

Chapter 4



WEB APPLICATION SECURITY

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:

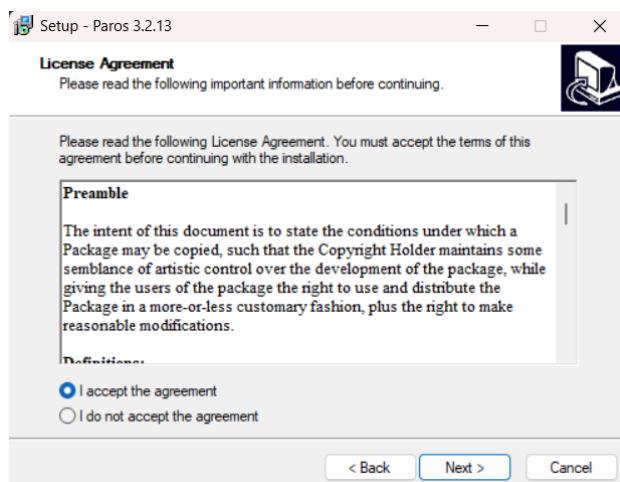
In most applications, most content and features can be found by manually browsing from the main page and following all links and multi-step processes like registration or password reset. A site map, if available, helps in listing the content. However, to fully and accurately inspect and record everything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

Web Spidering:

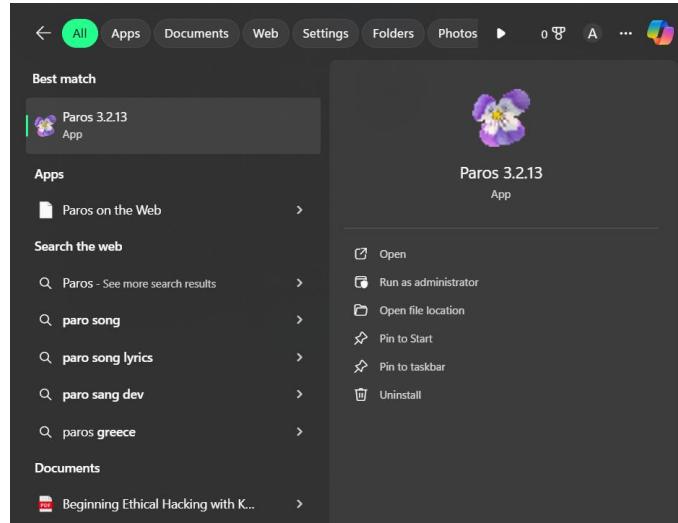
Web spidering uses automated tools to crawl a web application by following links, submitting forms, and sometimes analysing JavaScript to discover content and functionality. Tools like Paros, Burp Spider, and WebScarab help map applications more thoroughly than manual browsing and may even use URLs from 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:

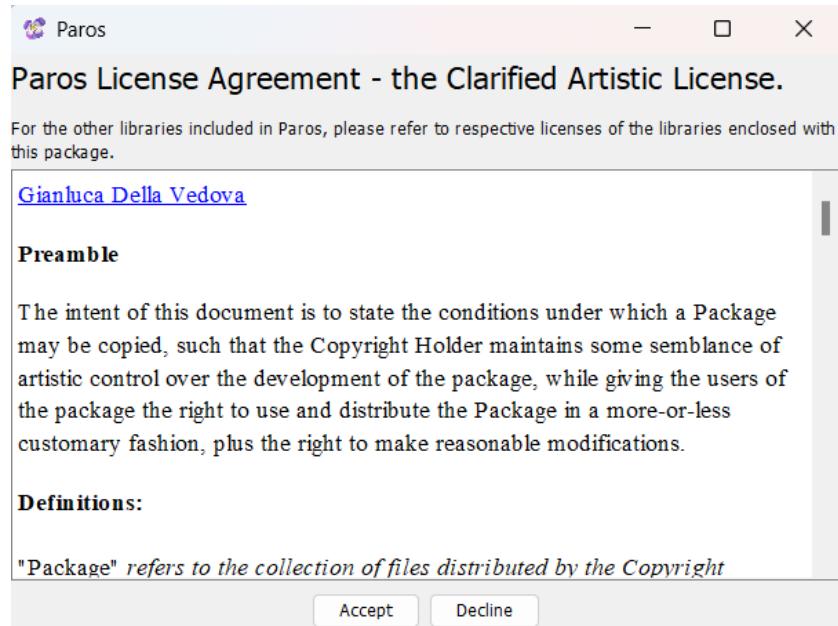
Install the Paros:



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:

A screenshot of the Apify Website Content Crawler tool. The interface is divided into several sections. On the left, there is a sidebar with navigation links such as 'Get started', 'Apify Store', 'Home', 'Actors', 'Runs', 'Saved tasks', 'Integrations', 'Schedules', 'Development', 'My Actors', 'Insights', and 'Messaging'. Below this sidebar, there is information about RAM usage: '8 GB / 8 GB' and 'Usage \$0.06 / \$5.00', along with a link to 'Upgrade to Starter'. The main central area shows a 'Run' titled 'Website Content Crawler Actor'. It displays the status 'Running Crawled 2/519 pages, 0 failed requests, desired concurrency 1' with a cost of '\$0.093'. Below this, there is a table with two rows of data. The first row corresponds to the URL 'https://indianexpress.com/' and the second row to 'https://indianexpress.com/archives/'. Both rows show extracted text and a preview of the crawled content. At the bottom of the main area, there is a 'Items per page:' dropdown set to '50' and a 'Go' button. Above the main table, there are tabs for 'Text', 'Markdown', and 'All fields'. At the very top of the interface, there are buttons for 'Run', 'API', 'Share', 'Export', and 'Abort'.

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

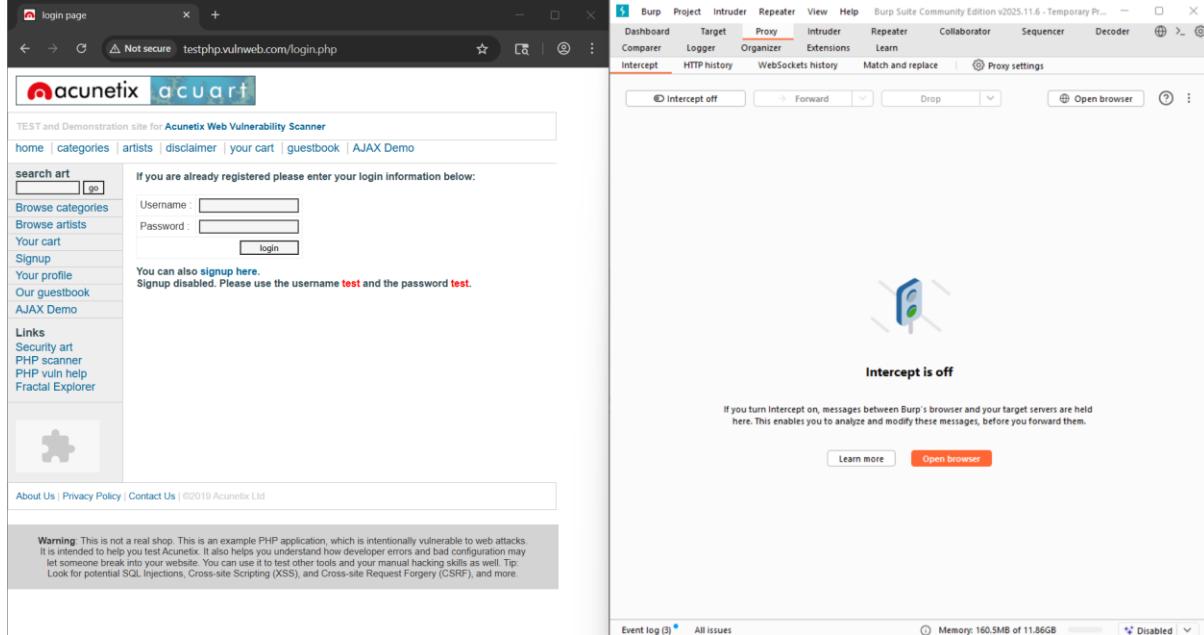
Brute-Force Techniques:

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

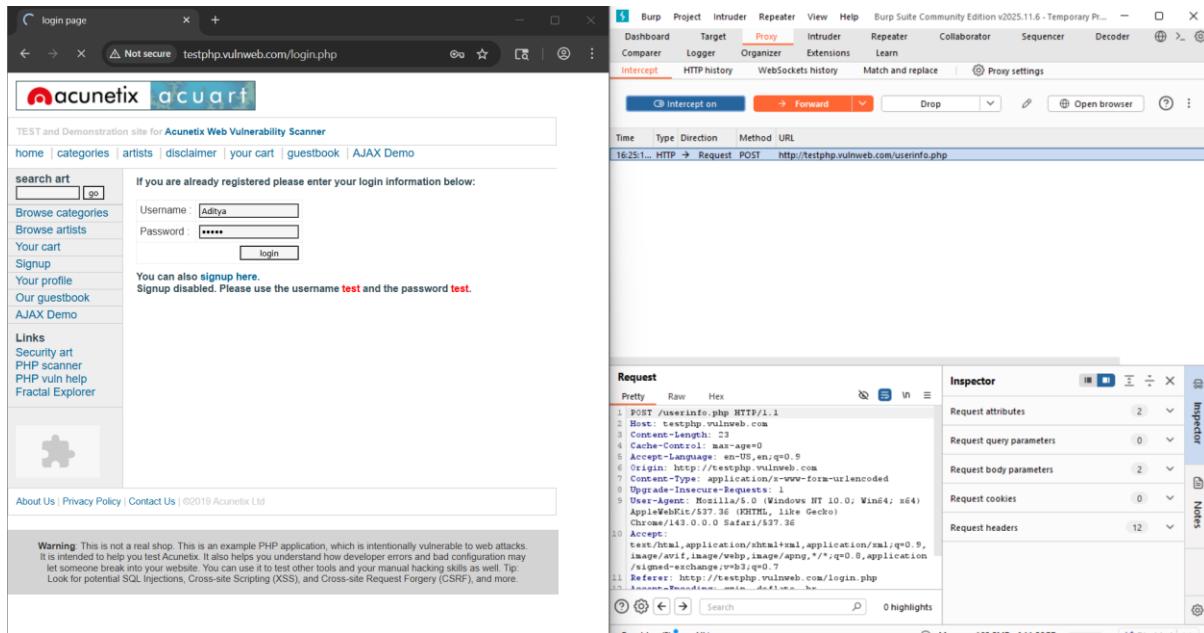
combines automation with manual review of server responses to identify hidden or restricted content.

Brute forcing password using Burp Suite:

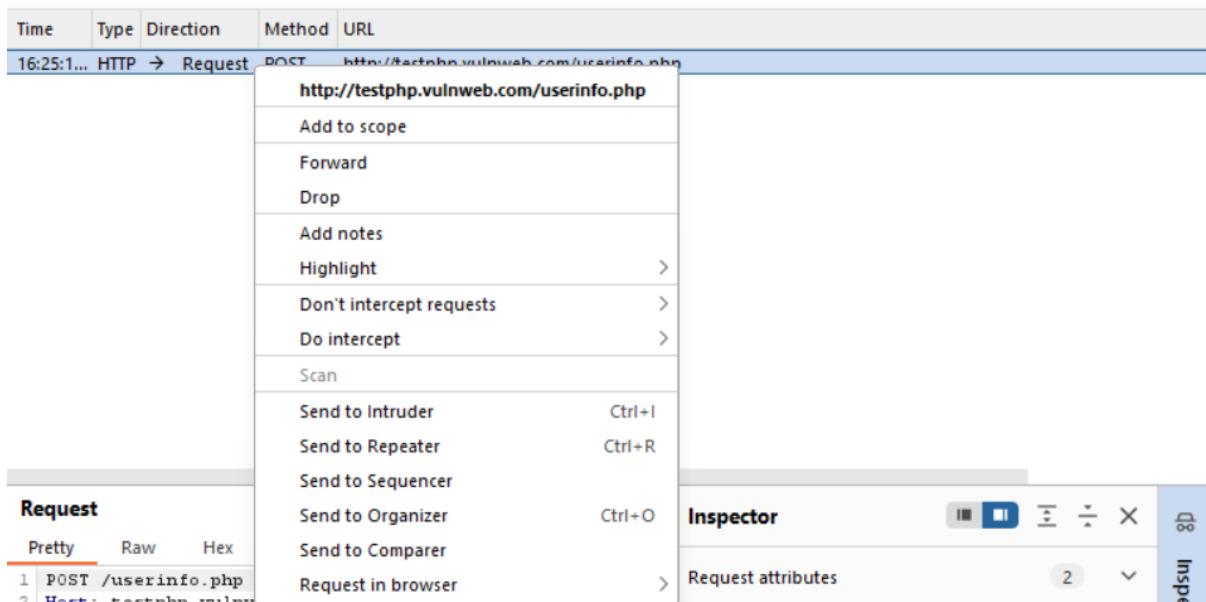
Step1: Open burp suite and the login page (keep intercept off).



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.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. The captured POST request to 'http://testphp.vulnweb.com' is displayed. At the top left of the request list, there are buttons for 'Add \$', 'Clear \$', and 'Auto \$'. The request body contains the following data:

```
1 POST /userinfo.php HTTP/1.1
2 Host: testphp.vulnweb.com
3 Content-Length: 23
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://testphp.vulnweb.com
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 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
10 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
11 Referer: http://testphp.vulnweb.com/login.php
12 Accept-Encoding: gzip, deflate, br
13 Connection: keep-alive
14
15 uname=admin&pass=Rumar
```

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

```
14
15 uname=admin&pass=$admin$
```

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

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. A dropdown menu is open under the 'Sniper attack' section, listing several attack types: Battering ram attack, Pitchfork attack, Cluster bomb attack, and Oracle. The 'Cluster bomb attack' is currently selected. On the right side of the interface, there is a sidebar with sections for 'Payloads', 'Resource pool', and 'Settings'. A red box highlights the 'Start attack' button in the top right corner of the main window.

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

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. The 'Payloads' section is open in the right sidebar. It displays a configuration panel for a 'Simple list' payload type. The 'Payload position' is set to 'All payload positions', and the 'Payload count' is 0. Below this is a 'Payload configuration' panel with buttons for Paste, Load..., Remove, Clear, and Deduplicate. There is also a text input field for 'Enter a new item' and a dropdown for 'Add from list... [Pro version only]'. At the bottom of the sidebar, there is a 'Payload processing' section with buttons for Add, Edit, Remove, Up, and Down. The status bar at the bottom indicates 'Event log (3) All Issues' and 'Memory: 174.4MB of 11.86GB'.

Step8: Add some passwords.

The screenshot shows the 'Payload configuration' dialog box. It contains a text area with a list of strings: 'admin', '1234', 'password', 'abcd', and 'qwer'. To the left of this area is a vertical toolbar with buttons for Paste, Load..., Remove, Clear, and Deduplicate. Below the text area is an 'Add' button and an 'Enter a new item' input field. At the bottom is a dropdown menu labeled 'Add from list... [Pro version only]'.

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

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		302	288			258	
1	admin	302	287			258	
2	1234	302	293			258	
3	password	302	288			258	
4	abcd	302	280			258	
5	qwer	302	294			258	
6	test	200	291			6219	

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”.

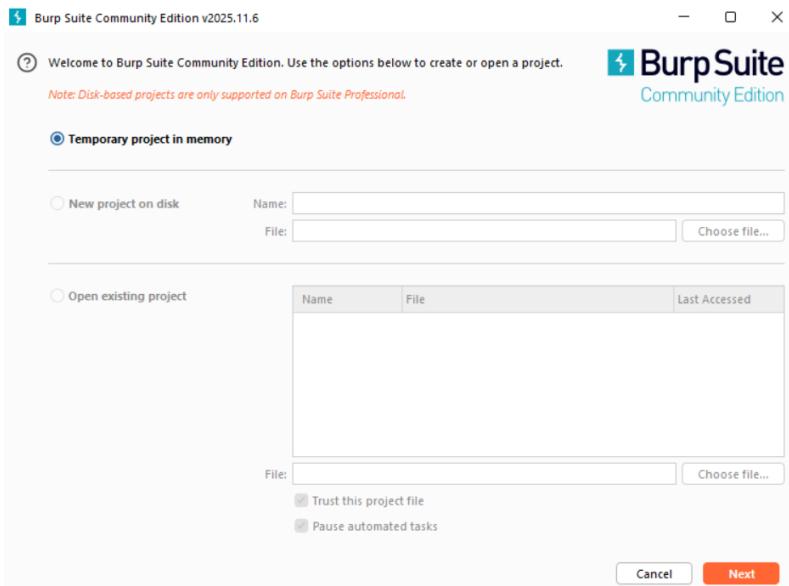
Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		302	288			258	
1	admin	302	287			258	
2	1234	302	293			258	
3	password	302	288			258	
4	abcd	302	280			258	
5	qwer	302	294			258	
6	test	200	291			6219	

Request Response

```
Pretty Raw Hex
1 POST /userinfo.php HTTP/1.1
2 Host: testphp.vulnweb.com
3 Content-Length: 20
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.5
6 Origin: http://testphp.vulnweb.com
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 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
10 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
11 Referer: http://testphp.vulnweb.com/login.php
12 Accept-Encoding: gzip, deflate, br
13 Connection: keep-alive
14
15 uname=test&pass=test
```

Brute spidering using Burp Suite:

Step1: open the Burp suite



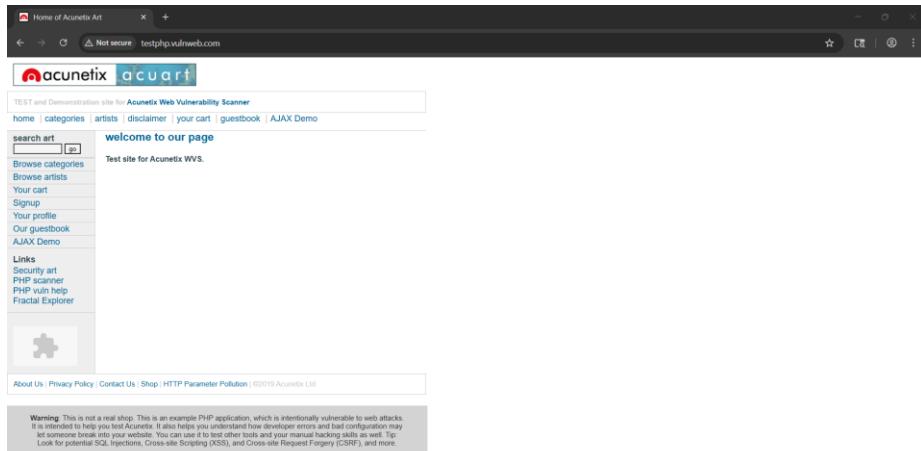
Step2: Following screen will appear.

The screenshot shows the 'Proxy' tab of the Burp Suite interface. It displays a 'Live passive crawl from Proxy (all traffic)' task. The 'Summary' section says 'No items to show'. The 'Task configuration' section shows 'Task type: Live passive crawl', 'Scope: Proxy (all traffic)', and 'Configuration: Add item itself, same domain and URLs in suite scope'. The 'Task progress' section shows 'Site map items added: 0', 'Responses processed: 0', and 'Responses queued: 0'. The 'Task log' section is empty.

Step3: Go to the ‘proxy’ section, and click on the “open browser” button.

The screenshot shows the 'Proxy' tab of the Burp Suite interface. The 'Intercept' button is off. The 'Open browser' button is highlighted in red. There is also a 'Forward' and a 'Drop' 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.

The screenshot shows the Burp Suite interface. The top navigation bar has 'Intruder' selected. The main window displays a single request: a GET /\$dir\$/ HTTP/1.1 to the target http://testphp.vulnweb.com. The 'Payloads' panel on the right is open, showing a simple list payload type with an empty list. A message at the top of the payloads panel says: "To get started, highlight the part of the request or target you want to replace, then click Add \$ to set a payload position." Below the payloads panel are buttons for 'Close', 'Learn more', and 'Don't show this again'.

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

```
1 GET /$dir$/ HTTP/1.1
2 Host: testphp.vulnweb.com
3 Cache-Control: max-age=0
4 Accept-Language: en-US,en;q=0.9
5 Upgrade-Insecure-Requests: 1
6 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
7 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
8 Accept-Encoding: gzip, deflate, br
9 Connection: keep-alive
10
11
```

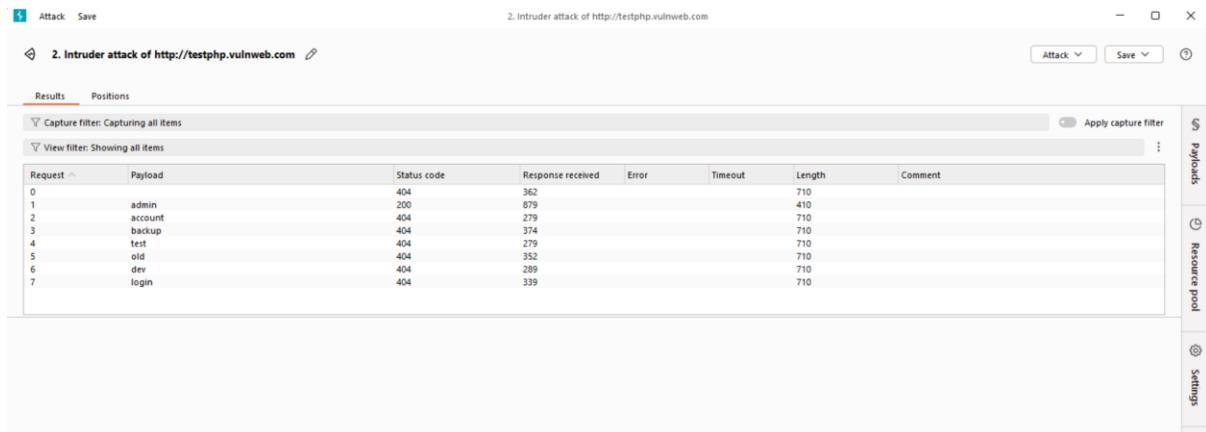
Step9: make sure to select the “sniper” in the attack type.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. The 'Attack type' dropdown is set to 'Sniper attack'. The payloads panel on the right shows a 'Simple list' payload type with several items listed: 'admin', 'account', 'backup', 'test', 'old', 'dev', and 'login'. There are buttons for Paste, Load..., Remove, Clear, Deduplicate, Add, and Enter a new item. A note at the bottom of the payloads panel says: "This payload type lets you configure a simple list of strings that are used as payloads."

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

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. The 'Attack type' dropdown is set to 'Sniper attack'. The payloads panel on the right shows a 'Simple list' payload type with several items listed: 'admin', 'account', 'backup', 'test', 'old', 'dev', and 'login'. The 'Payload configuration' section below the payloads panel lists these same items. A note at the bottom of the payloads panel says: "This payload type lets you configure a simple list of strings that are used as payloads."

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



The screenshot shows the Burp Suite interface with the "Attack" tab selected. A table titled "2. Intruder attack of http://testphp.vulnweb.com" displays the results of an attack. The columns are Request, Payload, Status code, Response received, Error, Timeout, Length, and Comment. The data shows 7 rows of requests, all with a status code of 404 except for row 1 which has a status code of 200. The "Results" tab is selected in the top navigation bar.

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.

```
1 GET /admin/$file$ HTTP/1.1
2 Host: testphp.vulnweb.com
3 Cache-Control: max-age=0
4 Accept-Language: en-US,en;q=0.9
5 Upgrade-Insecure-Requests: 1
6 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
7 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
8 Accept-Encoding: gzip, deflate, br
9 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.

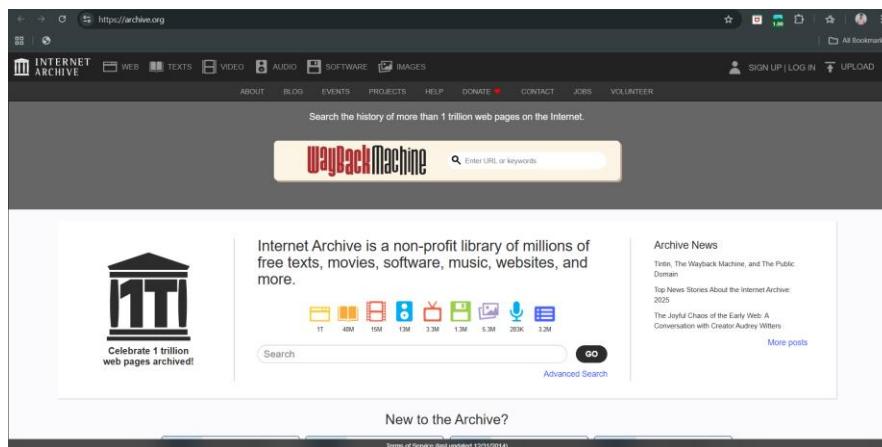
Use of Public Information:

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.

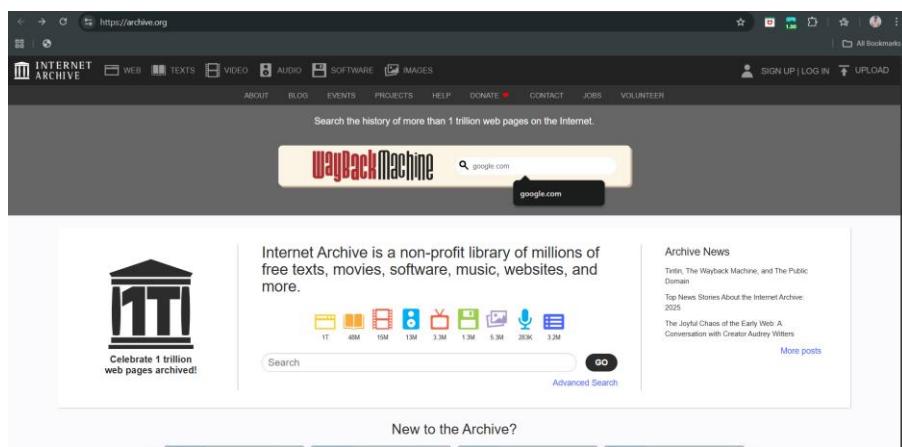
Using way back machines to get an idea of website:

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

Step2: In the appeared window, fill the details.



After filling details:



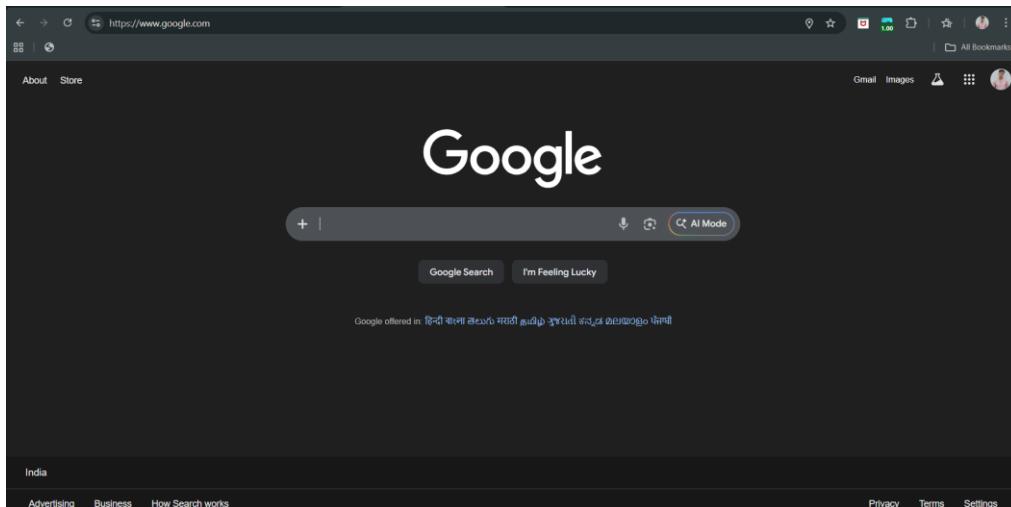
Step3: Following screen will appear, browse to which ever section you wish.



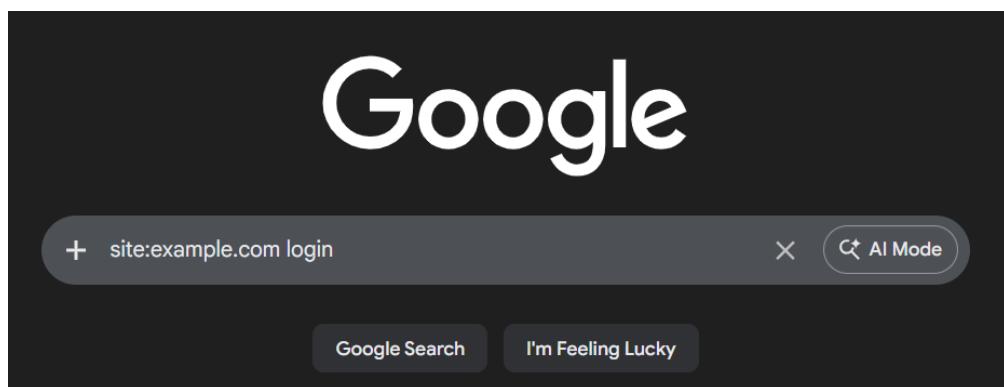
Using google dorks to get an idea of website:

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

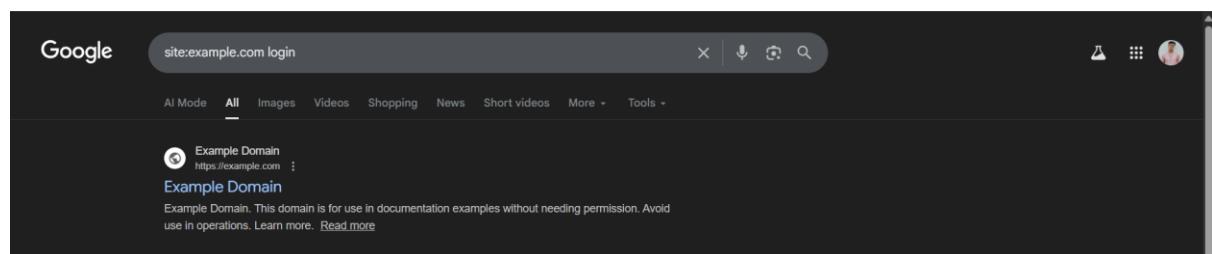
Step2: Following screen will appear.



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

<i>site:example.com "change password"</i>	<i>site:example.com TODO</i>	<i>site:example.com api</i>
<i>site:example.com "db_password"</i>	<i>site:example.com FIXME</i>	<i>site:example.com "api/v1"</i>
<i>site:example.com "database"</i>	<i>site:example.com "under construction"</i>	<i>site:example.com swagger</i>
<i>site:example.com ext:log</i>	<i>site:example.com ext:pdf</i>	<i>site:example.com graphql</i>
<i>site:example.com debug</i>	<i>site:example.com ext:doc</i>	<i>link:example.com</i>
<i>site:example.com error</i>	<i>site:example.com ext:docx</i>	<i>related:example.com</i>
<i>site:example.com warning</i>	<i>site:example.com ext:xls</i>	<i>cache:example.com</i>
	<i>site:example.com ext:xlsx</i>	
	<i>site:example.com ext:ppt</i>	

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.

Using Nikto for scanning vulnerabilities: (Part of Leveraging the Web Server)

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

```
(root㉿kali)-[~]
# nikto -h
Option host requires an argument

Options:
  -ask+           Whether to ask about submitting updates
                  yes  Ask about each (default)
                  no   Don't ask, don't send
                  auto Don't ask, just send
  -check6         Check if IPv6 is working (connects to ipv6.google.com or value set in nikto.conf)
  -Cgidirs+       Scan these CGI dirs: "none", "all", or values like "/cgi /cgi-a/"
  -config+        Use this config file
  -Display+       Turn on/off display outputs:
```

Step2: specify the target.

Command: *nikto -h http://testphp.vulnweb.com/*

```
(root㉿kali)-[~]
# nikto -h http://testphp.vulnweb.com/
- Nikto v2.5.0
-----
+ Target IP:      44.228.249.3
+ Target Hostname: testphp.vulnweb.com
+ Target Port:    80
+ Start Time:    2025-12-30 10:18:29 (GMT5.5)
-----
+ Server: nginx/1.19.0
+ /: Retrieved x-powered-by header: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1.
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-FRAME-OPTIONS
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion-content-type-header/
```

Application Pages vs. Functional Paths:

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 URL-based mapping can be misleading. Mapping functional paths instead shows the real flow of actions, relationships between functions, and possible security weaknesses.

Identify parameter-based functionality using Burp Suite:

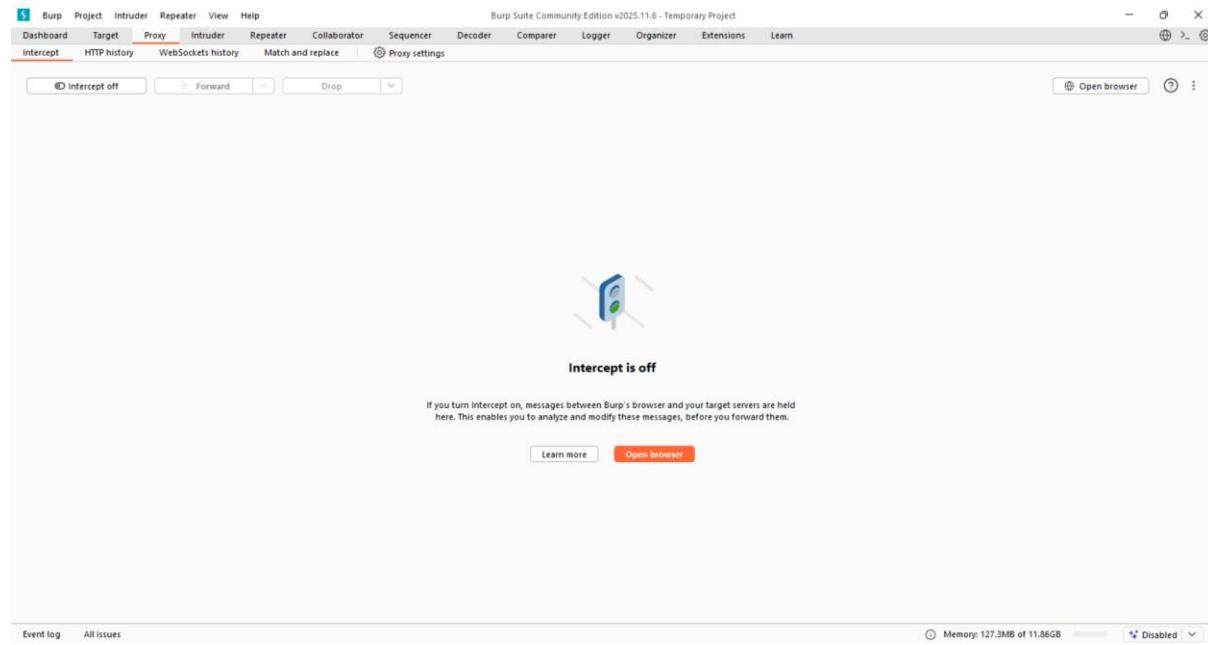
Goal: You want to identify functionality controlled by parameters, not by separate pages.

Example: /index.php?page=login

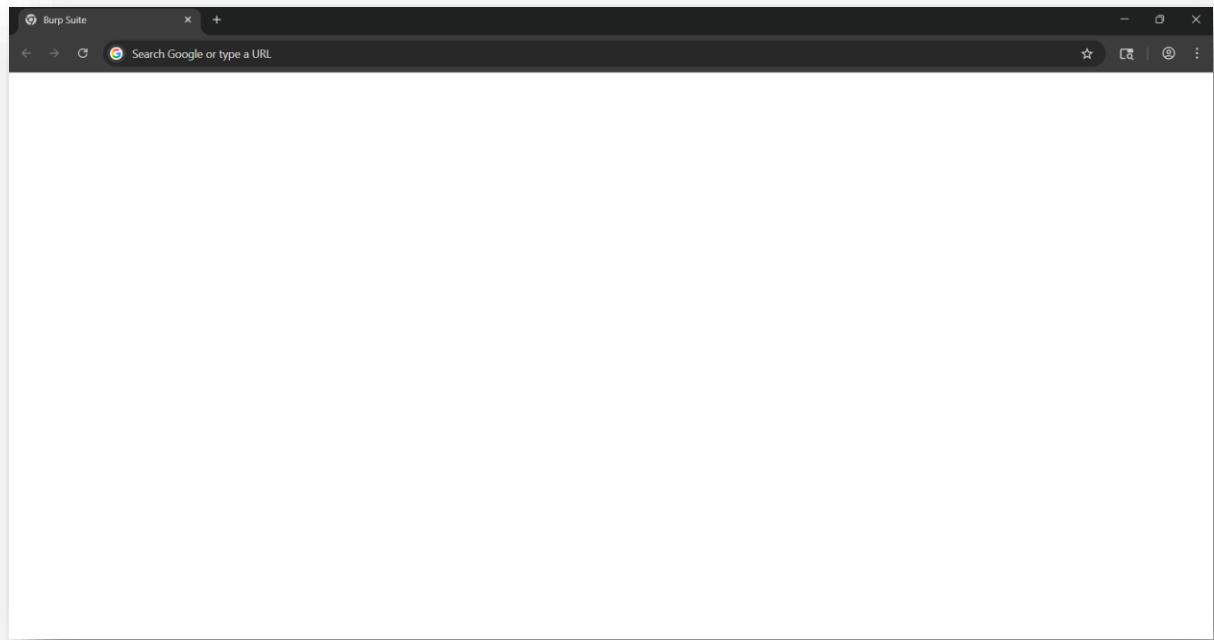
Instead of: /login.php

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:

The screenshot shows a web browser window with the URL <http://testphp.vulnweb.com>. The page is titled "Welcome to our page" and contains a search bar with placeholder "search art" and a "go" button. On the left, there's a sidebar with links: "Browse categories", "Browse artists", "Your cart", "Signup", "Your profile", "Our guestbook", "AJAX Demo", and "Links" (Security art, PHP scanner, PHP vuln help, Fractal Explorer). The main content area features a large puzzle piece icon and a warning message: "Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL Injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more." At the bottom, there are links for "About Us", "Privacy Policy", "Contact Us", "Shop", and "HTTP Parameter Pollution".

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

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response...
1	http://testphp.vulnweb.com	GET	/			200	5180	HTML		Home of Acunetix Art		44.228.249.3			15:18:04 30 ...	8080	278
2	http://testphp.vulnweb.com	GET	/categories.php			200	6337	HTML	php	picture categories		44.228.249.3			15:18:05 30 ...	8080	283
3	http://testphp.vulnweb.com	GET	/artists.php			200	5550	HTML	php	artists		44.228.249.3			15:18:10 30 ...	8080	286
4	http://testphp.vulnweb.com	GET	/disclaimer.php			200	5746	HTML	php	disclaimer		44.228.249.3			15:18:12 30 ...	8080	284
5	http://testphp.vulnweb.com	GET	/cart.php			200	5125	HTML	php	you cart		44.228.249.3			15:18:14 30 ...	8080	286
6	http://testphp.vulnweb.com	GET	/guestbook.php			200	5612	HTML		guestbook		44.228.249.3			15:18:16 30 ...	8080	286
8	http://testphp.vulnweb.com	GET	/AJAX/index.php			200	4458	HTML	php	ajax test		44.228.249.3			15:18:18 30 ...	8080	285
10	http://testphp.vulnweb.com	GET	//			200	5180	HTML		Home of Acunetix Art		44.228.249.3			15:18:35 30 ...	8080	283
13	http://testphp.vulnweb.com	GET	/			200	5180	HTML		Home of Acunetix Art		44.228.249.3			15:18:42 30 ...	8080	285

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`)

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response...
1	http://testphp.vulnweb.com	GET	/			200	5180	HTML		Home of Acunetix Art		44.228.249.3			15:18:04 30 ...	8080	278
2	http://testphp.vulnweb.com	GET	/categories.php			200	6337	HTML	php	picture categories		44.228.249.3			15:18:08 30 ...	8080	283
3	http://testphp.vulnweb.com	GET	/artists.php			200	5550	HTML	php	artists		44.228.249.3			15:18:10 30 ...	8080	286
4	http://testphp.vulnweb.com	GET	/disclaimer.php			200	5746	HTML	php	disclaimer		44.228.249.3			15:18:12 30 ...	8080	284
5	http://testphp.vulnweb.com	GET	/cart.php			200	5125	HTML	php	you cart		44.228.249.3			15:18:14 30 ...	8080	286
6	http://testphp.vulnweb.com	GET	/guestbook.php			200	5612	HTML	php	guestbook		44.228.249.3			15:18:16 30 ...	8080	286
8	http://testphp.vulnweb.com	GET	/AJAX/index.php			200	4458	HTML	php	ajax test		44.228.249.3			15:18:18 30 ...	8080	285
10	http://testphp.vulnweb.com	GET	//			200	5180	HTML		Home of Acunetix Art		44.228.249.3			15:18:35 30 ...	8080	283
13	http://testphp.vulnweb.com	GET	/			200	5180	HTML		Home of Acunetix Art		44.228.249.3			15:18:42 30 ...	8080	285
14	http://testphp.vulnweb.com	GET	/categories.php			200	6337	HTML	php	picture categories		44.228.249.3			15:21:59 30 ...	8080	352
15	http://testphp.vulnweb.com	GET	/listproducts.php?cat=1		✓	200	8102	HTML	php	pictures		44.228.249.3			15:22:01 30 ...	8080	294

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.

The screenshot shows the ZAP interface with the 'Repeater' tab selected. The main pane displays a list of captured requests from the target URL `http://testphp.vulnweb.com`. One specific request is highlighted, showing a modified parameter `cat=1` in the URL. The 'Inspector' tab on the right shows the request attributes, query parameters, body parameters, cookies, and headers for this selected request.

Request ID	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response
1	GET	/listproducts.php			200	5180	HTML	php	Home of Acunetix Art			44.228.249.3		15:18:34 30 ...	8080	278
2	GET	/listproducts.php			200	6337	HTML	php	picture categories			44.228.249.3		15:18:39 30 ...	8080	283
3	GET	/categories.php			200	5550	HTML	php	artists			44.228.249.3		15:18:10 30 ...	8080	286
4	GET	/about.php			200	5746	HTML	php	disclaimer			44.228.249.3		15:18:12 30 ...	8080	284
5	GET	/disclaimer.php			200	5125	HTML	php	you cart			44.228.249.3		15:18:14 30 ...	8080	286
6	GET	/guestbook.php			200	5612	HTML	php	guestbook			44.228.249.3		15:18:15 30 ...	8080	286
8	GET	/AJAX/index.php			200	4458	HTML	php	ajax test			44.228.249.3		15:18:19 30 ...	8080	285
10	GET	//			200	5180	HTML	php	Home of Acunetix Art			44.228.249.3		15:18:33 30 ...	8080	283
13	GET	/			200	5180	HTML	php	Home of Acunetix Art			44.228.249.3		15:18:42 30 ...	8080	285
14	GET	/categories.php			200	6337	HTML	php	picture categories			44.228.249.3		15:21:59 30 ...	8080	282
15	GET	/listproducts.php?cat=1	✓		200	8102	HTML	php	pictures			44.228.249.3		15:22:01 30 ...	8080	284

In repeater this request can be seen as:

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. The 'Request' pane shows the captured request to `http://testphp.vulnweb.com/listproducts.php?cat=1`. The 'Response' pane shows the server's response. The 'Inspector' tab on the right shows the request attributes, query parameters, body parameters, cookies, and headers for this selected request.

Step6: Change cat = 1 to cat = 2.

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. The 'Request' pane shows the modified request where the parameter `cat` has been changed to `cat=2`. The 'Response' pane shows the server's response. The 'Inspector' tab on the right shows the request attributes, query parameters, body parameters, cookies, and headers for this selected request.

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

The screenshot shows the Burp Suite interface. In the Request tab, a GET request to `/listproducts.php?cat=1` is displayed. The response shows a painting titled "Thing" by r4w6173. The Inspector panel on the right shows the request attributes, query parameters (cat), and body parameters (empty). The response content includes the painting image and its details.

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 our baseline request.

The screenshot shows the Burp Suite interface with the Intercept tab selected. A list of captured requests is shown, including one for `/listproducts.php?cat=1`. The requests table includes columns for Host, Method, URL, Params, Status code, Length, MIME type, Extension, Title, Notes, TLS, IP, Cookies, Time, Listener port, and Start response.

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

The screenshot shows the Burp Suite interface with the Repeater tab selected. A request to `/listproducts.php?cat=1` is selected and ready to be sent. The response pane shows the same painting as before. The Inspector panel on the right shows the request attributes, query parameters (cat), and body parameters (empty).

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

The screenshot shows the Burp Suite interface with the following details:

Request

```
1 GET /listproducts.php?cat=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,image/*
   /*,*/*;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
11 |
```

Response

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.0
3 Date: Tue, 30 Dec 2025 10:33:00 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
7 Content-Length: 7080
8
9 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 4.01 Transitional//EN"
10 <html>
11   <head>
12     <!-- InstanceBegin template="/Templates/main_dynamic_template.dwt.php"
        codeOutsideHTMLLocked="false" -->
13     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
14
15     <!-- InstanceBeginEditable name="document_title_rgn" -->
16     <title>
17       <!-- InstanceEndEditable -->
18       <link rel="stylesheet" href="style.css" type="text/css">
19       <!-- InstanceBeginEditable name="headers_rgn" -->
20       <!-- InstanceEndEditable -->
21       <script language="JavaScript" type="text/JavaScript">
22         <!--
23           function (reloadPage) {
24             if (window.location.reload) {
25               if (!init=true) with (navigator) {
26                 if ((appName=="Netscape")&&(parseInt(appVersion)==4)) {
27                   document.MM_pgW.innerWidth;
28                   document.MM_pgH.innerHeight;
29                   onresize=MM_reloadPage;
30                 }
31               }
32             }
33           else if ((innerWidth!=document.MM_pgW||innerHeight!=document.MM_pgH)
34             ||(innerWidth!=document.MM_pgW||innerHeight!=document.MM_pgH))
35           window.location.reload();
36         }
37       </script>
38     </title>
39   </head>
40   <body>
```

Inspector

Target: http://testphp.vulnweb.com

Request attributes: 2

Request query parameters: 1

Request body parameters: 0

Request cookies: 0

Request headers: 8

Response headers: 6

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

Request

Pretty	Raw	Hex
1 GET /listproducts.php?cat=1&debug=true 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,image/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		

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

The screenshot shows the Burp Suite interface with the following details:

- Project Bar:** Burp, Project, Intruder, Repeater, View, Help.
- Toolbar:** Dashboard, Target, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer, Extensions, Learn.
- Request Section:** Target: http://testphp.vulnweb.com, HTTP/1.1
- Response Section:** Response code: 200 OK, Content-Type: text/html; charset=UTF-8, Content-Length: 7800. The response body contains a large amount of HTML and JavaScript code, including template definitions and a function named MM_reloadPage.
- Inspector Section:** Shows Request attributes, Request query parameters, Request body parameters, Request cookies, Request headers, and Response headers.

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,image/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
11
```

In Intruder it will appear like this:

The screenshot shows the Burp Suite interface. The top navigation bar has tabs for Burp, Project, Intruder, Repeater, View, Help, Dashboard, Target, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer, Extensions, and Learn. The 'Intruder' tab is selected. Below the tabs, there's a search bar and some status indicators: 'Event log All issues', 'Memory: 147.2MB of 11.86GB', and 'Disabled'. The main area shows a recorded request for 'http://testphp.vulnweb.com'. The 'Payloads' panel is open on the right, showing a list of payload positions. A message in the panel says: 'To get started, highlight the part of the request or target you want to replace, then click Add \$ to set a payload position.' Buttons for 'Close', 'Learn more', and 'Don't show this again' are at the bottom of the payloads panel.

Step7: Add the payload.

This screenshot is similar to the previous one, showing the Burp Suite interface with the 'Intruder' tab selected. The recorded request for 'http://testphp.vulnweb.com' is visible. The 'Payloads' panel is open, and a configuration dialog for a 'Simple list' payload type is displayed. The dialog lists several items: debug, test, testing, admin, administrator, isAdmin, root, dev, and developer. Buttons for 'Paste', 'Load...', 'Remove', 'Clear', 'Deduplicate', 'Add', and 'Add from list... [Pro version only]' are shown. The 'Payload processing' section at the bottom is collapsed. Status indicators at the bottom include 'Event log All issues', 'Memory: 132.5MB of 11.86GB', and 'Disabled'.

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

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	280			8102	
1	debug	200	279			8102	
2	test	200	290			8102	
3	testing	200	280			8102	
4	admin	200	283			8102	
5	administrator	200	292			8102	
6	isAdmin	200	290			8102	
7	root	200	280			8102	
8	dev	200	291			8102	
9	developer	200	286			8102	
10	mode	200	285			8102	
11	env	293				8102	
12	environment	200	285			8102	
13	config	200	287			8102	
14	configuration	200	282			8102	
15	settings	200	281			8102	
16	setup	200	281			8102	
17	install	200	285			8102	
18	trace	200	285			8102	

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.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. The 'Payloads' panel on the right lists the following payloads:

- Payload position: All payload positions
- Payload type: Simple list
- Payload count: 56
- Request count: 56

The 'Payload configuration' section shows a list of payloads: debug, test, testing, admin, administrator, root, dev, developer. Below this is a 'Paste' button and a text input field containing the payloads. There are also 'Load...', 'Remove', 'Clear', and 'Duplicate' buttons, along with an 'Add' button and a 'Enter a new item' input field.

The 'Payload processing' section contains a table with columns for 'Add', 'Enabled', and 'Rule'. The table currently has one row with the 'Enabled' checkbox checked and the 'Rule' column empty.

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

```

Positions Add $ Clear $ Auto $ 

1 GET /listproducts.php?cat=1&$param$=$value$ 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,image/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

```

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

Burp Suite Community Edition v2025.11.6 - Temporary Project

Sniper attack
Sniper attack Inserts each payload into each position one at a time, using a single payload set.

Gathering raw attack
Gathering raw attack Simultaneously places the same payload into all positions, using a single payload set.

Pitchfork attack
Pitchfork attack Allocates a payload set to each position. Intruder iterates through each set in parallel.

Cluster bomb attack
Cluster bomb attack Allocates a payload set to each position. Intruder iterates through all possible combinations of each set.

Target: http://testphp.vulnweb.com/categories.php
Request header to match target: change:v=b3;q=0.7

Payloads

Payload position: All payload positions
Payload type: Simple list
Payload count: 56
Request count: 56

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

debug
test
testing
admin
administrator
isadmin
root
dev
developer

Add Enter a new item
Add from list... [Pro version only]

Now, intruder will look like this:

Burp Suite Community Edition v2025.11.6 - Temporary Project

Cluster bomb attack
Target: http://testphp.vulnweb.com
Update Host header to match target

Positions Add \$ Clear \$ Auto \$

1 GET /listproducts.php?cat=1&\$param\$=\$value\$ 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,image/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
11

Payloads

Payload position: 1 - param
Payload type: Simple list
Payload count: 56
Request count: 0

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

debug
test
testing
admin
administrator
isadmin
root
dev
developer

Add Enter a new item
Add from list... [Pro version only]

Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Add Enabled Rule
Edit
Remove
Up
Down

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

Payloads

Payload position: 1 - param

Payload type: 1 - param

Payload count: 56

Request count: 0

Step5: paste the list for the parameter 1.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. In the 'Payloads' panel, a payload list named '1 - param' is displayed with the following items: debug, test, testing, hide, hidden, source, view, viewsource, and show. Below this, a 'Payload configuration' section is visible. The main workspace shows an HTTP request to 'http://testphp.vulnweb.com/listproducts.php?cat=1¶m=\$values\$'. The 'Payloads' panel also shows a 'Payload processing' section with rules for each item.

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

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. In the 'Payloads' panel, a payload list named '2 - value' is displayed with the following items: true, false, yes, no, on, off, 1, 0, and debug. Below this, a 'Payload configuration' section is visible. The main workspace shows the same HTTP request as Step 5. The 'Payloads' panel also shows a 'Payload processing' section with rules for each item.

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

The screenshot shows the 'Results' table from the Burp Suite interface. It displays 18 rows of data corresponding to the payloads listed in Step 6. The columns include Request, Payload 1, Payload 2, Status code, Response received, Error, Timeout, Length, and Comment. All rows have a Length of 8102, indicating that all payloads were processed successfully.

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
0	debug	true	200	280			8102	
1	test	true	200	282			8102	
2	testing	true	200	295			8102	
3	hide	true	200	389			8102	
4	hidden	true	200	285			8102	
5	source	true	200	285			8102	
6	view	true	200	278			8102	
7	viewsource	true	200	282			8102	
8	show	true	200	281			8102	
9	admin	true	200	285			8102	
10	dev	true	200	285			8102	
11	developer	true	200	295			8102	
12	trac	true	200	289			8102	
13	verbose	true	200	292			8102	
14	config	true	200	281			8102	
15	mode	true	200	292			8102	
16	env	true	200	279			8102	
17	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:

```

1 POST /guestbook.php?paramS=$value$ HTTP/1.1
2 Host: testphp.vulnweb.com
3 Content-Length: 56
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://testphp.vulnweb.com
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 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
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://testphp.vulnweb.com/guestbook.php
12 Accept-Encoding: gzip, deflate, br
13 Connection: keep-alive
14
15 name=test&comment=test&paramS=$value$

```

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

The screenshot shows the Burp Suite interface with the following configuration for an Intruder attack:

- Target:** http://testphp.vulnweb.com
- Payload position:** 4 - value
- Payload type:** Simple list
- Payload count:** 6
- Payload configuration:** Contains items: true, 1, on, yes, debug, all.
- Payload processing:** Contains rules: Enabled, Rule.

Step4: Start the attack, and observe the length.

Request	Payload 1	Payload 2	Payload 3	Payload 4	Status code	Response len.	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”?

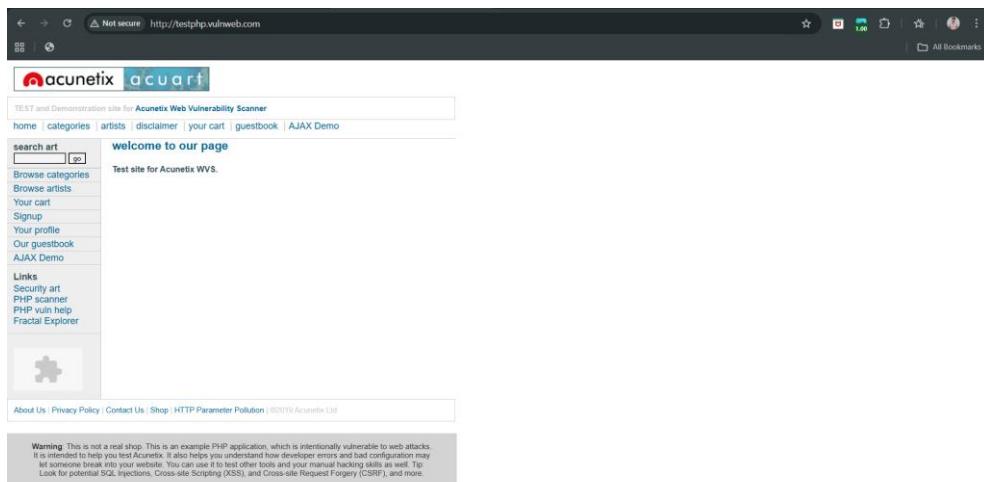
A banner is information sent by a server when it communicates with a browser or client. This information may include:

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

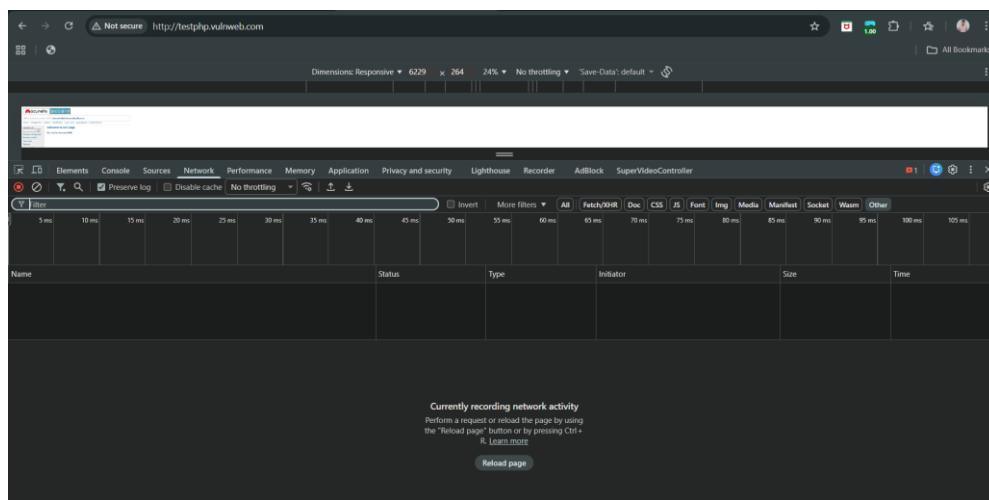
How web servers reveal information? Many web servers send detailed information in the HTTP response headers, especially in the Server header.

Banner Grabbing using Developer tools:

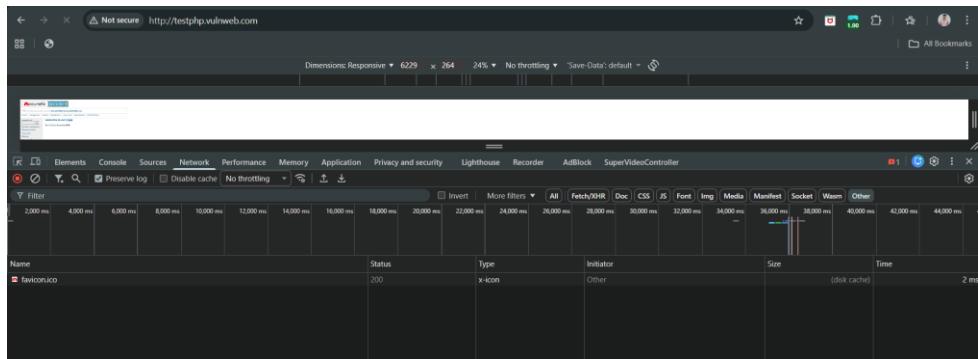
Step1: Open the website. Say <http://testphp.vulnweb.com/>



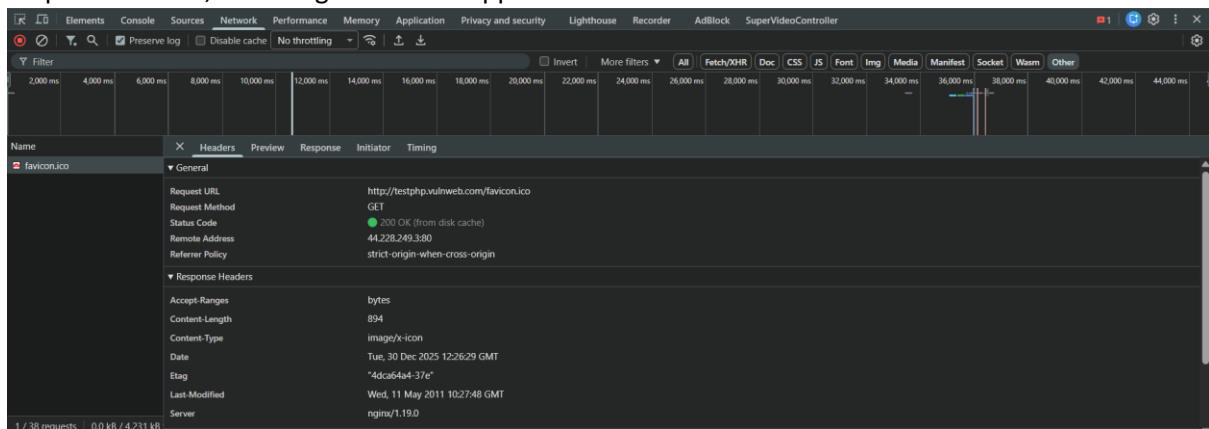
Step2: Open the “network” section: following screen will appear.



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).

▼ Response Headers	
Accept-Ranges	bytes
Content-Length	894
Content-Type	image/x-icon
Date	Tue, 30 Dec 2025 12:26:29 GMT
Etag	"4dca64a4-37e"
Last-Modified	Wed, 11 May 2011 10:27:48 GMT
Server	nginx/1.19.0

The server is using Nginx version 1.19.0.

Banner Grabbing using curl:

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

Banner Grabbing Using telnet or netcat (Manual Method):

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: 9b62fc68e1a3bf2-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
|   X-MTML: instashell
|_ (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
|   X-MTML: social-engl...
|_ (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
      ong.com  steveonly... social-engi... repos
    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:

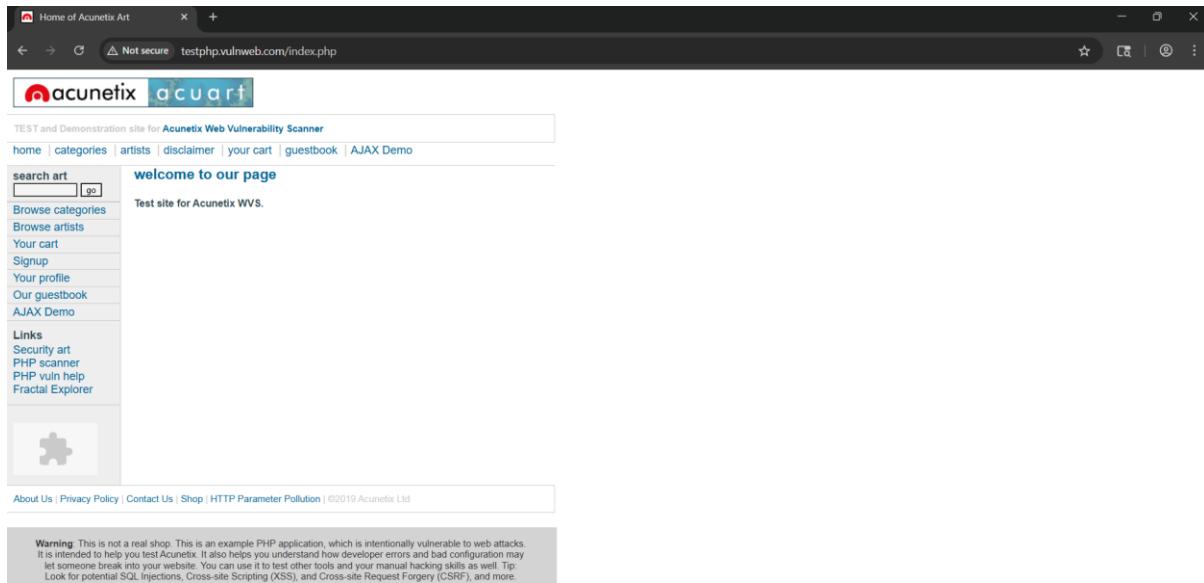
File extensions and server behavior can reveal the technology used by a web application. By passively observing traffic in Burp Suite while browsing testphp.vulnweb.com, we can see URLs ending in .php, PHP session cookies like PHPSESSID, and PHP-style error messages, all of which clearly indicate that 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.

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response...
1	http://testphp.vulnweb.com	GET	/			200	5180	HTML	php	Home of Acunetix Art		44.228.249.3			15:49:25 1 Ja...	8080	291
2	http://testphp.vulnweb.com	GET	/index.php			200	5180	HTML	php	Home of Acunetix Art		44.228.249.3			15:49:28 1 Ja...	8080	293
3	http://testphp.vulnweb.com	GET	/categories.php			200	6337	HTML	php	picture categories		44.228.249.3			15:49:30 1 Ja...	8080	292
4	http://testphp.vulnweb.com	GET	/artists.php			200	5550	HTML	php	artists		44.228.249.3			15:49:32 1 Ja...	8080	290
5	http://testphp.vulnweb.com	GET	/disclaimer.php			200	5746	HTML	php	disclaimer		44.228.249.3			15:49:34 1 Ja...	8080	293
6	http://testphp.vulnweb.com	GET	/cart.php			200	5125	HTML	php	you cart		44.228.249.3			15:49:35 1 Ja...	8080	290
7	http://testphp.vulnweb.com	GET	/guestbook.php			200	5613	HTML	php	guestbook		44.228.249.3			15:49:37 1 Ja...	8080	293
8	http://testphp.vulnweb.com	GET	/index.php			200	5180	HTML	php	Home of Acunetix Art		44.228.249.3			15:49:42 1 Ja...	8080	293

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.

Burp Suite Community Edition v2025.11.6 - Temporary Project

Dashboard Target Proxy Intruder Repeater View Help

Intercept **HTTP history** WebSockets history Match and replace Proxy settings

Filter settings: Hiding CSS and image content; hiding specific extensions Filter on

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start respon...
1	http://testphp.vulnweb.com	GET	/			200	5180	HTML	php	Home of Acunetix Art		44.228.249.3			15:49:25 1 Jan, 2000	291	
2	http://testphp.vulnweb.com	GET	/index.php			200	5180	HTML	php	Home of Acunetix Art		44.228.249.3			15:49:26 1 Jan, 2000	293	
3	http://testphp.vulnweb.com	GET	/categories.php			200	6337	HTML	php	picture categories		44.228.249.3			15:49:20 1 Jan, 2000	292	
4	http://testphp.vulnweb.com	GET	/artists.php			200	5550	HTML	php	artists		44.228.249.3			15:49:32 1 Jan, 2000	290	
5	http://testphp.vulnweb.com	GET	/disclaimer.php			200	5746	HTML	php	disclaimer		44.228.249.3			15:49:34 1 Jan, 2000	293	
6	http://testphp.vulnweb.com	GET	/cart.php			200	5125	HTML	php	you can't		44.228.249.3			15:49:35 1 Jan, 2000	290	
7	http://testphp.vulnweb.com	GET	/guestbook.php			200	5613	HTML	php	guestbook		44.228.249.3			15:49:37 1 Jan, 2000	293	
8	http://testphp.vulnweb.com	GET	/index.php			200	5180	HTML	php	Home of Acunetix Art		44.228.249.3			15:49:42 1 Jan, 2000	293	

Request

Pretty	Raw	Hex
1 GET /index.php 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,image/apng,*/*q=0.8,application/signed-exchange;v=b3;q=0.7		
7 Referer: http://testphp.vulnweb.com/		
8 Accept-Encoding: gzip, deflate, br		
9 Connection: keep-alive		
10		
11		

Response

Pretty	Raw	Hex	Render
1 HTTP/1.1 200 OK			
2 Server: nginx/1.19.0			
3 Date: Thu, 01 Jan 2026 10:19:28 GMT			
4 Content-Type: text/html; charset=UTF-8			
5 Connection: keep-alive			
6 X-Powered-By: PHP/8.4.40-38+ubuntu20.04.1+deb.sury.org+1			
7 Content-Length: 4938			
8			
9 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 4.01 Transitional//EN"			
10 <html><head>			
11 <!-- InstanceBegin template="/Templates/main_dynamic_template.dwt.php" -->			
12 <code>use strict;use warnings;use utf8;use Acunetix::Util::HTML::Locked;-->			
13 <head>			
14 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">			
15 <!-- InstanceBeginEditable name="document_title_rgn" -->			
16 <title>			
17 Home of Acunetix Art			
18 </title>			
19 <!-- InstanceEndEditable -->			
20 <link rel="stylesheet" href="style.css" type="text/css">			
21 <!-- InstanceBeginEditable name="headers_rgn" -->			
22 <!-- here goes headers headers -->			
23 <!-- InstanceEndEditable -->			

Inspector

Request attributes	2	v
1		
2		

Request headers

8	v
1	
2	

Response headers

6	v
1	
2	

Event log All issues 0 highlights Search 0 highlights Search 0 highlights Search Memory: 135.5MB of 11.06GB Disabled

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

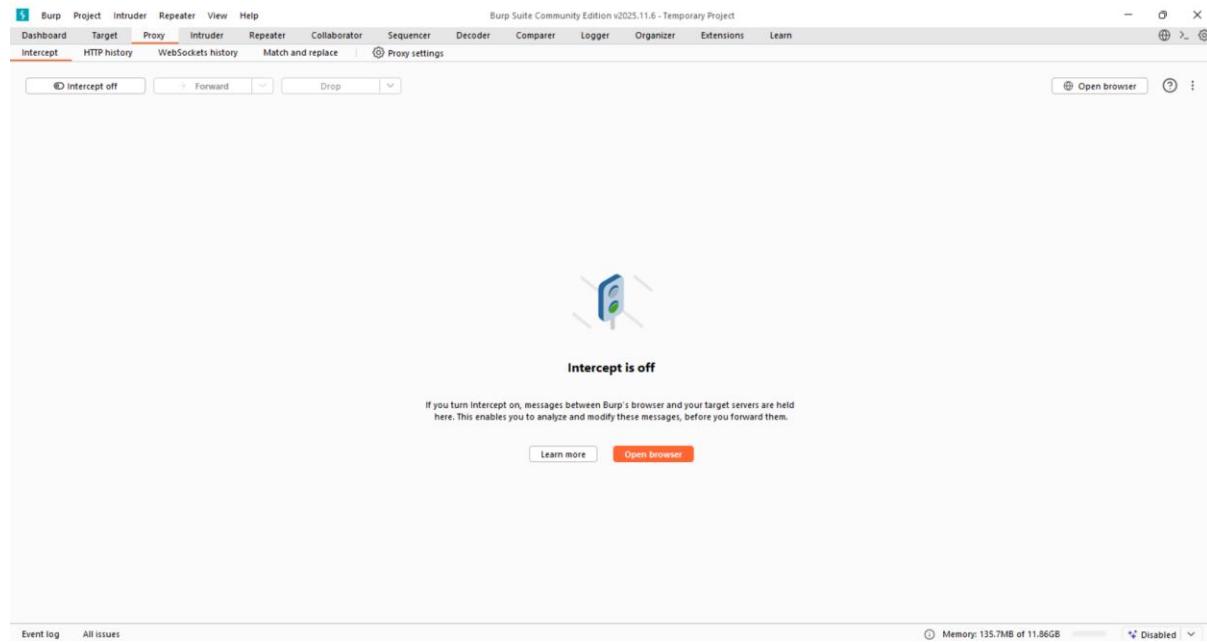
This creates hidden attack surfaces.

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”.

The site map displays information about the contents of your target applications, along with any issues that have been discovered. The URL view shows your targets as a tree of URLs, organized hierarchically by domain and directory. To populate the URL view, run a scan or browse using Burp's browser.

[Learn more](#) [Open browser](#)

Step3: Click on “Scope”.

Target scope

Use these settings to define exactly what hosts and URLs constitute the target for your current work. This configuration affects the behavior of tools throughout the suite.

Use advanced scope control

Include in scope

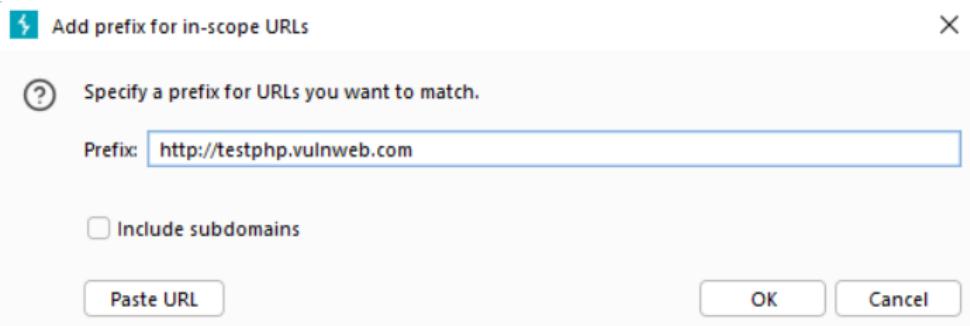
Add	Enabled	Prefix	Include subdomains
<input type="button" value="Add"/>	<input checked="" type="checkbox"/>		
<input type="button" value="Edit"/>			
<input type="button" value="Remove"/>			
<input type="button" value="Paste URL"/>			
<input type="button" value="Load ..."/>			

Exclude from scope

Add	Enabled	Prefix	Include subdomains
<input type="button" value="Add"/>	<input checked="" type="checkbox"/>		
<input type="button" value="Edit"/>			
<input type="button" value="Remove"/>			
<input type="button" value="Paste URL"/>			
<input type="button" value="Load ..."/>			

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

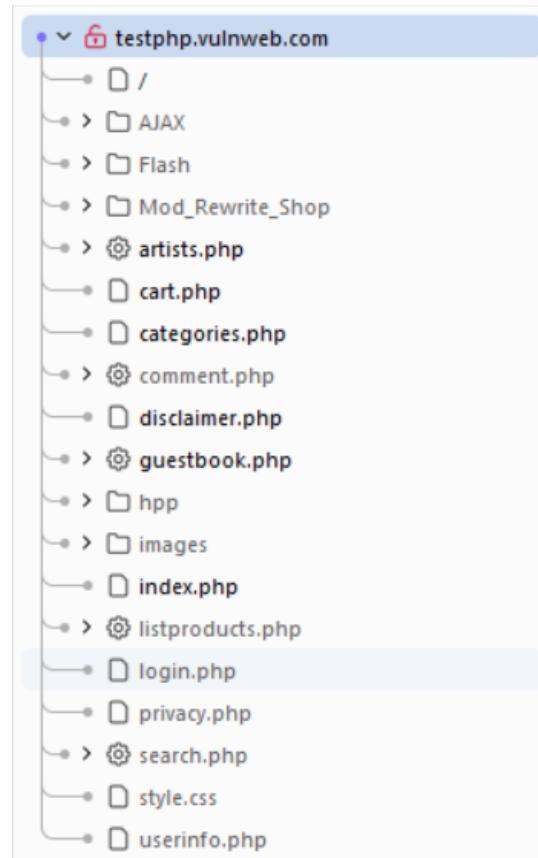
Step5: Write the URL (say <http://testphp.vulnweb.com>)



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.

Host	Method	URL	Params	Status code	Length	MIME type	Title	Notes	Time requested
http://testphp.vulnweb.com	GET	/		200	5180	HTML	Home of Acunetix Art		21:28:39 1 Jan 2026
http://testphp.vulnweb.com	GET	/artists.php		200	5550	HTML	artists		21:28:51 1 Jan 2026
http://testphp.vulnweb.com	GET	/cart.php		200	5125	HTML	you cart		21:28:52 1 Jan 2026
http://testphp.vulnweb.com	GET	/categories.php		200	6337	HTML	picture categories		21:28:50 1 Jan 2026
http://testphp.vulnweb.com	GET	/download.php		200	5746	HTML	distances		21:28:49 1 Jan 2026
http://testphp.vulnweb.com	GET	/guestbook.php		200	5612	HTML	guestbook		21:28:40 1 Jan 2026
http://testphp.vulnweb.com	GET	/index.php		200	5180	HTML	Home of Acunetix Art		21:28:42 1 Jan 2026
http://testphp.vulnweb.com	GET	/AJAX/index.php							
http://testphp.vulnweb.com	GET	/Flash/add.swf							
http://testphp.vulnweb.com	GET	/Mod_Rewrite_Shop/							
http://testphp.vulnweb.com	GET	/artists.php?Artist=1							
http://testphp.vulnweb.com	GET	/artists.php?Artist=2							
http://testphp.vulnweb.com	GET	/artists.php?Artist=3							
http://testphp.vulnweb.com	GET	/comment.php							

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

Host	Method	URL	Params	Status code	Length	MIME type	Title	Notes	Time requested
http://testphp.vulnweb.com	GET	/hpp/							
http://testphp.vulnweb.com	GET	/images/logo.gif							
http://testphp.vulnweb.com	GET	/images/remark.gif							
http://testphp.vulnweb.com	GET	/listproducts.php							
http://testphp.vulnweb.com	GET	/listproducts.php?cat=1		✓					
http://testphp.vulnweb.com	GET	/listproducts.php?cat=2		✓					
http://testphp.vulnweb.com	GET	/listproducts.php?cat=3		✓					
http://testphp.vulnweb.com	GET	/listproducts.php?cat=4		✓					
http://testphp.vulnweb.com	GET	/login.php							
http://testphp.vulnweb.com	GET	/privacy.php							
http://testphp.vulnweb.com	GET	/search.php							
http://testphp.vulnweb.com	POST	/search.php		✓					
http://testphp.vulnweb.com	GET	/style.css							
http://testphp.vulnweb.com	GET	/userinfo.php							

Request

```

GET /search.php HTTP/1.1
Host: testphp.vulnweb.com
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="143", "Not;A=Brand";v="24",
"Google Chrome";v="143"
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US;q=0.9,en;q=0.8
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-ex
change;v=1;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br

```

Response

```

HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 12345
Date: Mon, 12 Jun 2023 14:23:45 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
X-Powered-By: PHP/8.1.12-1+ubuntu22.04.1+deb.sury.org+1

<!DOCTYPE html>
<html>
<head>
<title>Search Results</title>
</head>
<body>
<h1>Search Results</h1>
<p>Your search results are displayed below.</p>
<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
<li>Item 4</li>
<li>Item 5</li>
</ul>
</body>

```

Inspector

- Request attributes: 2
- Request headers: 14

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

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. In the 'Request' pane, a GET request to 'http://testphp.vulnweb.com/search.php' is displayed with the following headers:

```

1 GET /search.php HTTP/1.1
2 Host: testphp.vulnweb.com
3 Cache-Control: max-age=0
4 Sec-Ch-Ua: "Chromium";v="143", "Not A Brand";v="24", "Google Chrome";v="143"
5 Sec-Ch-Ua-Mobile: 70
6 Sec-Ch-Ua-Platform: "Windows"
7 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
8 Sec-Fetch-Site: none
9 Sec-Fetch-Mode: navigate
10 Sec-Fetch-User: ?1
11 Sec-Fetch-Dest: document
12 Accept-Encoding: gzip, deflate, br
13 Connection: close
14
15
16
17

```

The 'Response' pane shows the server's response, which includes the modified User-Agent header in the 'Content-Type' header.

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

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. In the 'Request' pane, the same GET request to 'http://testphp.vulnweb.com/search.php' is displayed, but the 'User-Agent' header has been modified to 'User-Agent: OOB_USER_AGENT_123'.

The 'Response' pane shows the server's response, which includes the modified User-Agent header in the 'Content-Type' header.

- 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.