SSRF (Server-Side Request Forgery) - In-Depth Explanation

1. What is SSRF?

SSRF is a web security vulnerability that allows an attacker to make the server send unauthorized requests to internal or external systems. The server acts as a proxy for the attacker, bypassing security controls like firewalls.

Real-World Analogy:

Imagine a bank teller (the server) who follows customer requests blindly. A criminal (attacker) tricks the teller into:

- Opening the bank vault (internal system)
- Accessing other customers' accounts (external systems)
- Retrieving confidential documents (sensitive data)

This is how SSRF works—the attacker abuses server trust to access restricted resources.

---

2. How SSRF Works

Step-by-Step Attack Flow:

1. Attacker sends a malicious request to a vulnerable web app:

http

https://example.com/fetch?url=http://internal-server/admin

2. Server processes the request and fetches the internal resource.
3. Server returns sensitive data (e.g., admin panel, database info, cloud metadata) to the attacker.

Types of SSRF:

Type	Description	Example
Basic SSRF	Directly retrieves data from internal systems	*http://localhost:8080*
Blind SSRF	No response is returned, but internal actions occur	Triggering internal API calls
Semi-Blind SSRF	Indirectly leaks data (time delays, errors)	Checking if a port is open via response time

3. Why is SSRF Dangerous?

Exploitable Scenarios:

- ✔ Accessing Internal Systems *(e.g., http://localhost, http://192.168.1.1)*
- ✔ Reading Sensitive Files *(e.g., file:///etc/passwd)*
- ✔ AWS/GCP/Azure Cloud Metadata Attacks *(e.g., http://169.254.169.254)*
- ✔ Bypassing Firewalls (since the request comes from a trusted server)
- ✔ Port Scanning (checking open ports on internal networks)

Real-World Impact:

- Data breaches (customer info, passwords, API keys)
- Cloud takeover (stealing IAM roles from metadata)
- Remote Code Execution (RCE) in some cases

4. How to Exploit SSRF (For Ethical Testing)

Common Payloads:

http

http://vulnerable-site.com/api?url=http://localhost/admin

http://vulnerable-site.com/export?url=file:///etc/passwd

http://vulnerable-site.com/fetch?url=http://169.254.169.254/latest/meta-data

Advanced Techniques:

- DNS Rebinding (Bypassing IP restrictions)
- Using URL Shorteners (Obfuscating malicious URLs)
- CRLF Injection (Adding malicious headers)

---

5. How to Prevent SSRF

Defense Mechanisms:

Method	Implementation
Input Validation	Allow only whitelisted domains *(example.com)*
Block Private IPs	Deny *127.0.0.1, 192.168.x.x, 10.x.x.x*
Restrict URL Schemes	Allow only *HTTP/HTTPS*, block *file://*, *gopher://*, etc.
Use a Proxy	Route external requests through a controlled proxy
Network Segmentation	Isolate internal services from web servers

Code Example (Python):

```
from urllib.parse import urlparse

def safe_fetch(url):
    parsed = urlparse(url)

    # Allow only HTTP/HTTPS
    if parsed.scheme not in ("http", "https"):
        raise ValueError("Invalid URL scheme")

    # Block internal IPs
    blocked_nets = ["127.0.0.0/8", "10.0.0.0/8", "192.168.0.0/16"]
    for net in blocked_nets:
        if ipaddress.ip_address(parsed.hostname) in ipaddress.ip_network(net):
            raise ValueError("Internal IP blocked")

    # Fetch only from allowed domains
    allowed_domains = ["api.trusted.com", "cdn.safe.org"]
    if parsed.hostname not in allowed_domains:
        raise ValueError("Domain not allowed")

    return requests.get(url).text
```

6. Testing for SSRF

Steps to Check for SSRF:

1. Find URL Parameters *(e.g., ?url=, ?endpoint=, ?api=)*
2. Try Internal IPs *(127.0.0.1, 192.168.1.1)*
3. Test Cloud Metadata *(169.254.169.254 for AWS)*
4. Check for File Access *(file:///etc/passwd)*
5. Monitor Responses (Errors, delays, data leaks)

Tools for SSRF Testing:

- Burp Suite (Manual testing)
- SSRFmap (Automated exploitation)
- ffuf (Fuzzing for SSRF endpoints)

---

7. Conclusion

- SSRF allows attackers to abuse server trust to access internal systems.
- Critical impact: Data leaks, cloud breaches, RCE.
- Prevention: Whitelist domains, block private IPs, restrict URL schemes.
- Testing: Always check URL parameters in web apps.