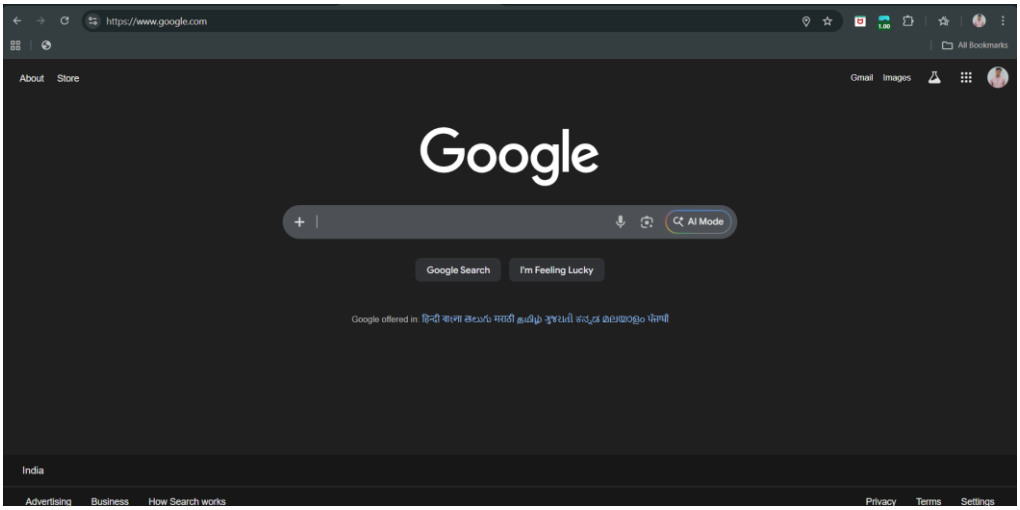
Step3: Following screen will appear, browse to which ever section you wish.



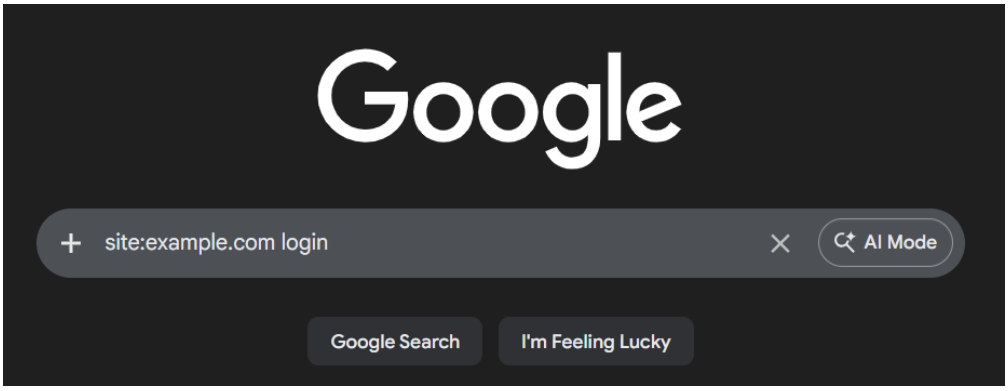
Using google dorks to get an idea of website:

Step1: Visit the website of google search https://www.google.com/

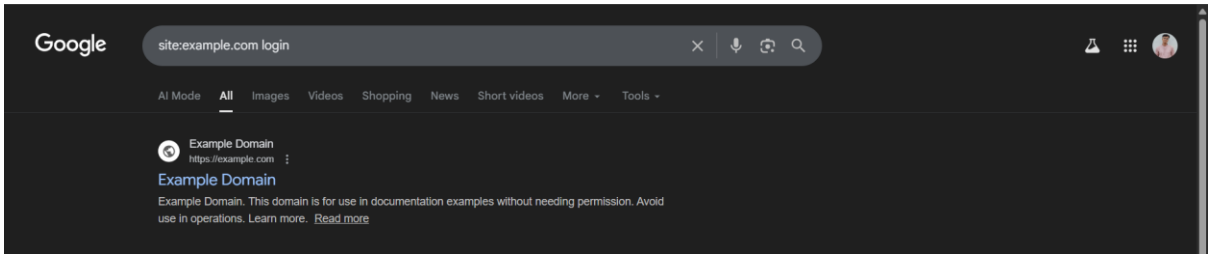
Step2: Following screen will appear.



Step3: In the search bar type the following: *site:example.com login*



Following output will come:



Some useful dorks:

site:example.com
site:example.com login
site:example.com admin
site:example.com "sign in"
site:example.com "log in"
site:example.com "password reset"
site:example.com "forgot password"

site:example.com ext:bak
site:example.com ext:old
site:example.com ext:backup
site:example.com ext:tmp
site:example.com ext:zip
site:example.com ext:rar
site:example.com ext:7z
site:example.com ext:sql
site:example.com ext:db
site:example.com ext:sqlite

site:example.com ext:conf
site:example.com ext:config
site:example.com ext:env
site:example.com ext:ini
site:example.com ext:inc
site:example.com ext:php
site:example.com ext:asp
site:example.com ext:aspx
site:example.com ext:jsp
site:example.com ext:java
site:example.com ext:cs

site:example.com "change password"
site:example.com "db_password"
site:example.com "database"
site:example.com ext:log
site:example.com debug
site:example.com error
site:example.com warning

site:example.com TODO
site:example.com FIXME
site:example.com "under construction"
site:example.com ext:pdf
site:example.com ext:doc
site:example.com ext:docx
site:example.com ext:xls
site:example.com ext:xlsx
site:example.com ext:ppt

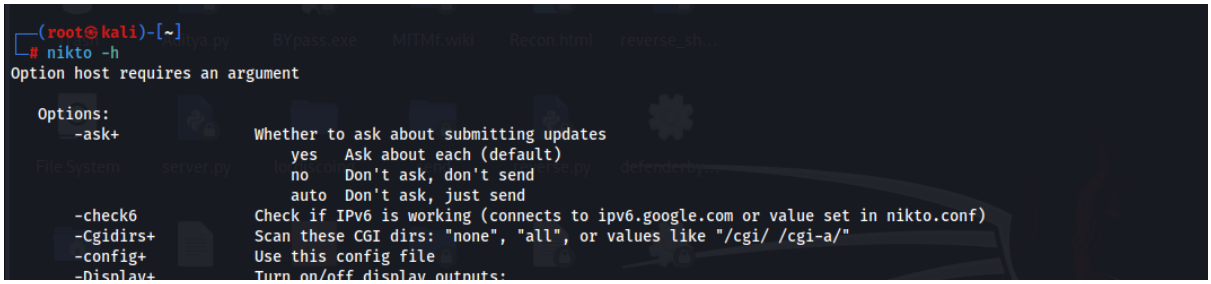
site:example.com api
site:example.com "api/v1"
site:example.com swagger
site:example.com graphql
link:example.com
related:example.com
cache:example.com

Leveraging the Web Server:

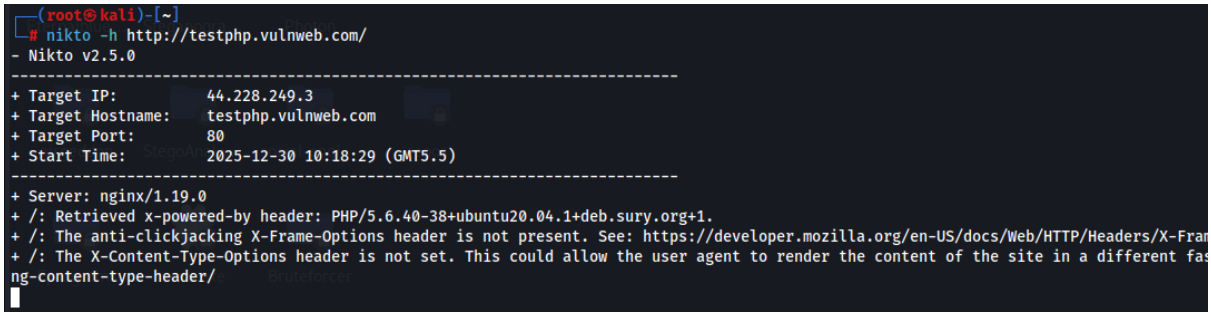
By exploiting web server vulnerabilities and default configurations—such as directory listing, exposed source code, sample scripts, and third-party components—you can discover hidden application content, often using automated tools like Nikto to identify unlinked but accessible resources.

<u>Using Nikto for scanning vulnerabilities:</u> (Part of Leveraging the Web Server)

Step1: Open the help menu of the nikto in the Kali Linux.



Step2: specify the target.
Command: *nikto -h http://testphp.vulnweb.com/*



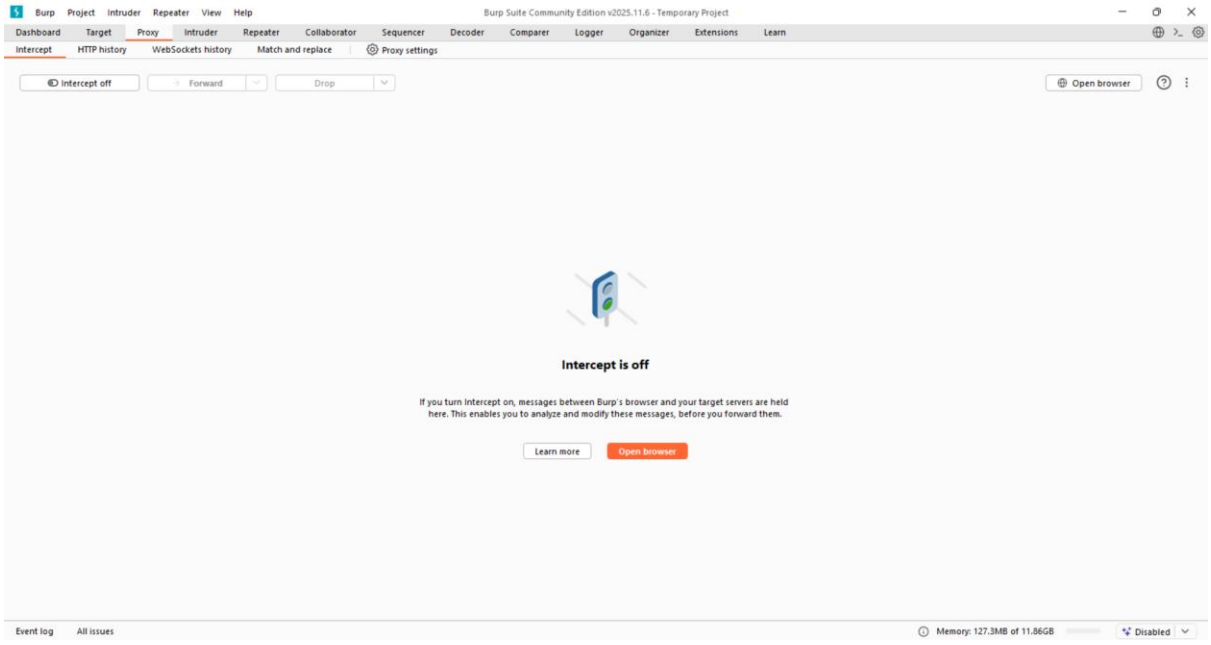
<u>Application Pages vs. Functional Paths:</u>

Earlier websites used separate URLs for each page, making them easy to map. Modern web applications often use one URL for many actions, controlled by request parameters, so <u>URL-based mapping can be misleading</u>. Mapping functional paths instead shows the real flow of actions, relationships between functions, and possible security weaknesses.

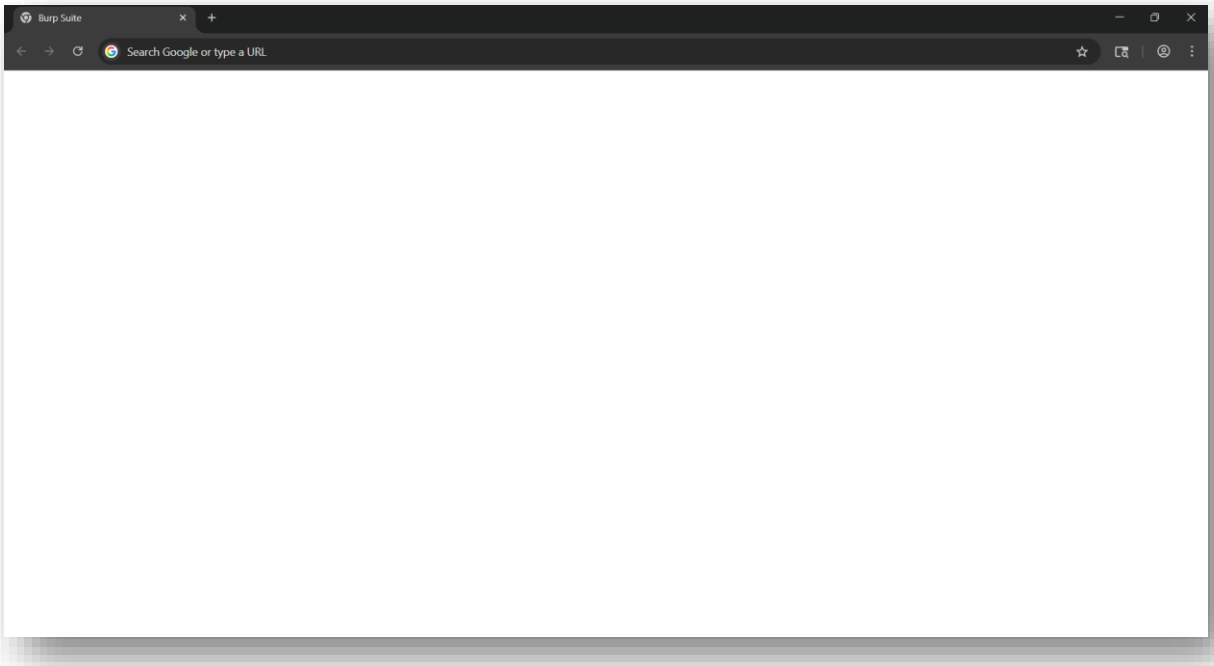
Identify parameter-based functionality using Burp Suite:

Goal: You want to identify functionality controlled by parameters, not by separate pages.
Example: /index.php?page=login
Instead of: /login.php

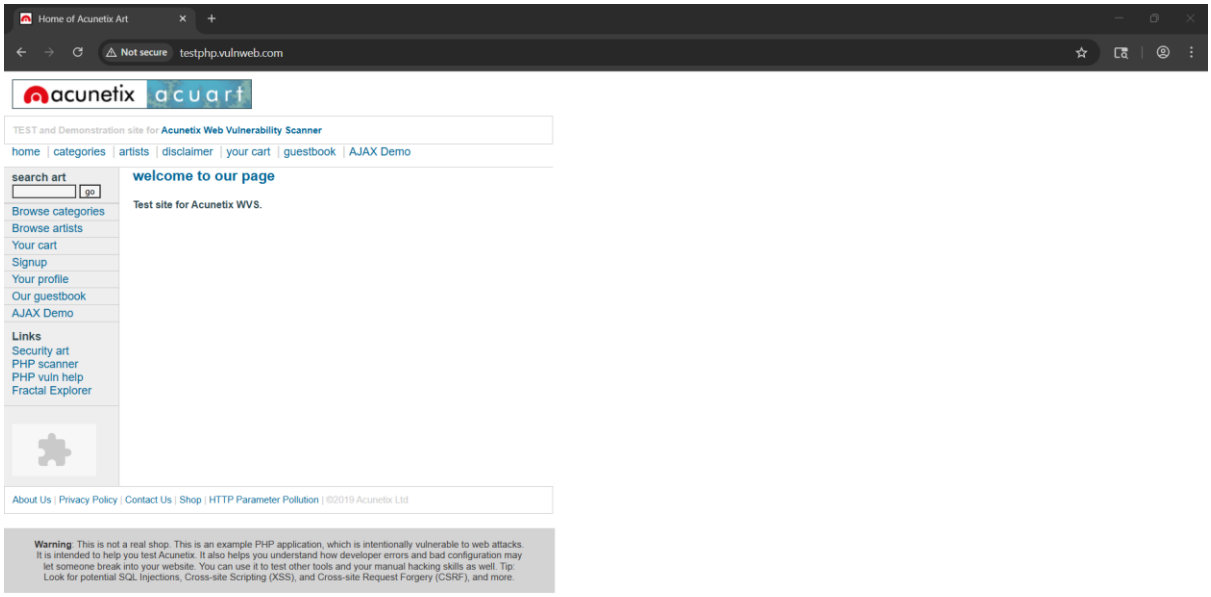
Step1: Open the burp suite, keep the intercept off, and open the browser.
Burpsuite:



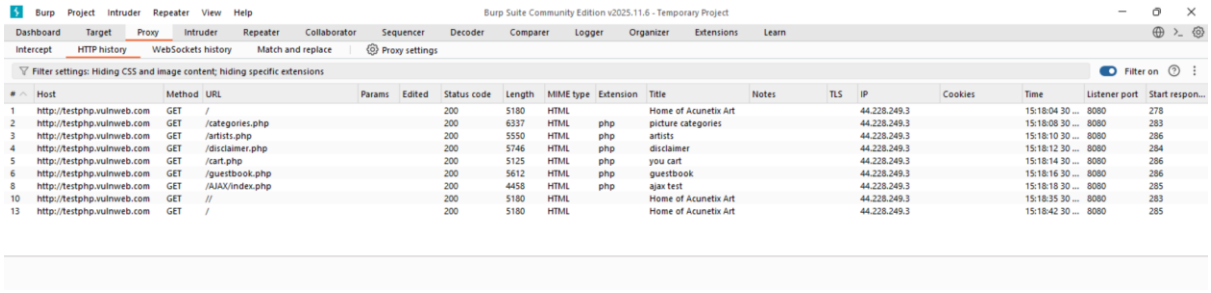
Browser:



Step2: Browse the site normally (say http://testphp.vulnweb.com/). Click around login, categories, etc. Let burp record the traffic. Initially the page will look like this:

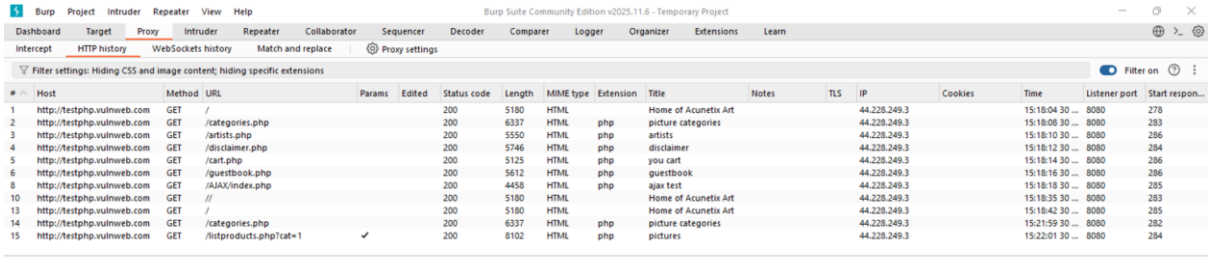


Step3: Check the HTTP history (at Proxy->HTTP History). Following things can be observed.



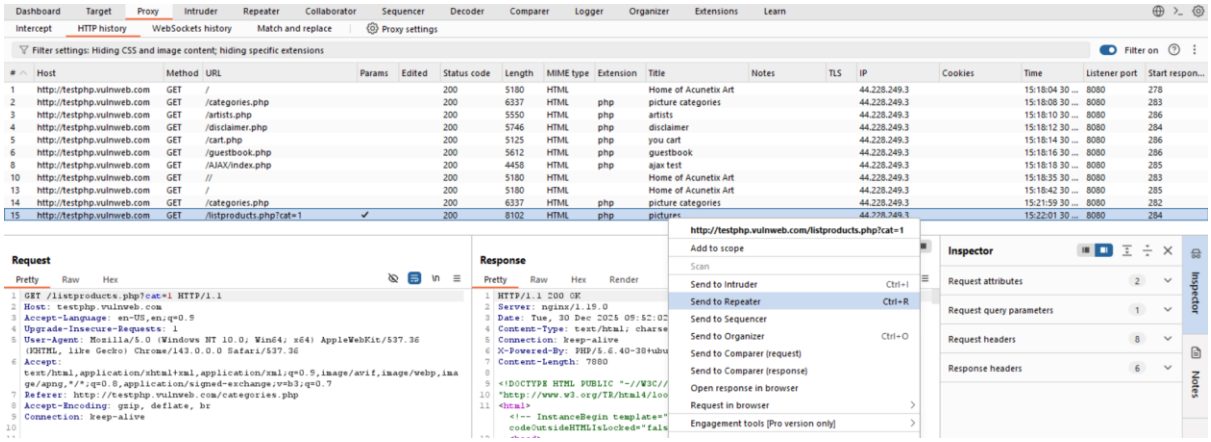
But since, we just roamed the pages we can't see anything useful here.

Step4: So, we again roamed in the page, now visited the http://testphp.vulnweb.com/listproducts.php?cat=1 . And clearly, it has parameters. This is the key finding (/listproducts.php?cat=1)

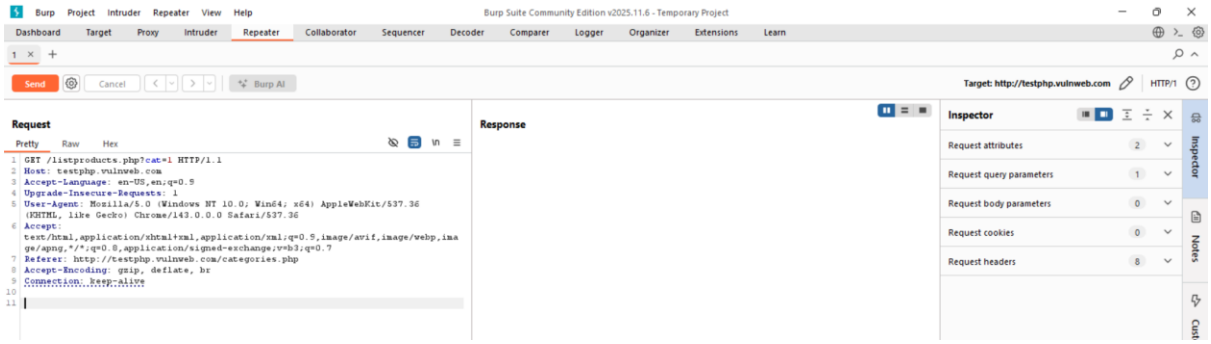


The site uses a single page (listproducts.php) to show different categories, controlled by the cat parameter. This means functionality and data change via URL parameters, not separate pages for each category.

Step5: Confirming the mis use of parameter, by changing cat = 1 -> cat = 2 in the Repeater. So, first send that request to the repeater.

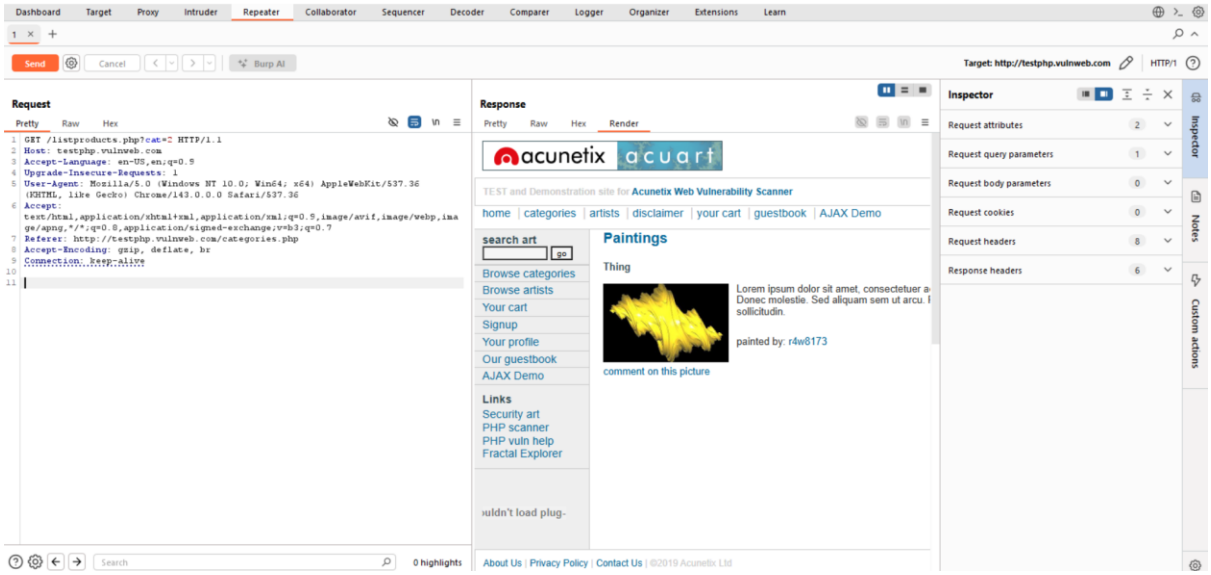


In repeater this request can be seen as:



Step6: Change cat = 1 to cat = 2.

Step7: Click on ‘send’ button. Following screen can be observed. Clearly, it allowed. That proves: The parameter controls application behaviour.



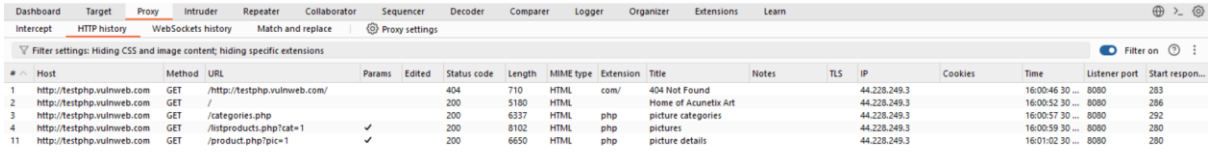
Clearly, Using Burp Suite HTTP history, it was observed that the same page (listproducts.php) is used to display different categories, with the cat parameter determining the functionality. This shows that application behavior is controlled through request parameters rather than distinct URLs.

Discovering Hidden Parameters:

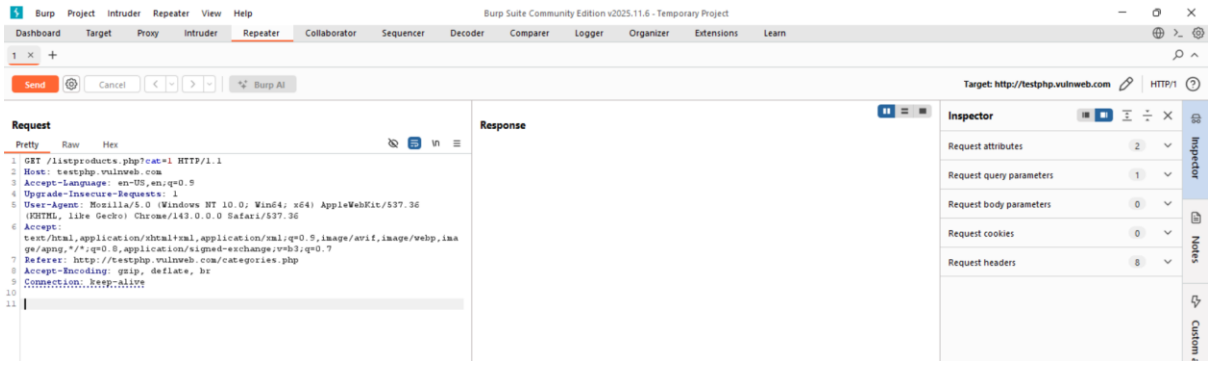
This means: Sending requests with extra parameters and observing whether the application reacts differently.

Discovering hidden parameters using Burp Suite:

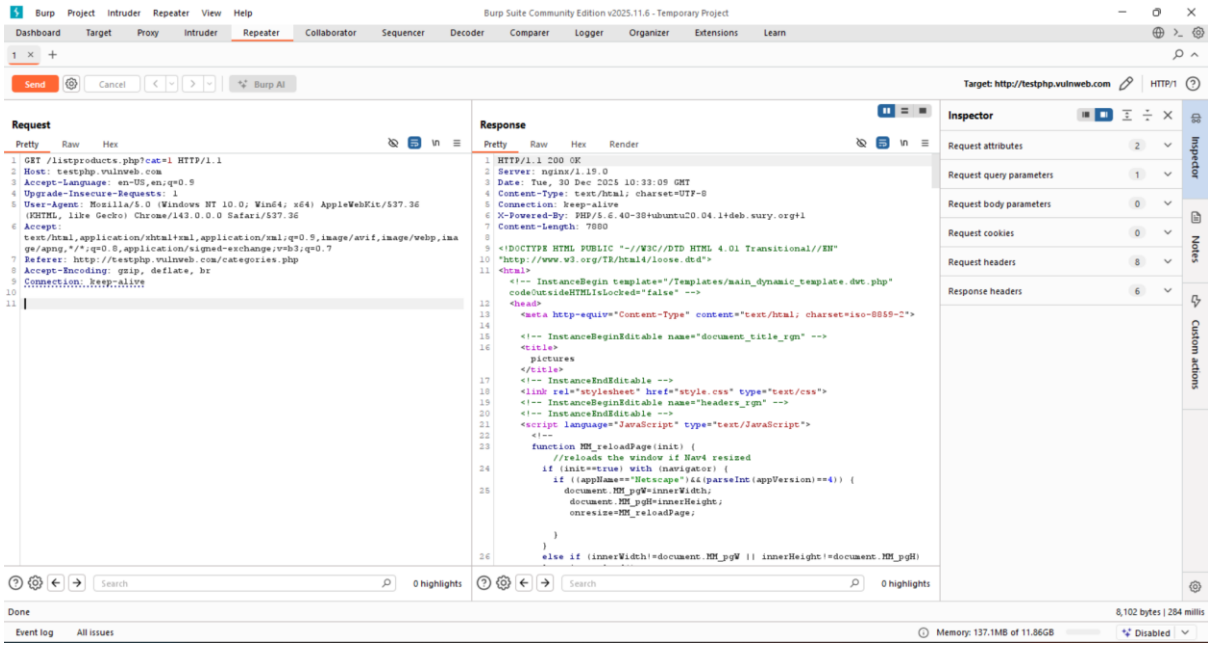
Step1: Open the Burp suite, and just simply look for those where parameters are involved. Clearly, we found such. This is out baseline request.



Step2: Right click and send to “Repeater”. It can be seen like this:

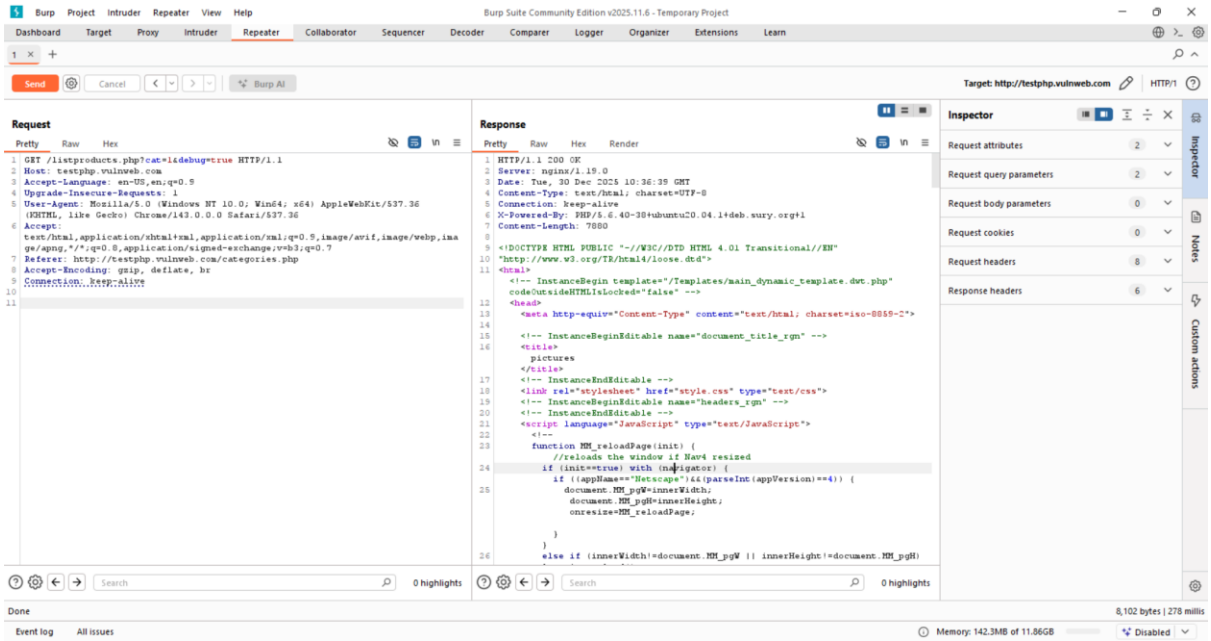


Step3: Observe the normal response using the “send” button: this is the baseline for future.



Step4: Add guessed hidden parameters. We added &debug = true or we can also test with &test=1.

Step5: Click on the ‘send’ button, clearly, the same result can be seen: it means &debug = true doesn’t found.



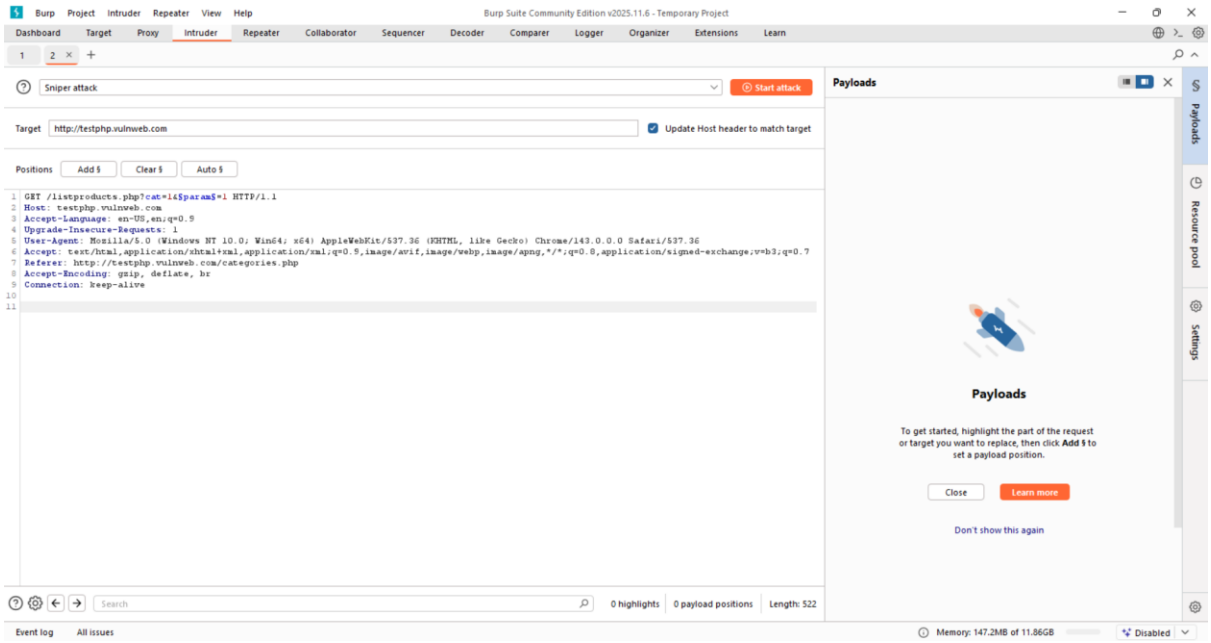
Step6: automating this. We added &$param$ = 1, and then send it to the intruder.

Request

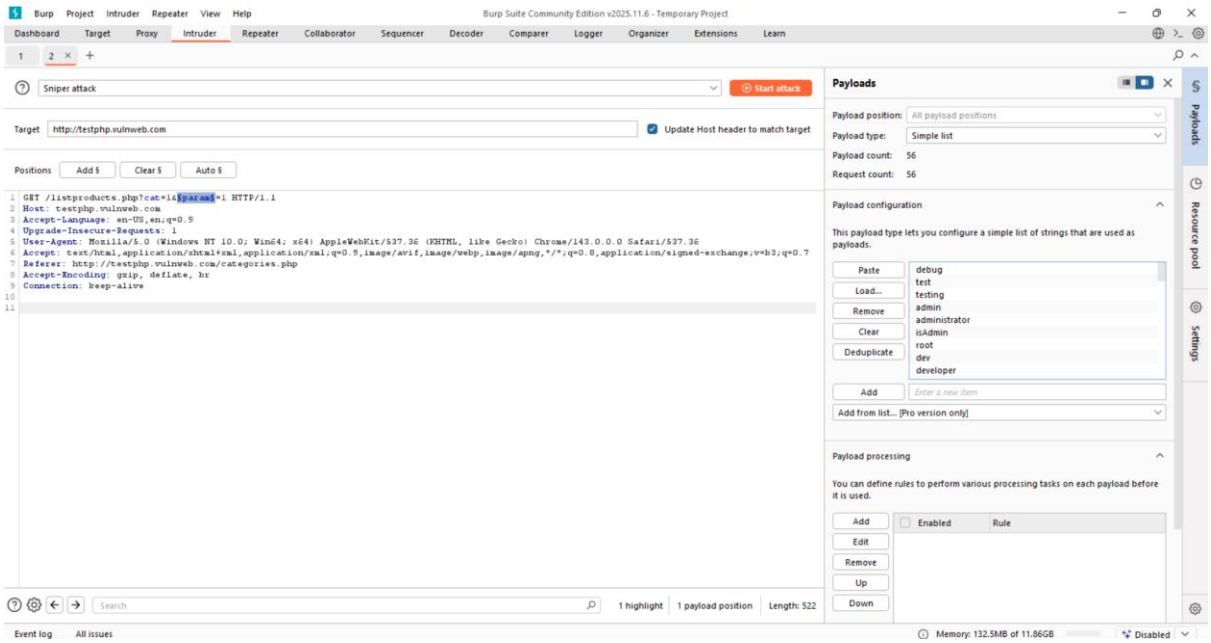
Pretty Raw Hex

```
1  GET /listproducts.php?cat=1&$param$=1 HTTP/1.1
2  Host: testphp.vulnweb.com
3  Accept-Language: en-US,en;q=0.9
4  Upgrade-Insecure-Requests: 1
5  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
6  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
   ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
7  Referer: http://testphp.vulnweb.com/categories.php
8  Accept-Encoding: gzip, deflate, br
9  Connection: keep-alive
10
```

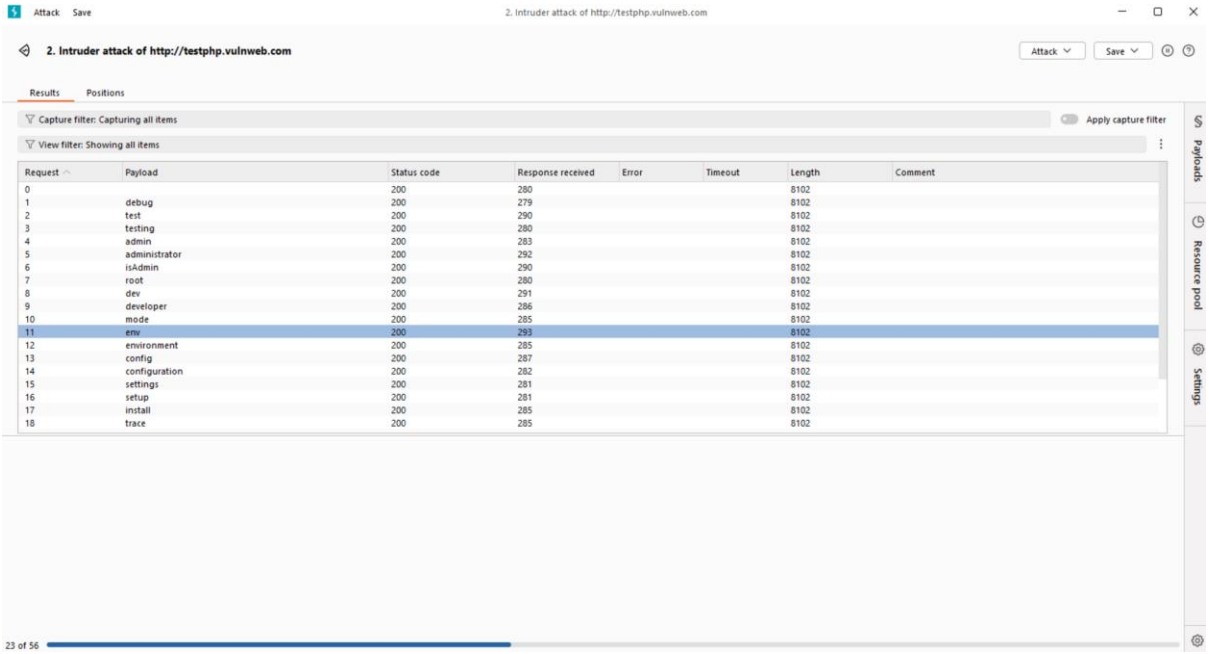
In Intruder it will appear like this:



Step7: Add the payload.



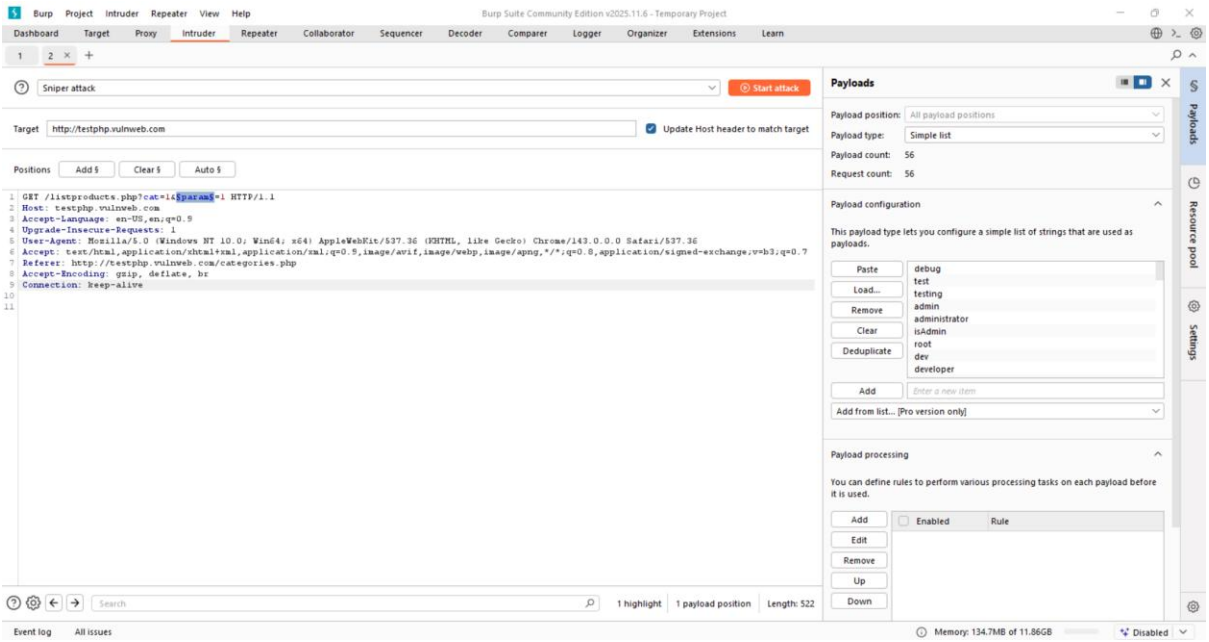
Step8: start the attack, following observation can be made: here 200 doesn't means that it exists.



Since, length of each is same, so we can say that we can't manage to find any hidden parameter for now, it might be present for any other sub domain of it. All tested payloads returned identical response lengths and content, indicating that the application ignores these parameters and does not expose hidden functionality through them.

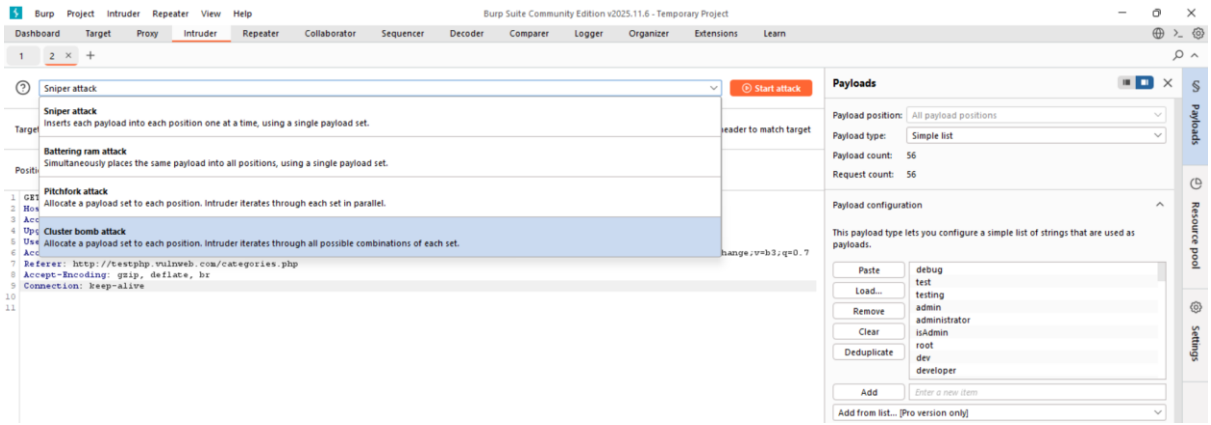
Permutation-based hidden parameter discovery using Burp Suite:

Step1: Get the request in intruder.

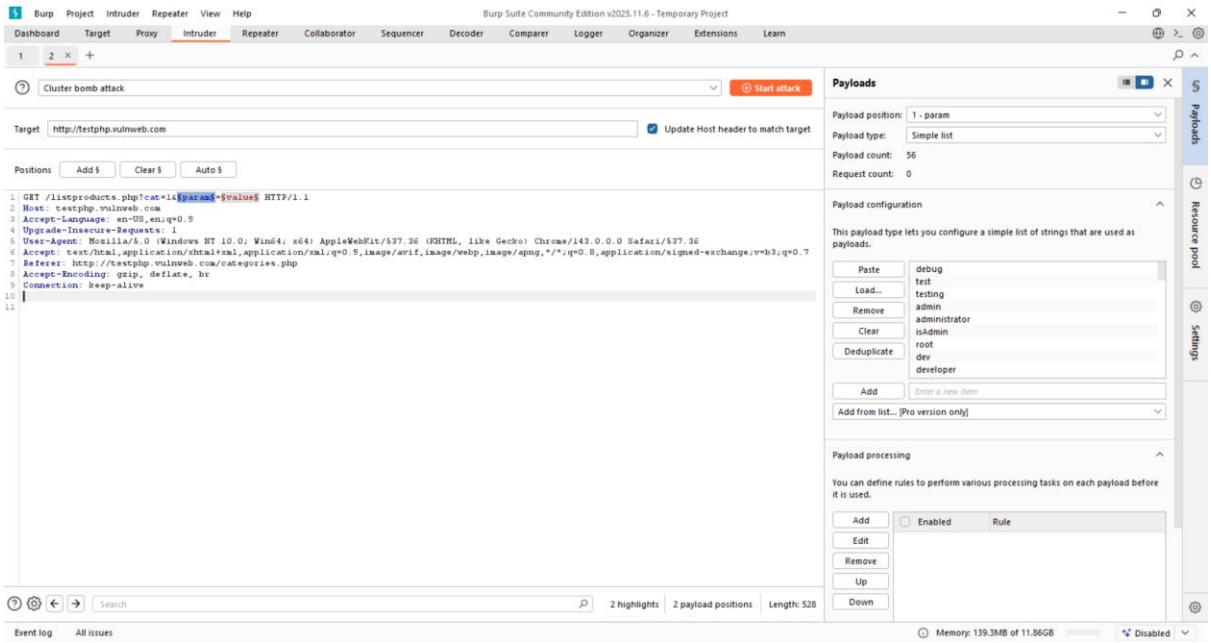


Step2: change the parameter. New is: *GET /listproducts.php?cat=1&§param§=§value§ HTTP/1.1*

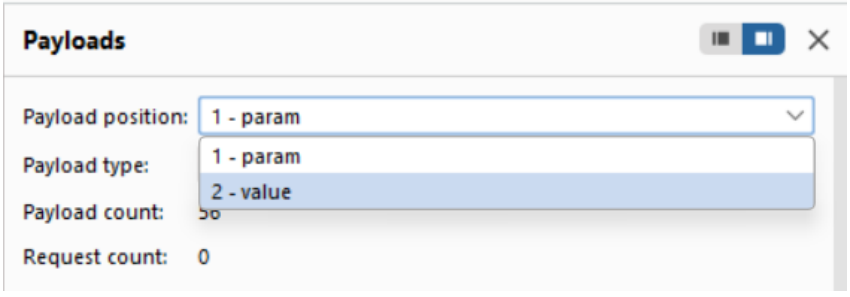
Step3: change the attack type to cluster, as we are using two parameters.



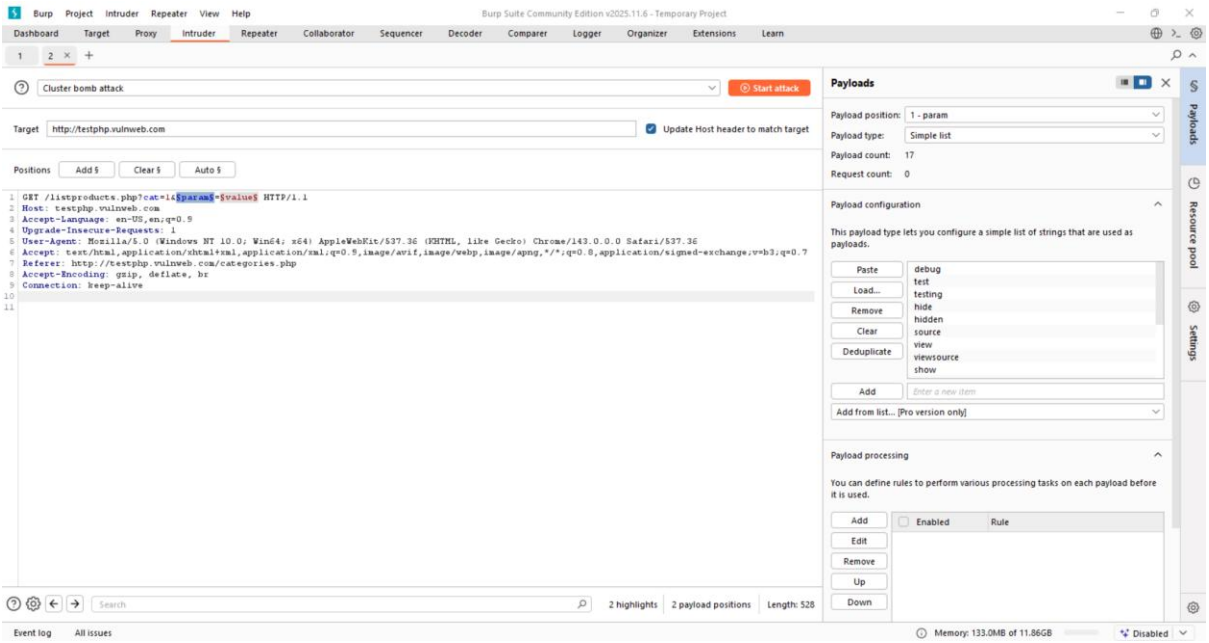
Now, intruder will look like this:



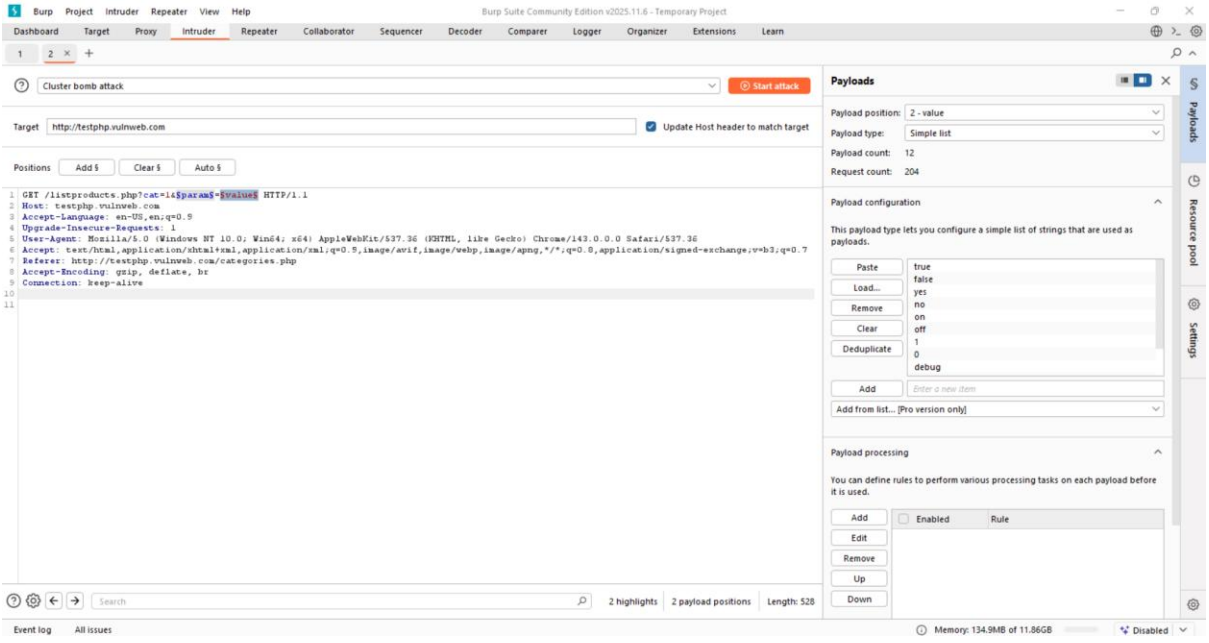
Step4: At the Payloads section, select the '1-param'.



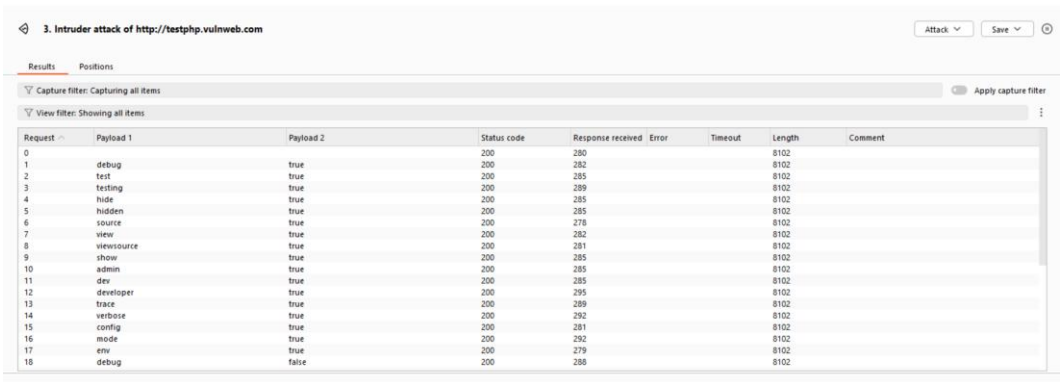
Step5: paste the list for the parameter 1.



Step6: Now change it to the '2-value' and paste the payload.



Step7: Click on "start attack" button. Following observation can be made. All length be same.

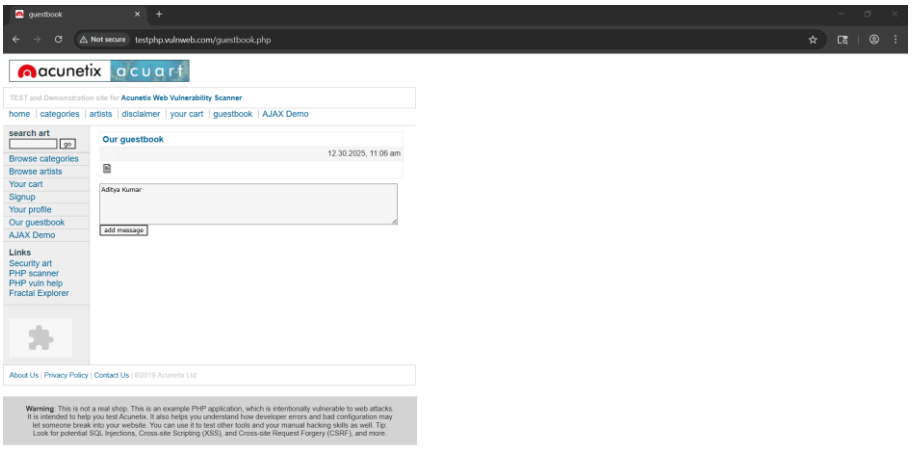


Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
0			200	280			8102	
1	debug	true	200	282			8102	
2	test	true	200	285			8102	
3	testing	true	200	289			8102	
4	hide	true	200	285			8102	
5	hidden	true	200	285			8102	
6	source	true	200	278			8102	
7	view	true	200	282			8102	
8	viewsource	true	200	281			8102	
9	show	true	200	285			8102	
10	admin	true	200	285			8102	
11	dev	true	200	285			8102	
12	developer	true	200	295			8102	
13	trace	true	200	289			8102	
14	verbose	true	200	292			8102	
15	config	true	200	281			8102	
16	mode	true	200	292			8102	
17	env	true	200	279			8102	
18	debug	false	200	288			8102	

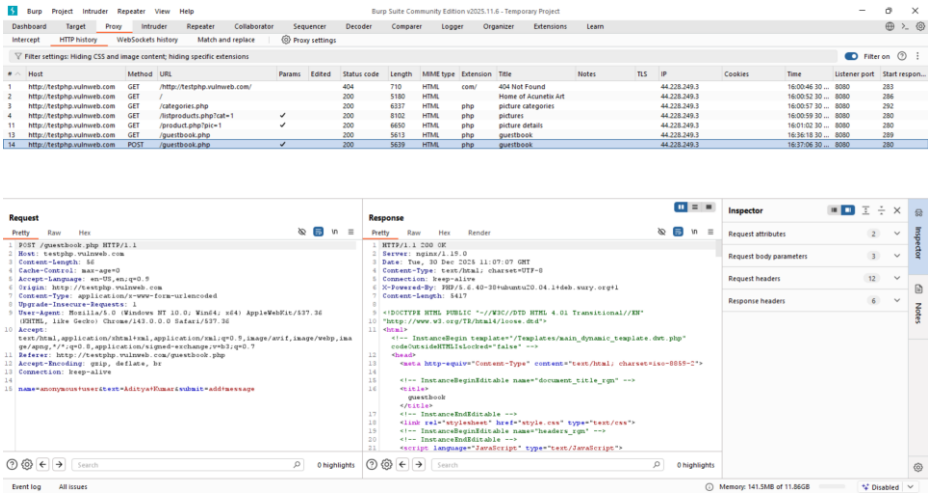
Insert the parameter both into the URL query string and into the message body:

Step1: Capture a POST request (try in http://testphp.vulnweb.com/guestbook.php) and send it to intruder.

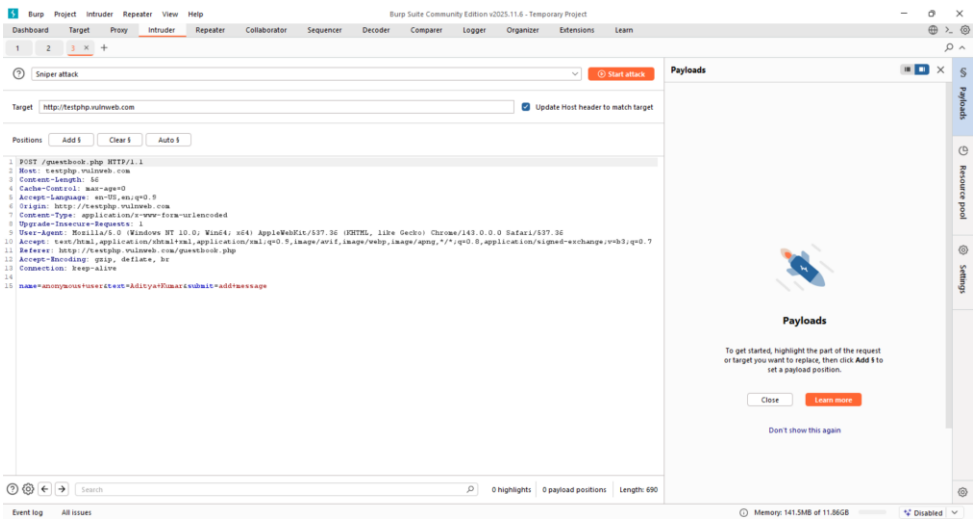
In browser:



In HTTP History: “POST” request can be seen clearly.



In intruder: the POST request can be seen.

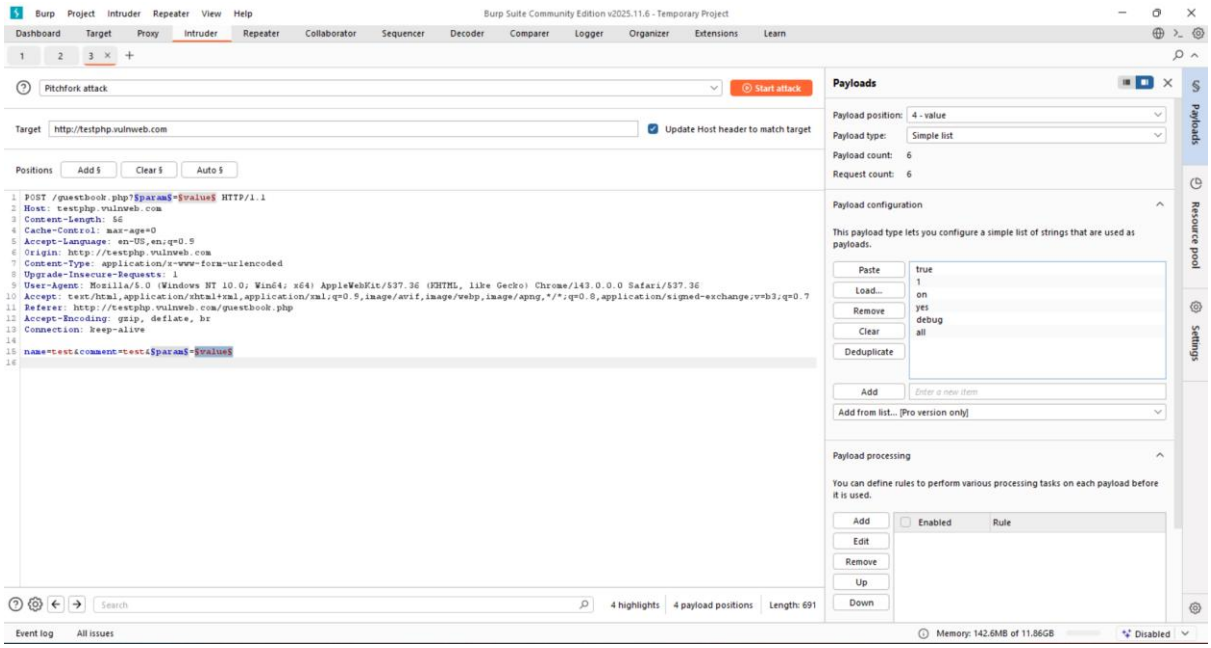


Step2:Modify the POST request and the Body:

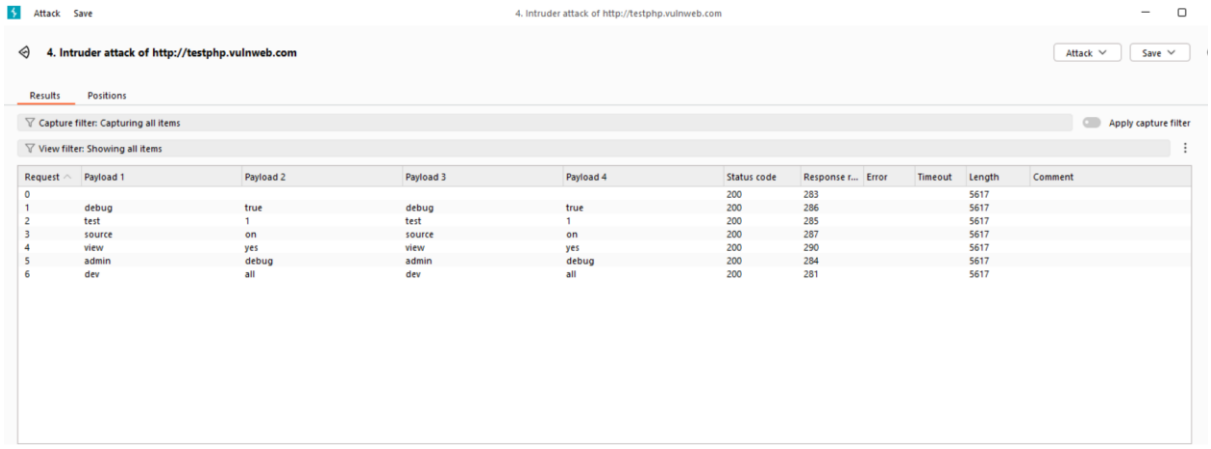
```
POST /guestbook.php?$param$=$value$ HTTP/1.1
Host: testphp.vulnweb.com
Content-Length: 56
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://testphp.vulnweb.com/guestbook.php
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

name=test&comment=test&$param$=$value$
```

Step3: Change the attack type to “Pitchfork”, and add the payloads:



Step4: Start the attack, and observe the length.



Request	Payload 1	Payload 2	Payload 3	Payload 4	Status code	Response r...	Error	Timeout	Length	Comment
0					200	283			5617	
1	debug	true	debug	true	200	286			5617	
2	test	1	test	1	200	285			5617	
3	source	on	source	on	200	287			5617	
4	view	yes	view	yes	200	290			5617	
5	admin	debug	admin	debug	200	284			5617	
6	dev	all	dev	all	200	281			5617	

An Intruder attack was performed against guestbook.php by injecting common debug-related parameter names and values into both the URL query string and POST request body using Pitchfork mode. All requests returned identical response lengths and content, indicating that the application ignores these parameters and does not expose hidden logic through them.

Analyzing the Application:

Understanding both client-side and server-side behavior helps in identifying and testing potential security vulnerabilities effectively.

Identifying Entry Points for User Input:

What are "entry points for user input"? Entry points are places where a user can send data to the application, and the server processes it.

Common places where applications take user input:

- URLs before the "?" – Sometimes data is hidden in the URL path itself.
- Query string parameters – Data after ? in a URL (e.g., ?id=10).
- POST request data – Data sent through forms (like login forms).
- Cookies – Small pieces of data stored in the browser and sent to the server.
- HTTP headers – Some applications read special headers like:
    - User-Agent (browser info)
    - Referer (previous page)
    - Accept, Accept-Language
    - Host

Out-of-band (indirect) input entry points:
Some applications receive user-controlled data outside normal web requests, meaning you won't see it just by watching HTTP traffic.
Examples:

- Webmail apps reading emails sent via SMTP
- Publishing apps fetching content from other websites
- Security tools collecting data from network traffic and showing it in a web interface

These inputs can still be controlled by users and may introduce vulnerabilities.

Entry points are all places where an application accepts user input. Input can come from URLs, forms, cookies, headers, custom formats, or even external sources. Understanding every input path is critical to finding hidden and serious security vulnerabilities.

Identifying Server-Side Technologies:

It is usually possible to identify the server technologies by observing different clues and indicators they leave behind. To identify we can use:

- Banner Grabbing
- HTTP Fingerprinting
- File extensions
- Directory names
- Session Tokens
- Third-party code components

Banner Grabbing:

What is a "banner"?

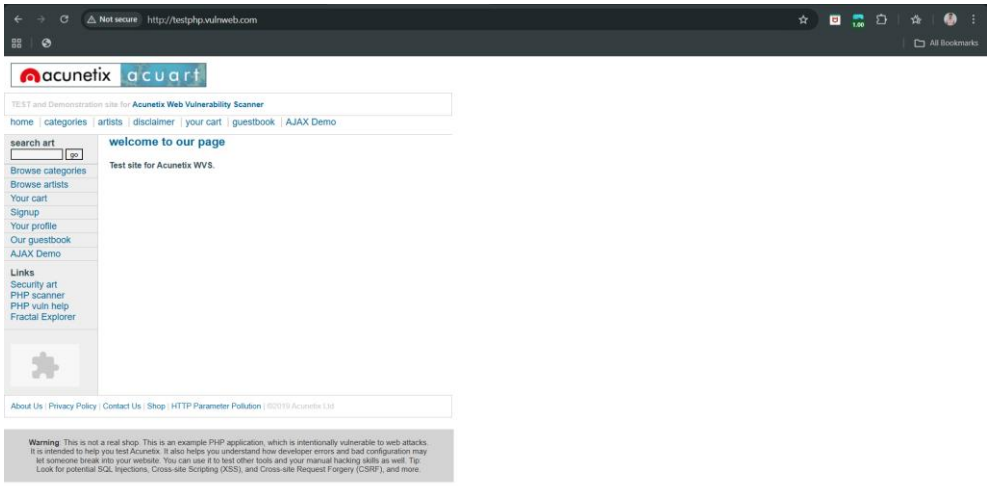
A banner is information sent by a server when it communicates with a browser or client. This information may include:

- The web server software name
- Its version number
- The operating system
- Installed modules or extensions

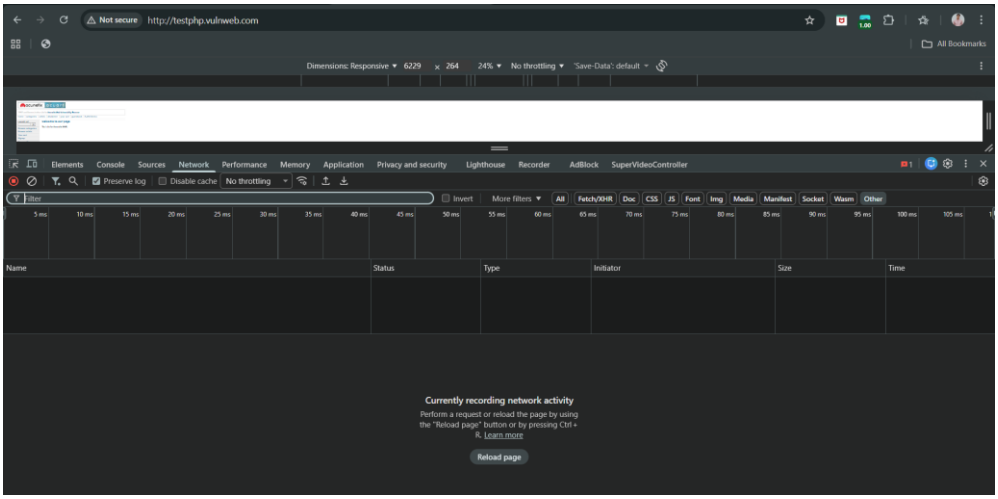
How web servers reveal information? Many web servers send detailed information in the HTTP response headers, especially in the Server header.

Banner Grabbing using Developer tools:

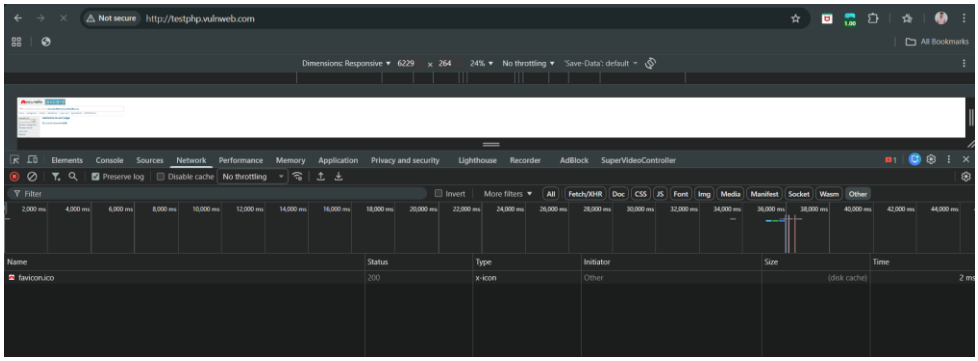
Step1: Open the website. Say http://testphp.vulnweb.com/



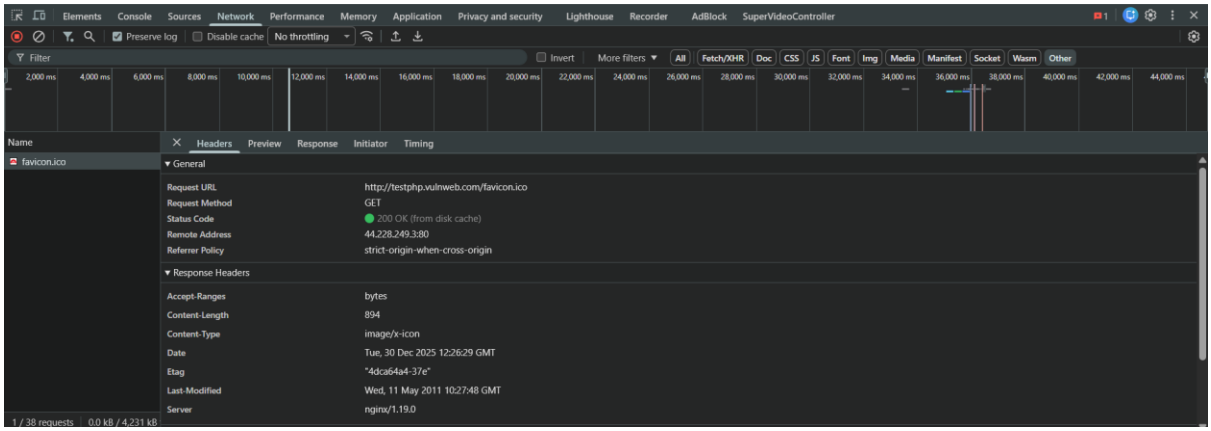
Step2: Open the “network” section: following screen will appear.



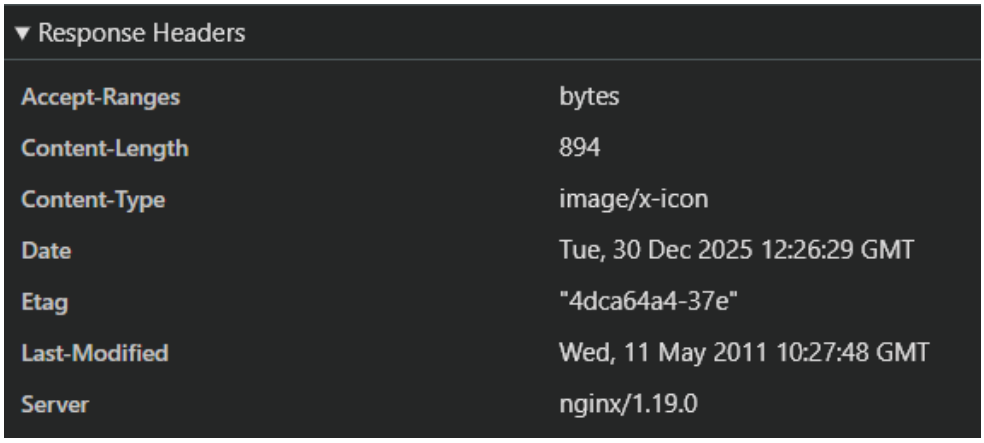
Step3: Reload the page, the first row under the “Name” be the This request represents the main response from the server.



Step4: Click on it, following screen will appear.



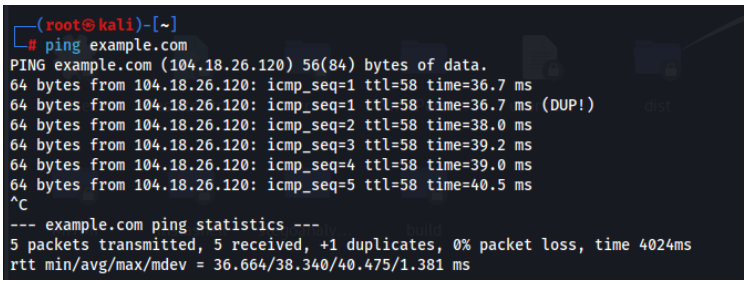
Step5: Check the response header (because it is sent by the server).



The server is using Nginx version 1.19.0.

<u>Banner Grabbing using curl:</u>

Step1: Check if the website is reachable. Yes, it reachable.



Step2: Use the curl command to know about the server. The curl -I command retrieves only HTTP headers for banner grabbing.

```
┌──(root㉿kali)-[~]
└─# curl -I https://example.com

HTTP/2 200
date: Tue, 30 Dec 2025 16:35:30 GMT
content-type: text/html
cf-ray: 9b62f3611b453cb2-BOM
last-modified: Sat, 20 Dec 2025 05:43:40 GMT
allow: GET, HEAD
accept-ranges: bytes
age: 8851
cf-cache-status: HIT
server: cloudflare
```

Clearly, the server is cloudflare. Cloudflare hides the origin server.

<u>Banner Grabbing Using telnet or netcat (Manual Method):</u>

Step1: check if the website is reachable.

```
┌──(root㉿kali)-[~]
└─# ping example.com
PING example.com (104.18.26.120) 56(84) bytes of data.
64 bytes from 104.18.26.120: icmp_seq=1 ttl=58 time=36.7 ms
64 bytes from 104.18.26.120: icmp_seq=1 ttl=58 time=36.7 ms (DUP!)
64 bytes from 104.18.26.120: icmp_seq=2 ttl=58 time=38.0 ms
64 bytes from 104.18.26.120: icmp_seq=3 ttl=58 time=39.2 ms
64 bytes from 104.18.26.120: icmp_seq=4 ttl=58 time=39.0 ms
64 bytes from 104.18.26.120: icmp_seq=5 ttl=58 time=40.5 ms
^C
--- example.com ping statistics ---
5 packets transmitted, 5 received, +1 duplicates, 0% packet loss, time 4024ms
rtt min/avg/max/mdev = 36.664/38.340/40.475/1.381 ms
```

Step2: *telnet example.com 80*

```
┌──(root㉿kali)-[~]
└─# telnet example.com 80

Trying 104.18.26.120...
Connected to example.com.
Escape character is '^]'.
HEAD / HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Date: Tue, 30 Dec 2025 16:41:57 GMT
Content-Type: text/html
Connection: keep-alive
CF-RAY: 9b62fcd68e1a3bf2-BOM
Last-Modified: Sat, 20 Dec 2025 05:43:40 GMT
Allow: GET, HEAD
Accept-Ranges: bytes
Age: 13492
cf-cache-status: HIT
Server: cloudflare
```

Banner Grabbing Using Security Tools:

Way1: Using the “whatweb”.

```
┌──(root㉿kali)-[~]
└─# whatweb example.com
http://example.com [200 OK] Allow[GET, HEAD], Country[UNITED STATES][US], HTML5, HTTPServer[cloudflare], IP[104.18.26.120], Title[Example Domain], UncommonHeaders[cf-cache-status,cf-ray]
https://example.com [200 OK] Allow[GET, HEAD], Country[UNITED STATES][US], HTML5, HTTPServer[cloudflare], IP[104.18.26.120], Title[Example Domain], UncommonHeaders[cf-cache-status,cf-ray]
```

Way2: Using “Nmap”

nmap -sV <ip_address> or *nmap --script http-headers example.com -p 80,443*

```
┌──(root㉿kali)-[~]
└─# nmap --script http-headers example.com -p 80,443

Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-30 22:16 IST
Nmap scan report for example.com (104.18.27.120)
Host is up (0.037s latency).
Other addresses for example.com (not scanned): 104.18.26.120 2606:4700::6812:1a78 2606:4700::6812:1b78

PORT    STATE SERVICE
80/tcp  open  http
| http-headers:
|   Date: Tue, 30 Dec 2025 16:46:54 GMT
|   Content-Type: text/html
|   Connection: close
|   CF-RAY: 9b6304164c3748ef-BOM
|   Last-Modified: Sat, 20 Dec 2025 05:43:40 GMT
|   Allow: GET, HEAD
|   Accept-Ranges: bytes
|   Age: 13789
|   cf-cache-status: HIT
|   Server: cloudflare
|
|_  (Request type: HEAD)
443/tcp open  https
| http-headers:
|   Date: Tue, 30 Dec 2025 16:46:54 GMT
|   Content-Type: text/html
|   Connection: close
|   CF-RAY: 9b6304165c71d91f-BOM
|   last-modified: Sat, 20 Dec 2025 05:43:40 GMT
|   allow: GET, HEAD
|   Accept-Ranges: bytes
|   Age: 9536
|   cf-cache-status: HIT
|   Server: cloudflare
|
|_  (Request type: HEAD)

Nmap done: 1 IP address (1 host up) scanned in 0.54 seconds
```

HTTP Fingerprinting:

Why banner grabbing is not always reliable?

Server software allows admins to change or fake the Server header. Security tools can hide or mask the real server identity. So the information you see may be:

- Incomplete
- Modified
- Completely false

Why attackers can still identify the server?

Even if banners are hidden or faked, web servers behave differently.

Reasons:

- HTTP standards leave many behaviours optional
- Different servers implement HTTP in slightly different ways
- Some servers:
    - Handle errors differently
    - Respond differently to invalid requests

- Format headers differently

These small differences act like a fingerprint.

What is HTTP fingerprinting?

HTTP fingerprinting means: Identifying a web server based on how it behaves, not what it claims. Observing subtle differences in:

- Header order
- Error messages
- Response formatting
- Protocol handling

HTTP Fingerprinting using Nmap:

Command:

nmap -sV --script http-headers,http-server-header,http-methods -p 80,443 testphp.vulnweb.com

```
┌──(root㉿kali)-[/usr/share/httprint]
└─# nmap -sV --script http-headers,http-server-header,http-methods -p 80,443 testphp.vulnweb.com

Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-30 22:41 IST
```

Output:

```
┌──(root㉿kali)-[/usr/share/httprint]
└─# nmap -sV --script http-headers,http-server-header,http-methods -p 80,443 testphp.vulnweb.com
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-30 22:41 IST
Nmap scan report for testphp.vulnweb.com (44.228.249.3)
Host is up (0.29s latency).
rDNS record for 44.228.249.3: ec2-44-228-249-3.us-west-2.compute.amazonaws.com

PORT    STATE    SERVICE VERSION
80/tcp  open     http    nginx 1.19.0
| http-headers:
|   Server: nginx/1.19.0
|   Date: Tue, 30 Dec 2025 17:11:51 GMT
|   Content-Type: text/html; charset=UTF-8
|   Connection: close
|   X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
|
|_  (Request type: HEAD)
| http-methods:
|_  Supported Methods: GET HEAD POST
443/tcp filtered https

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 25.89 seconds
```

File Extensions:

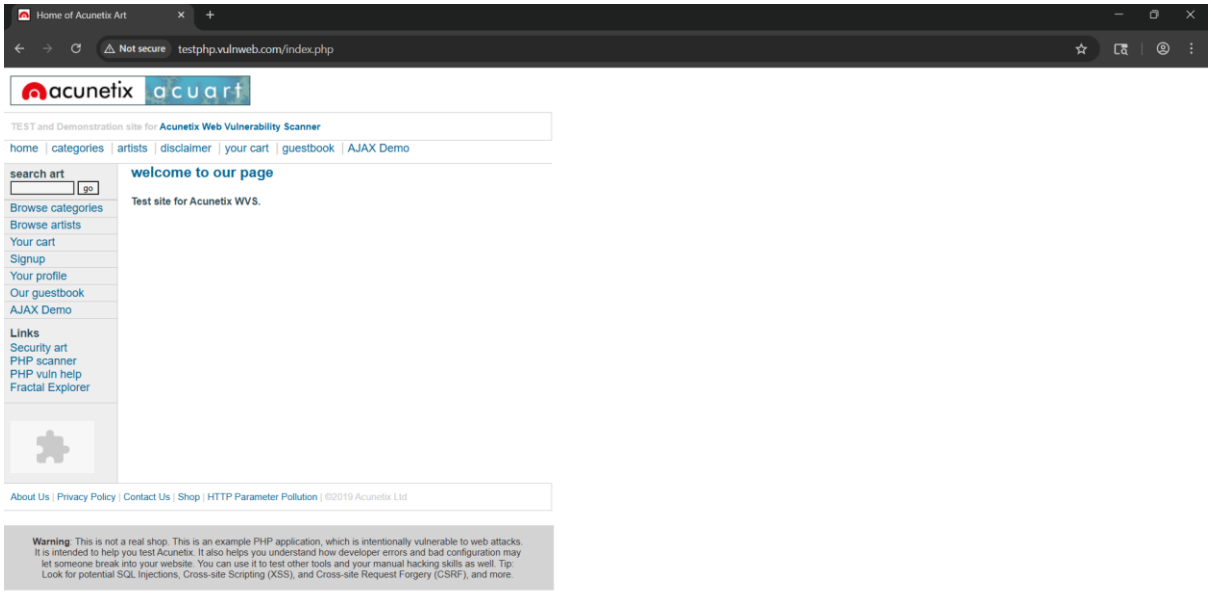
File extensions and server behavior can reveal the technology used by a web application. By passively observing traffic in Burp Suite while browsing testphp.vulnweb.com, we can see URLs ending in .php, PHP session cookies like PHPSESSID, and PHP-style error messages, all of which clearly indicate that the application is built using PHP. Even if file extensions were hidden, differences in error handling and response headers would still help identify the underlying platform, because web servers map different file extensions to different server-side components that behave differently.

Using Burp Suite to find the file extensions:

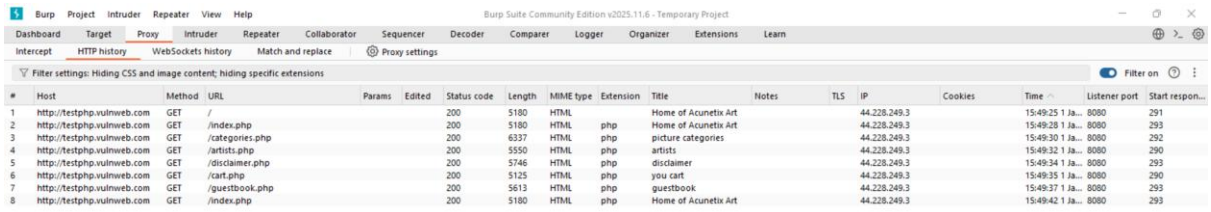
Step1: Set up Burp Suite:

- Open Burp Suite
- Go to Proxy → Intercept
- Make sure Intercept is OFF (so browsing feels normal)
- Open your browser through Burp (Burp's built-in browser is easiest)

Step2: Paste the URL to check on Say : http://testphp.vulnweb.com/ . Following screen will appear.



Step3: Roam around the website, and check Proxy → HTTP history. Following history can be seen to be recorded.

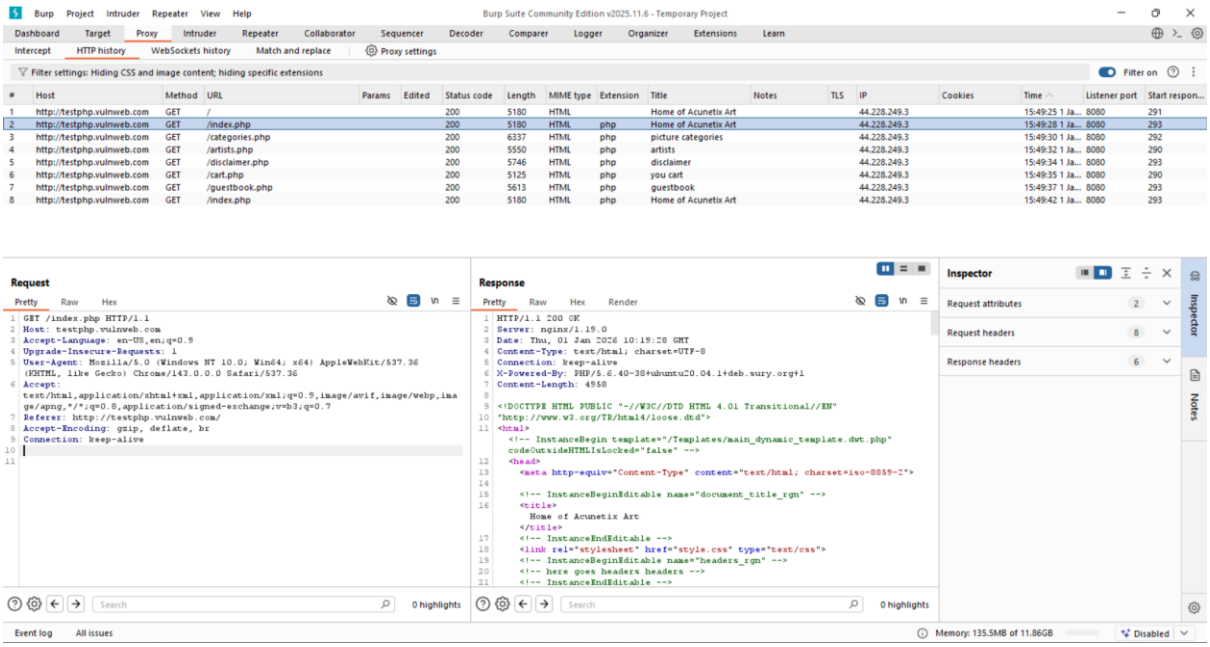


What we saw

- The .php file extension is clearly visible
- This tells us the application is written in PHP

Thus, File extensions in URLs disclose the programming language used.

Step4: Inspect request headers (technology clues). Click any request → Request tab. See the request and response section.



This HTTP exchange shows that the browser requested /index.php, which already indicates the application uses PHP. The response headers clearly confirm this: the server is nginx 1.19.0, and the header X-Powered-By: PHP/5.6.40 explicitly states that PHP is the server-side language. The 200 OK status means the request was handled successfully, and the returned HTML content is generated dynamically by PHP. Together, the file extension (.php), the response headers, and the PHP-generated page content demonstrate how observing requests and responses (for example in Burp Suite) allows us to identify the underlying web technology without needing direct access to the server.

Directory Names:

Even when file extensions are hidden, technology can often be identified by recognizing standard directory names that are commonly created by specific web application platforms.

Note: Burp cannot do OR searches, so search one term at a time. For the case we are using any filter in the Burp.

Session Tokens:

Backend technologies can be identified by analysing session token names set in HTTP cookies. Using Burp Suite's HTTP history, the tester can inspect Set-Cookie headers and identify standard session cookie names such as PHPSESSID, JSESSIONID, or ASP.NET_SessionId, which are commonly generated by specific platforms. This technique allows reliable technology fingerprinting even when file extensions and directory names are hidden.

Third-Party Code Components:

Many web applications use third-party components for common features. In Burp Suite, these can be spotted through URLs, file names, scripts, or comments. Since the same components appear on other sites, checking them elsewhere or their documentation can reveal extra functionality and known security issues.

Identifying out-of-bound input channels:

Out-of-bound data:

Any data that enters an application through channels other than the intended, documented input mechanisms (e.g., form fields or API parameters), but is still user-controllable or third-party controlled. That means external data is flowing into the application outside expected boundaries.

Why Does This Matter?

Because developers often assume these channels are “safe” or “internal”.

When data enters through out-of-bound channels:

- Validation is often missing
- Sanitization is inconsistent
- Security controls may not apply
- Logging and monitoring may mishandle it

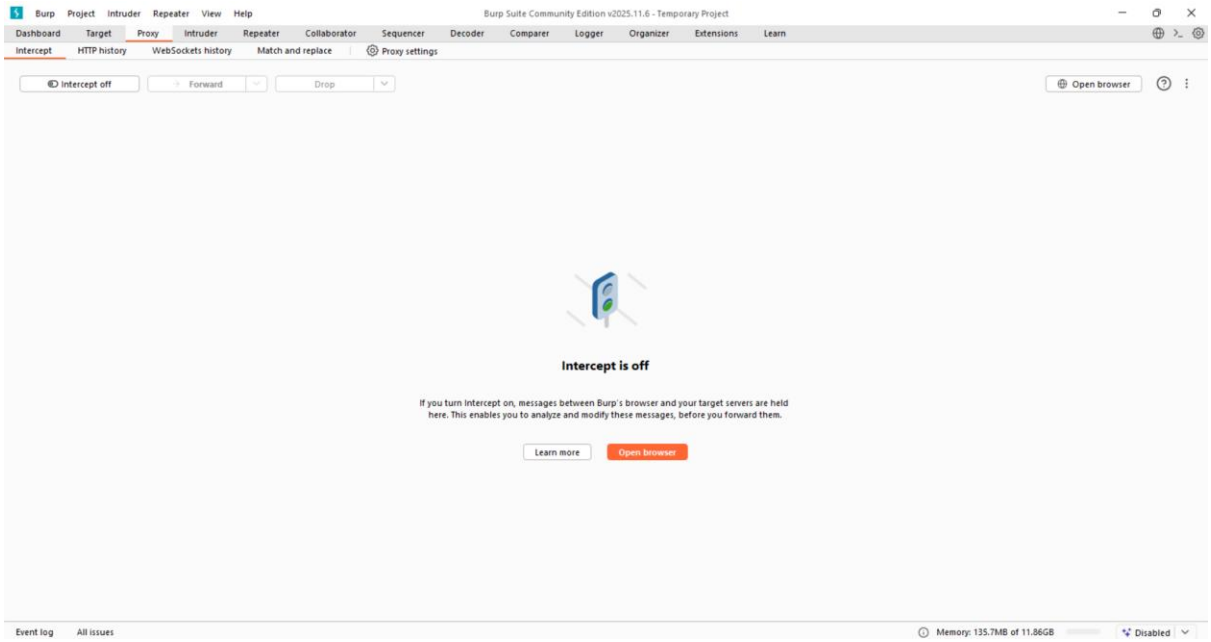
This creates hidden attack surfaces.

Is the Presence of Out-of-Bound Data a Vulnerability?

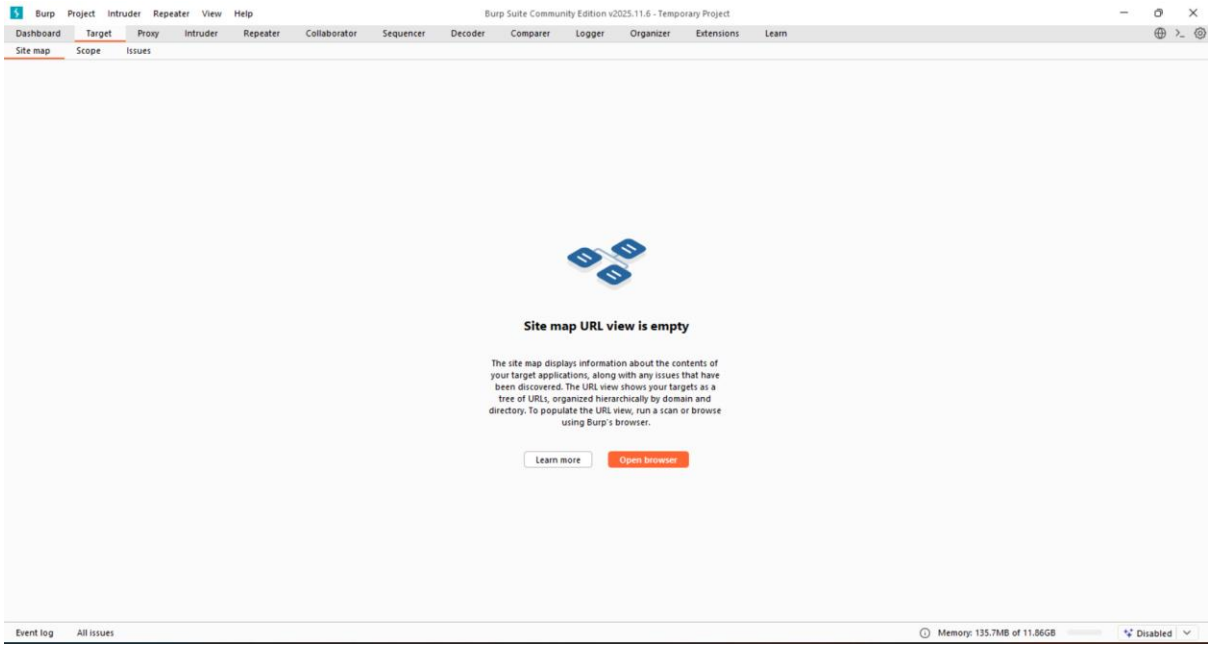
The mere presence of out-of-bound data is not a vulnerability. It is a SECURITY RISK and PRECONDITION. It becomes dangerous when combined with unsafe use.

Identifying out-of-bound input channels burp suite:

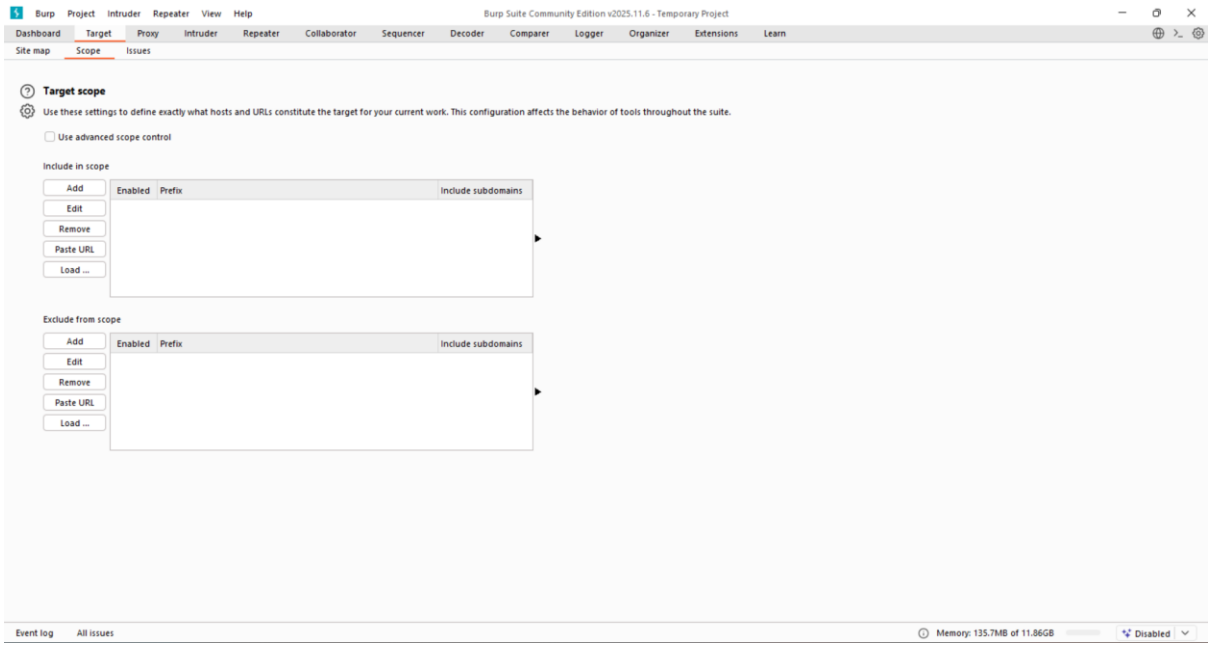
Step1: open burp suite, keep intercept off for now.



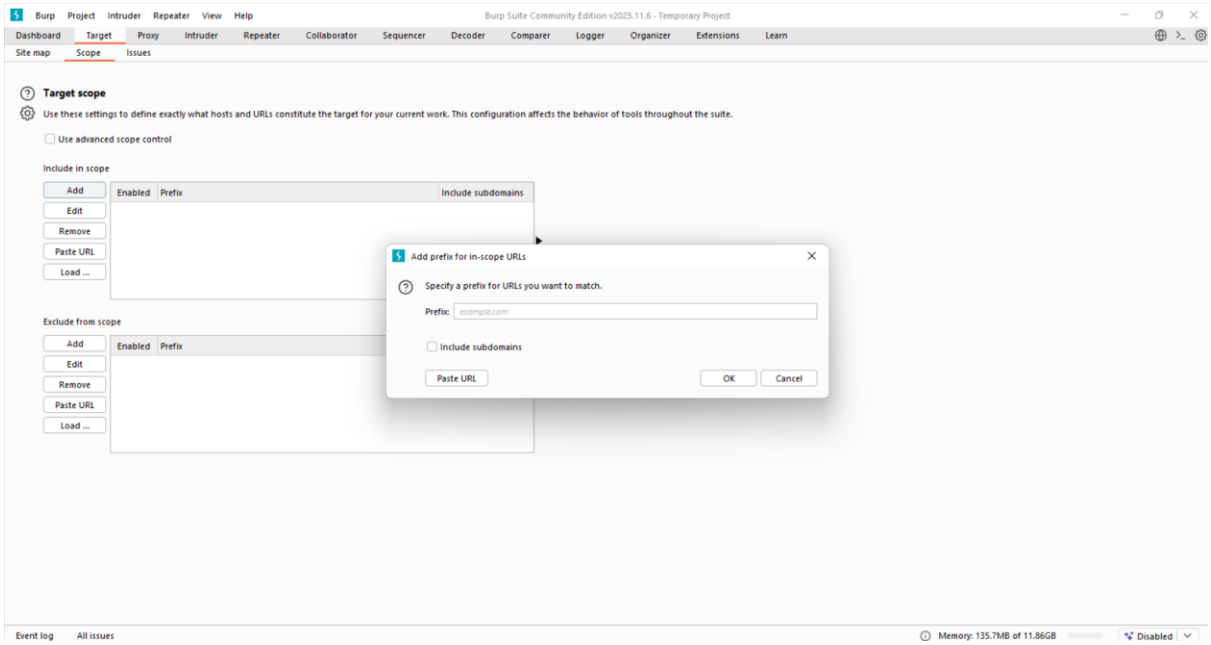
Step2: Open the “target”.



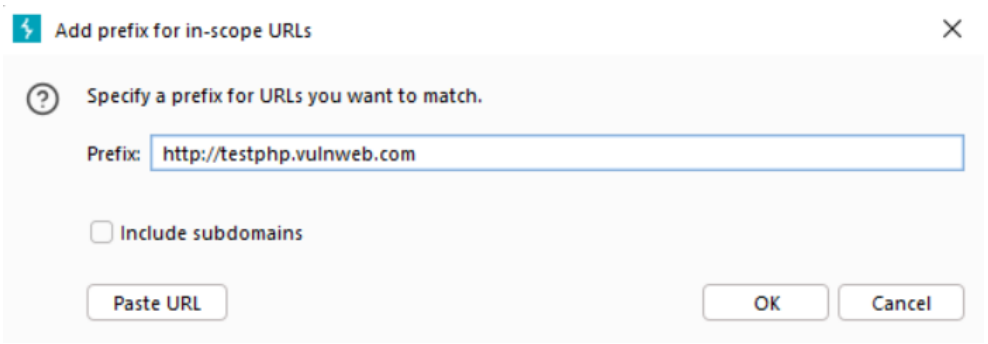
Step3: Click on “Scope”.



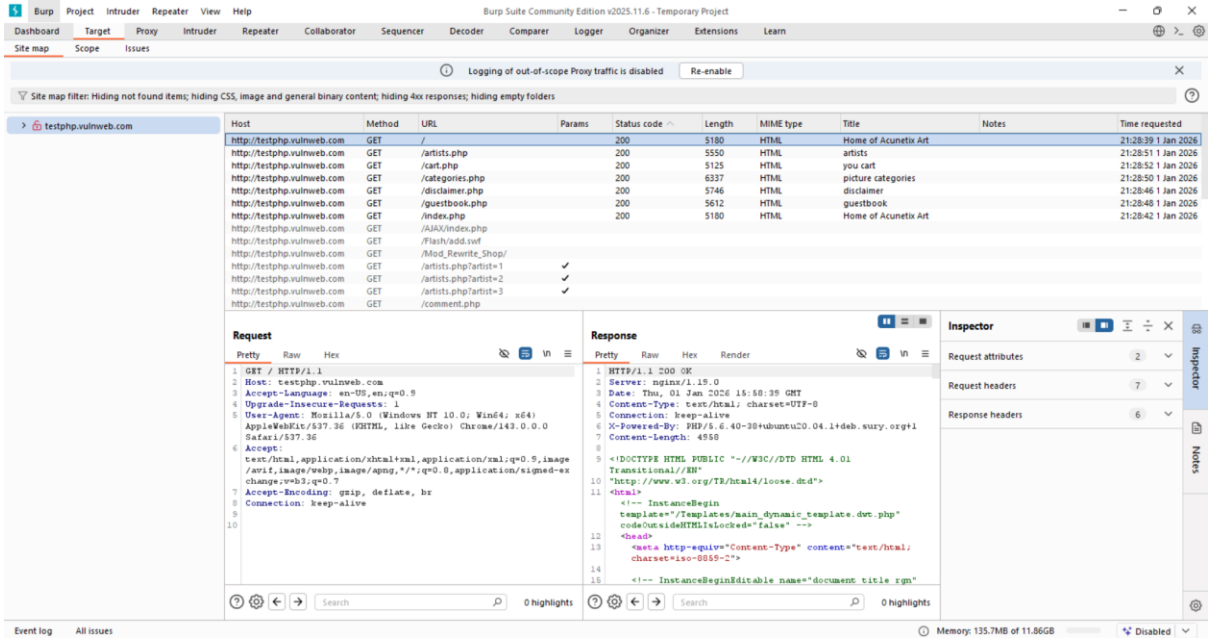
Step4: Click on “Add” button, following input field will appear.



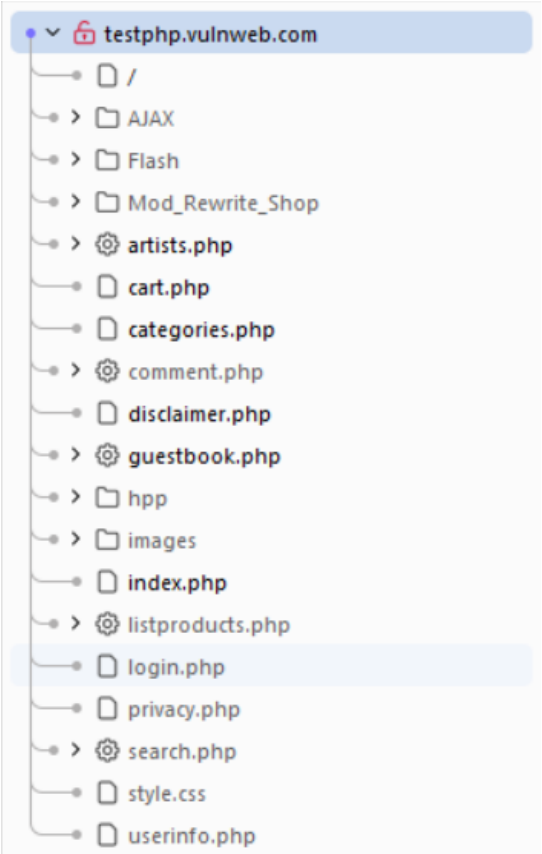
Step5: Write the URL (say http://testphp.vulnweb.com)



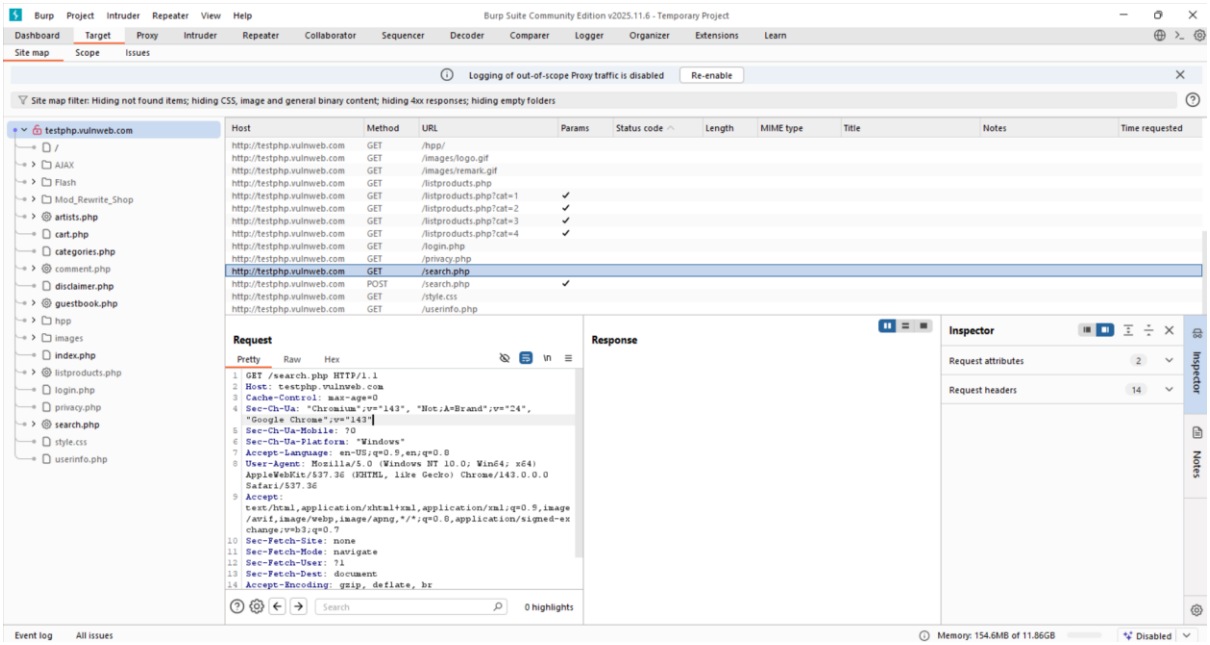
Step6: Browse the Website Normally from the chromium browser opened. Do not attack yet — just generate traffic. Then go to the Target->Site map, following screen can be observed.



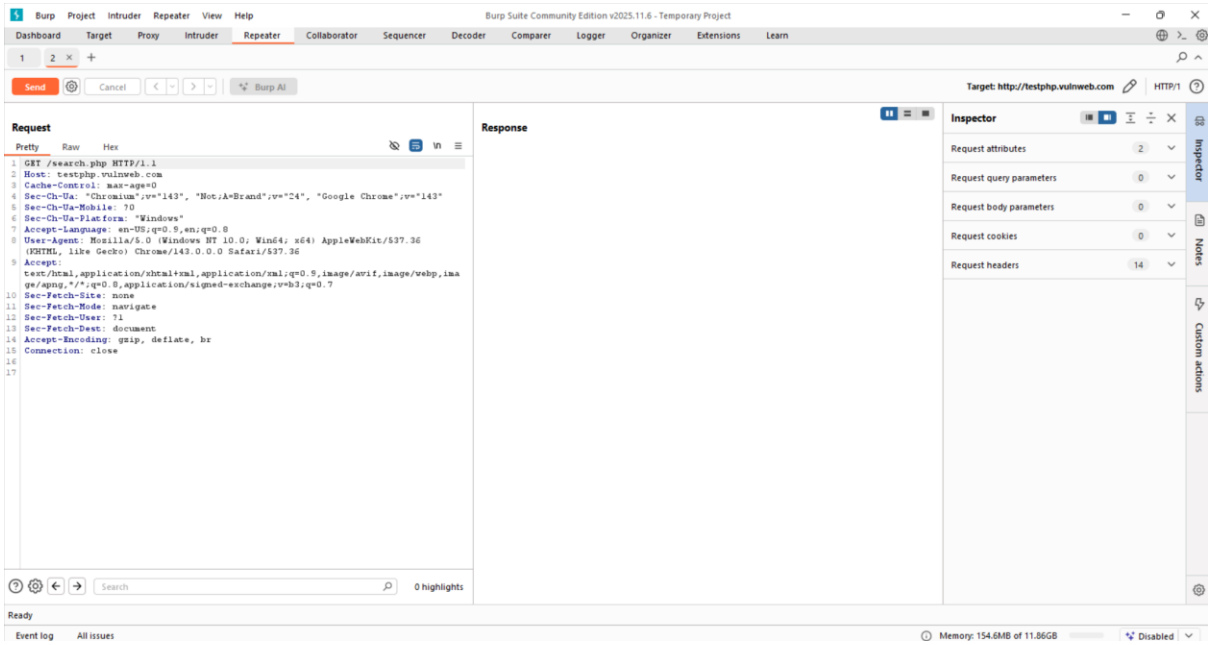
Step7: Review All Discovered Endpoints (Site Map). Go to Target → Site map. These are entry points for external data.



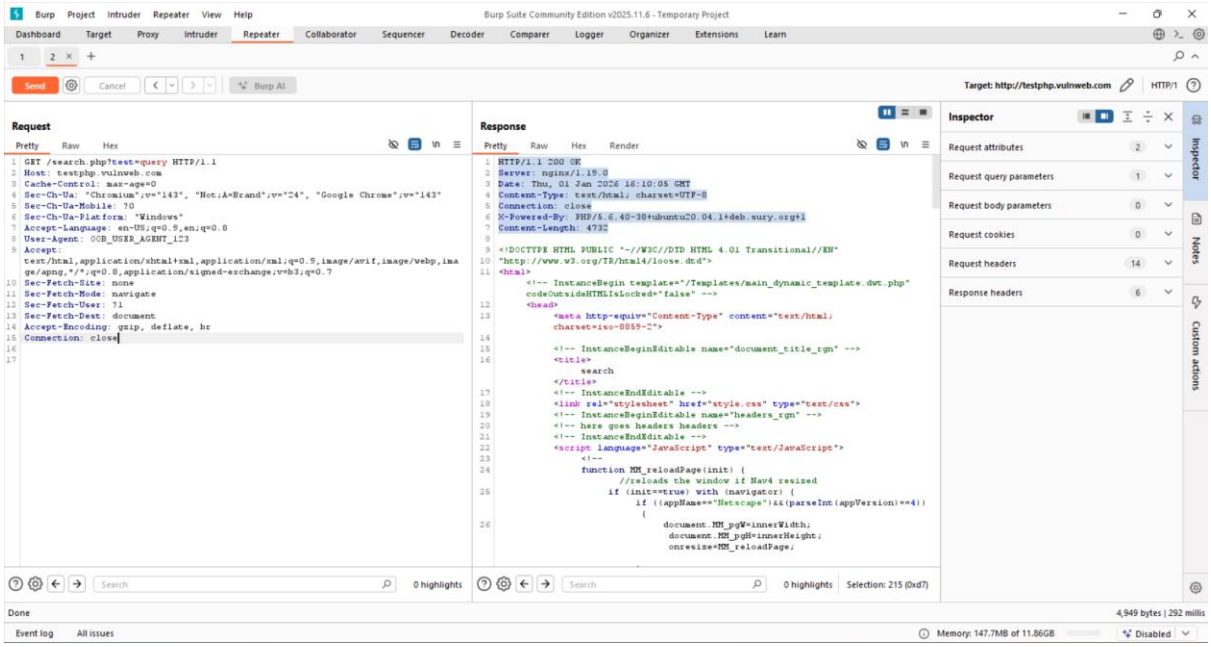
Step8: Pick One Request and Identify Input Channels



Step9: Send that request to repeater. It can be seen there as this:



Step10: Change the parameter “User-Agent” as shown below:



- Not part of UI
- Still user-controlled
- Accepted silently

That is the definition of an out-of-bound channel.

The application accepts user-controlled data via non-standard input channels. During testing of /search.php, modification of HTTP headers such as User-Agent was accepted by the server without validation or rejection, confirming the presence of out-of-bound data ingestion paths beyond intended request parameters.

Note: Inspection of HTML source code revealed comments that may disclose application structure or development artifacts.

Identifying Server-Side Functionality:

- Dissecting Requests
- Extrapolating Application Behaviour

Dissecting Requests

What "Dissecting Requests" Means?

Looking closely at a web address or request to understand what the application is doing behind the scenes, just by reading the URL or request fields. We are basically reading clues left by the developers.

Dissecting requests means carefully reading URLs and requests to understand how a web application works and where user input is used. By looking at file extensions like .jsp, .aspx, or .php, we can tell what technology the application uses.

The parameters in the URL give clues about what happens in the background, such as searching a database, loading files, or sending emails. For example, parameters like OrderBy may control database queries, isExpired or edit may control access or behavior, and parameters like template or loc may refer to files or folders on the server. If these values can be changed by users, they may lead to security problems such as unauthorized access, file exposure, or misuse of email functions.

Extrapolating Application Behaviour:

Extrapolating application behaviour means learning how one part of an application works and then using that knowledge to understand and test other parts, because many applications reuse the same validation rules, encoding methods, and error-handling logic throughout.

Mapping the Attack Surface:

"Mapping the attack surface" means looking at all the places where a hacker might try to break into or misuse an application. You check each feature of the app and think: *What could go wrong here?*

Common areas and their risks:

- Client-side checks – Can be bypassed if the server doesn't verify.
- Database – Risk of SQL injection.
- File upload/download – Risk of accessing wrong files.
- User input display – Risk of cross-site scripting (XSS).
- Redirects – Can send users to fake sites.
- Login systems – Weak passwords or brute-force attacks.
- Sessions – Stolen or predictable session IDs.
- Access control – Users accessing data they shouldn't.
- Error messages – Leaking sensitive information.
- Third-party components – Known security flaws.

In short: it helps identify weak points so they can be secured.