Overview

The SSRF Scanner is a multi-functional tool designed to detect Server-Side Request Forgery (SSRF) vulnerabilities. Key features include:

**Content Fetching**: Retrieves HTML content from URLs.

**HTML Parsing**: Extracts links, form actions, input values, and input names.

**SSRF Testing**: Injects payloads to test for vulnerabilities.

**Request Crafting**: Sends HTTP requests with custom headers and payloads (GET/POST).

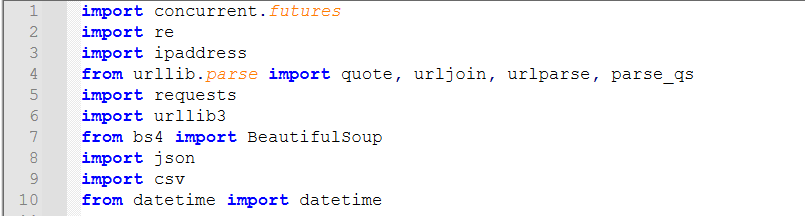
**OS Detection**: Identifies server OS via response headers (e.g., Server header).

**Collaboration Support**: Detects out-of-band interactions using domains like Burp Collaborator.

**Output Formats**: Generates results in JSON or CSV.

**Efficiency**: Supports multithreading, URL lists, and proxy configurations.

Imports & Setup



**concurrent.futures**  
Used for running tasks concurrently using threads or processes (helps with parallel execution).

**re**  
Provides support for **regular expressions**, allowing pattern matching and text searching.

**ipaddress**  
Allows manipulation and validation of **IPv4 and IPv6 addresses**.

**urllib.parse (quote, urljoin, urlparse, parse\_qs)**  
Tools for **manipulating and parsing URLs**, such as encoding, joining, and extracting components or query parameters.

**requests**  
A popular library for making **HTTP requests** (GET, POST, etc.) to communicate with web servers.

**urllib3**  
A powerful, low-level HTTP client, often used internally by libraries like requests.

**bs4 (BeautifulSoup)**  
Used for **parsing and extracting data from HTML or XML** documents (web scraping).

json, csv: Result formatting/output.

 **datetime**  
Provides classes for working with **dates and times**, useful for timestamps and scheduling.

Disable SSL Warnings



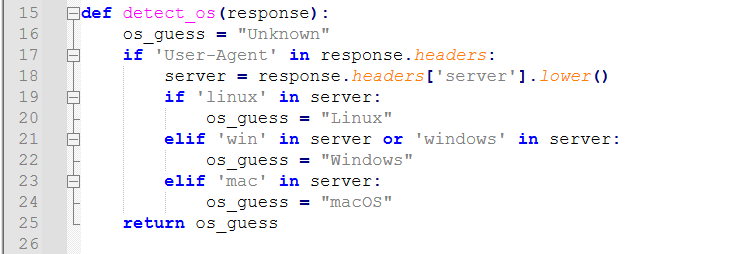
Suppresses SSL certificate warnings when making insecure HTTPS requests.

Global Variables



Stores scan results as a list of dictionaries containing details about successful SSRF attempts.

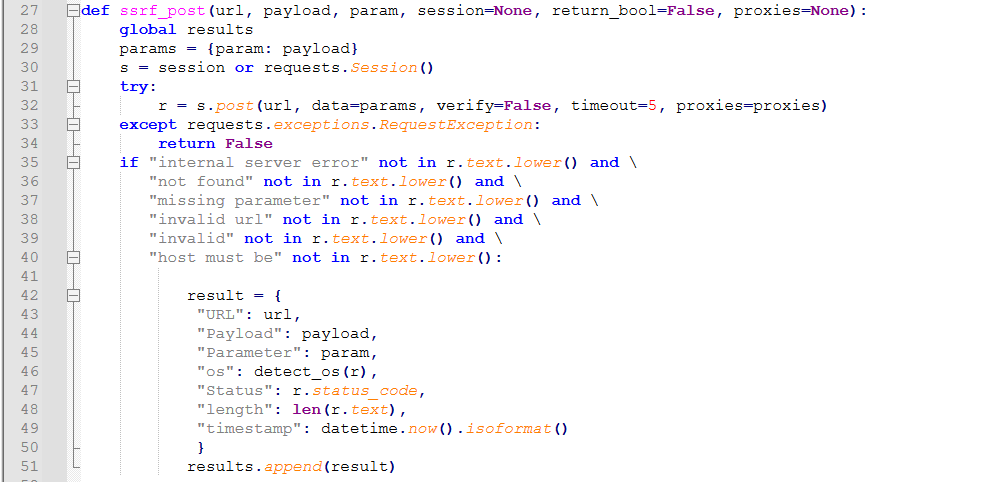
Detection Functions

1. detect\_os(response)  


Detects the operating system of the server by inspecting the Server header.

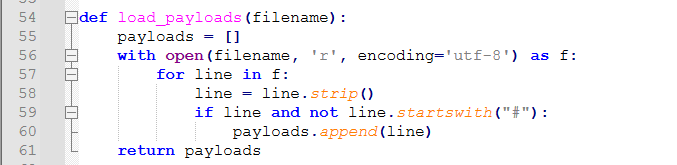
Attack Functions

1) ssrf\_post(url, payload, param, session=None, return\_bool=False, proxies=None)



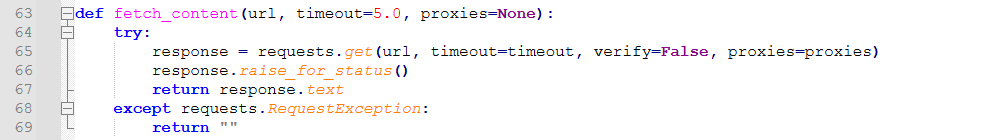
Performs an SSRF POST request with a given payload injected into a parameter.

2. load\_payloads(filename)



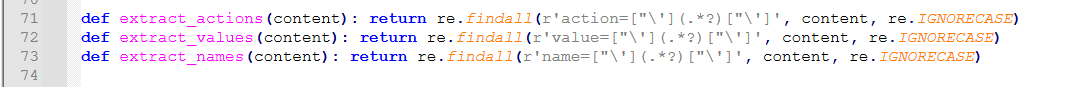
Loads payloads from a file (e.g., payload.txt) line by line.

3. fetch\_content(url, timeout=5.0, proxies=None)



Fetches page content using GET requests.

4. extract\_actions(content), extract\_values(content), extract\_names(content)



Use regex to extract:

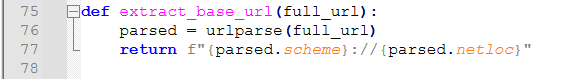
Form action attributes

Input value attributes

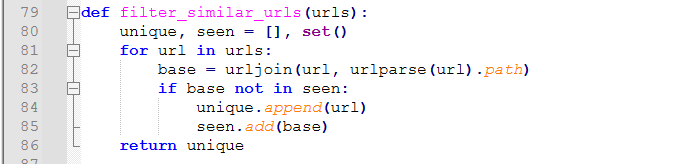
Input name attributes

Used to identify where to inject payloads.

5. extract\_base\_url(full\_url)

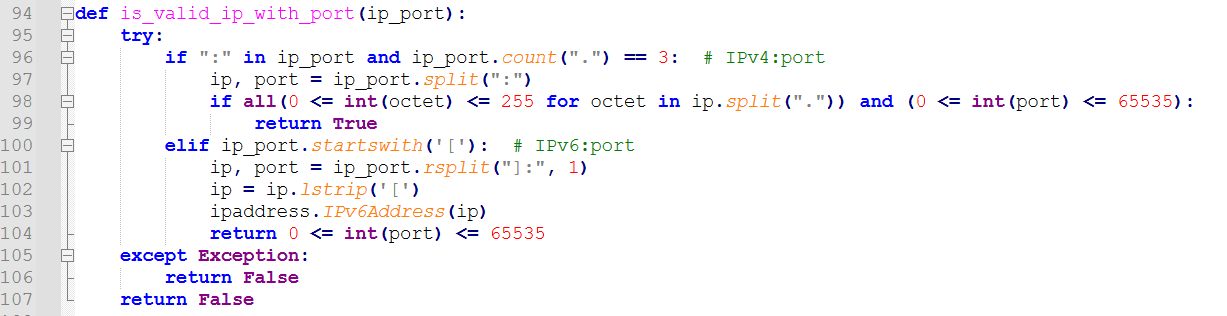


Returns the base URL (scheme + domain) of a full URL.  
6. filter\_similar\_urls(urls)

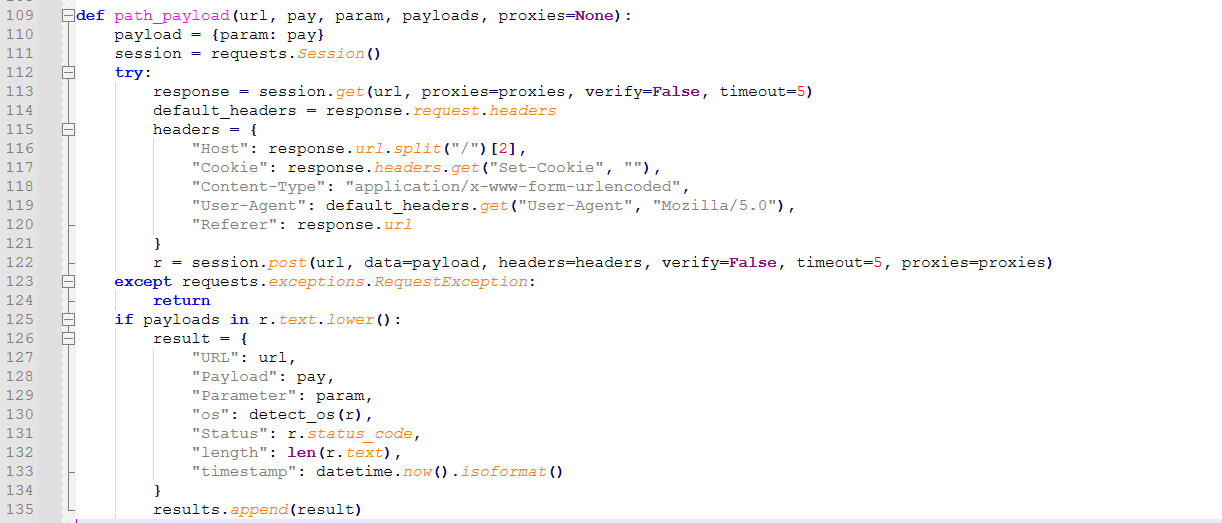


Filters duplicate URLs by comparing base paths.

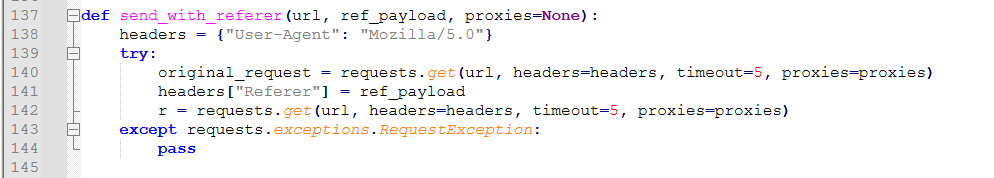
7. is\_valid\_ip\_with\_port(ip\_port)



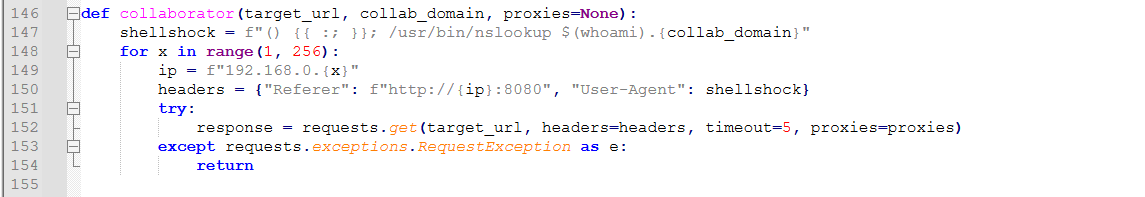
Validates IPv4 or IPv6 addresses with optional port numbers.  
8. path\_payload(...)



Injects payloads into the path of a URL and sends POST requests with custom headers.  
9. send\_with\_referer(...)

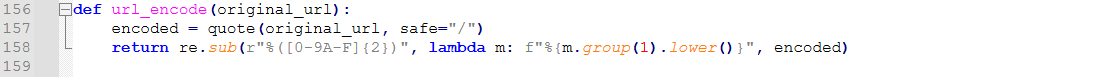


Sets a custom Referer header in a GET request.  
10. collaborator(target\_url, collab\_domain, proxies=None)



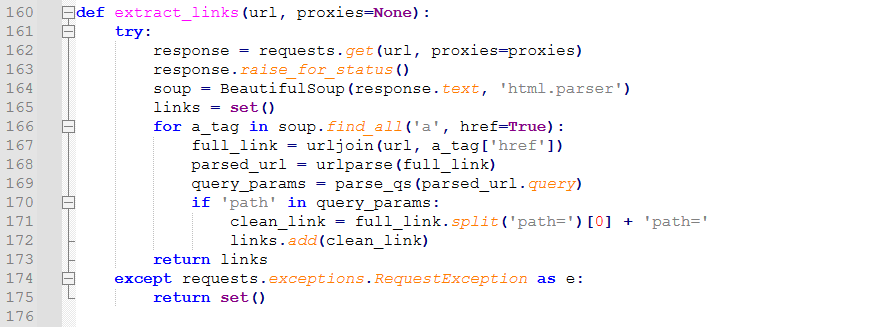
Attempts to trigger an external DNS lookup to a collaborator domain (e.g., Burp Collaborator).

11. url\_encode(original\_url)



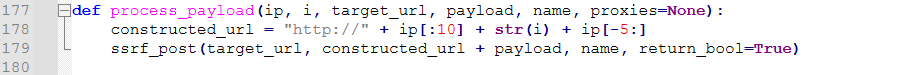
URL-encodes a string.

12. extract\_links(url, proxies=None)



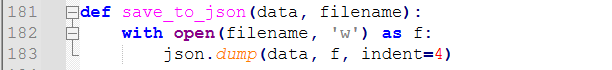
Uses BeautifulSoup to extract all <a href> links and looks for path= query parameters.

13.process\_payload(…)



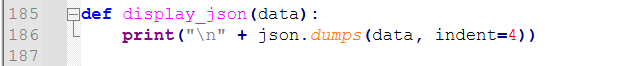
Takes an IP template and modifies the middle part with a number i to generate a new internal IP address and send it to ssrf\_post().(Bruteforce attack)

14. save\_to\_json(data, filename)



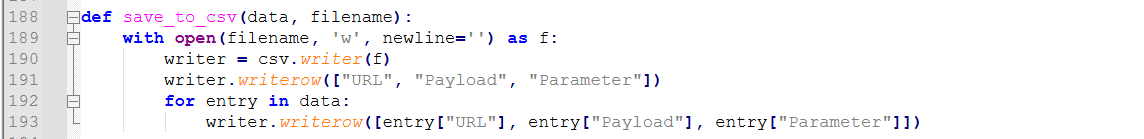
Saves results to a JSON file.

15. display\_json(data)



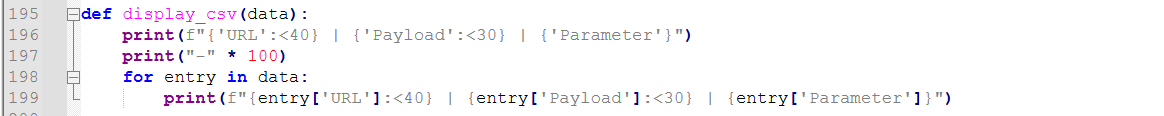
Prints JSON-formatted results to console.

16. save\_to\_csv(data, filename)



Saves results to a CSV file.

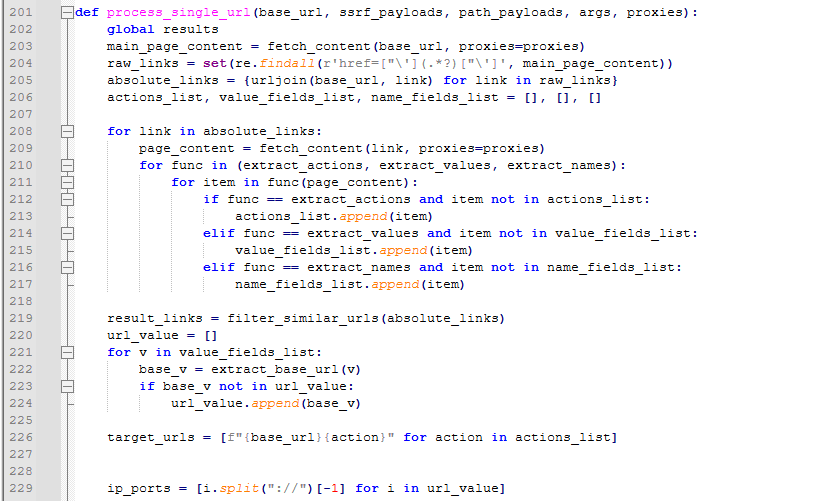
17. display\_csv(data)

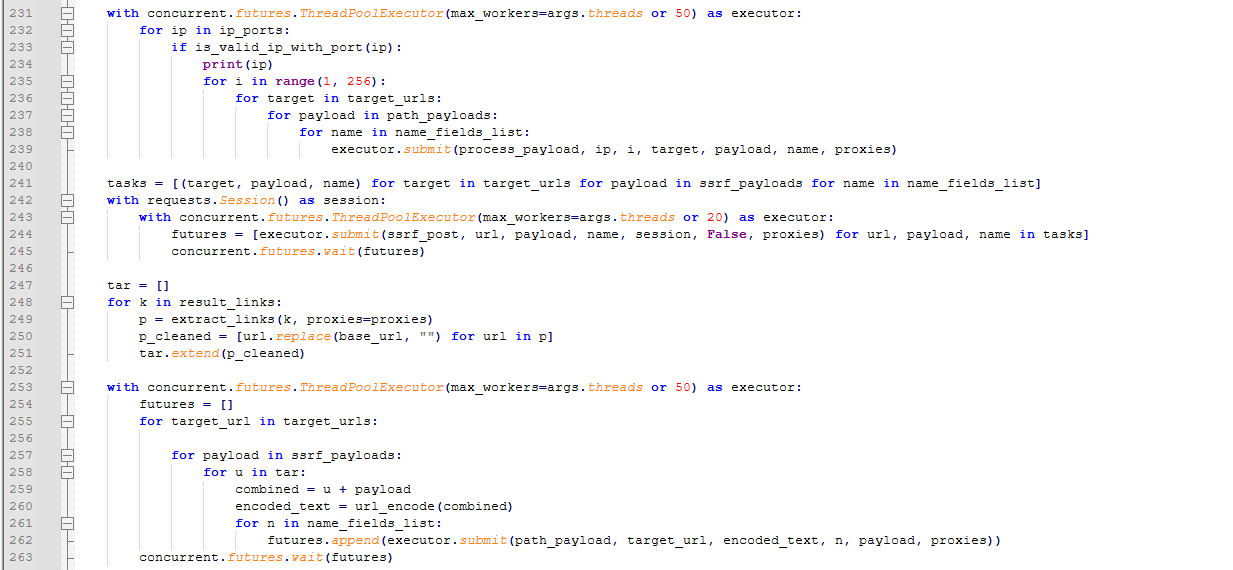


Prints CSV-style formatted results to console.

🚀 Main Scanner Logic

1. process\_single\_url(base\_url, ssrf\_payloads, path\_payloads, args, proxies)







This is the heart of one-target scanning:

Crawl main page to collect all links (href=) and normalize to absolute URLs.

For each link:

Fetch page content.

Extract all action=, value=, name= fields.

Deduplicate similar URLs and base URLs.

Build list of SSRF targets by combining base URL + each extracted form action.

Brute‑force path-based SSRF

For each discovered hostname:port from URL values, try varying trailing numbers 1–255.

Concurrently submit path\_payload to each form endpoint.

Standard SSRF blast

Using all (form‑action, ssrf\_payload, param\_name) combinations, POST concurrently.

Chained SSRF via “path=”

Extract additional “path=” links from all result pages, append each SSRF payload, URL‑encode, re‑POST.

Blind SSRF (Collaborator)

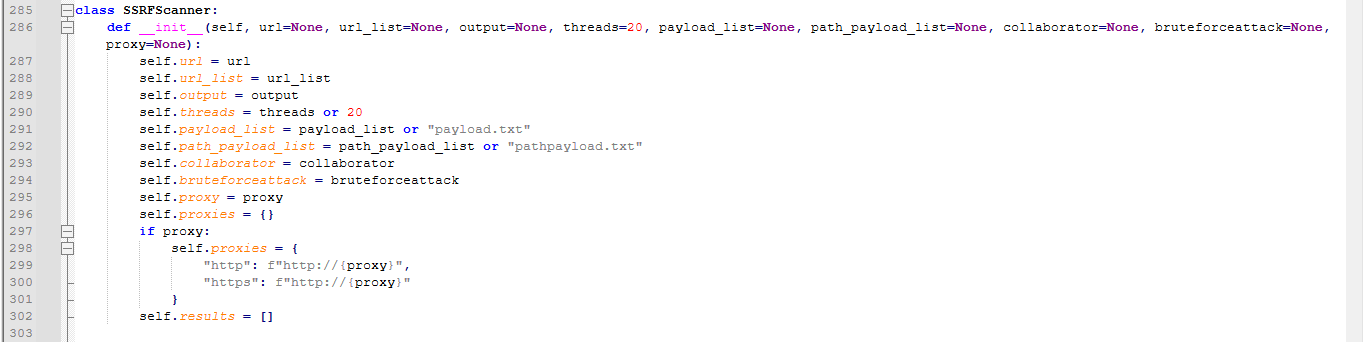
If configured, run send\_with\_referer and full collaborator scan.

Output

Print or save to JSON/CSV, depending on CLI flags.

2. class SSRFScanner:

A) def \_\_init\_\_(….)



Initializes the scanner with:

Target URL or list

Payload files

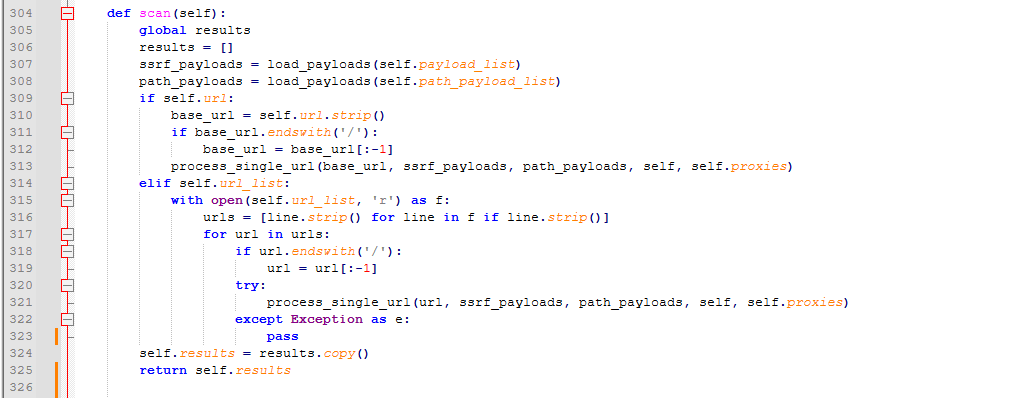
Collaborator domain

Proxy settings

Thread count

Output format

B)scan(self)



Runs the scanner against each URL.