Perfect 👌 Let’s now go into **Server-Side Request Forgery (SSRF)** testing in the same detailed WSTG style.

**🌐 Testing for Server-Side Request Forgery (SSRF)**

**🔹 1. What is SSRF?**

* **SSRF** occurs when an application fetches a remote resource (e.g., image, PDF, API request) **based on user input**, without proper validation.
* An attacker can manipulate the request to:
  + Access **internal services** (http://localhost:8080/admin).
  + Query **cloud metadata endpoints** (AWS, Azure, GCP).
  + Perform **port scanning** on internal systems.
  + Bypass firewalls and pivot deeper into the network.

**🔹 2. Example of Vulnerable Code**

**PHP (Image Fetcher):**

<?php

$url = $\_GET['url'];

$image = file\_get\_contents($url);

echo $image;

?>

Attacker payload:

http://target.com/fetch?url=http://localhost:8080/admin

→ Fetches **internal admin panel**, exposing sensitive data.

**🔹 3. Types of SSRF**

1. **Basic SSRF**
   * Fetch internal resources via manipulated URLs.
2. **Blind SSRF**
   * No response to attacker directly, but request is still made (use DNS logs or Burp Collaborator to detect).
3. **Advanced SSRF (Cloud-specific)**
   * AWS: http://169.254.169.254/latest/meta-data/
   * GCP: http://metadata.google.internal/computeMetadata/v1/
   * Azure: http://169.254.169.254/metadata/instance

**🔹 4. Manual Testing Steps**

1. **Identify input fields** that fetch resources (URL parameters, file uploads, webhooks, XML parsers).
   * Example params: url=, fetch=, redirect=, next=, src=, file=, domain=.
2. **Try controlled external URLs**:
3. http://burp-collaborator-url.com

If you see an interaction → SSRF exists.

1. **Test for internal access**:
2. http://127.0.0.1:80/
3. http://localhost:8080/
4. http://169.254.169.254/latest/meta-data/
5. **Port Scanning via SSRF**
6. http://127.0.0.1:22
7. http://127.0.0.1:3306
8. http://127.0.0.1:8080
9. **Blind SSRF Detection**
   * Use **Burp Collaborator**, **interactsh**, or **requestbin**.

**🔹 5. Testing with Tools**

**🔸 Burp Suite**

* Intercept requests with URL parameters.
* Replace with SSRF payloads (internal IPs, Collaborator URL).
* Use **Burp Collaborator Client** to detect out-of-band requests.

**🔸 OWASP ZAP**

* Scan for URL-based parameters automatically.

**🔸 Specialized Tools**

* **Gopherus** → Generate gopher:// payloads for RCE.
* **SSRFmap** → Automates SSRF exploitation.
* **Interactsh** → Catch blind SSRF.

**🔹 6. Impact of Successful SSRF**

* **Internal Network Reconnaissance**
* **Bypass Firewalls** → Reach services not exposed publicly.
* **Sensitive Data Exposure** (cloud instance metadata, DB credentials).
* **Pivoting** to Remote Code Execution (via gopher, file, dict protocols).

**🔹 7. Mitigation Strategies**

✅ **Defensive Controls**

1. Strictly **validate and allowlist URLs**.
   * Only allow https://trusted.com/resource/\*.
2. Disallow dangerous schemas: file://, gopher://, dict://, ftp://.
3. Use **network-level egress filtering** to block requests to:
   * 127.0.0.1
   * 169.254.169.254
   * Private IP ranges (10.0.0.0/8, 192.168.0.0/16).
4. Disable or tightly control server-side URL fetching if not required.
5. For cloud: **block metadata service access** (IMDSv2 for AWS).

**🔹 8. Quick SSRF Payloads Reference**

| **Purpose** | **Payload** |
| --- | --- |
| Test external | http://burp-collaborator-url.com |
| Localhost | http://127.0.0.1:80/ |
| Cloud AWS | http://169.254.169.254/latest/meta-data/ |
| Cloud GCP | http://metadata.google.internal/computeMetadata/v1/ |
| File read | file:///etc/passwd |
| Redis exploit | gopher://127.0.0.1:6379/\_FLUSHALL |

✅ That’s the **complete methodology for SSRF testing** (manual + Burp + cloud exploitation + mitigations).

Do you want me to also prepare a **Burp Intruder payload list for SSRF detection** (common IPs, schemas, cloud endpoints) so you can directly plug it in during testing?