Server-Side Request Forgery (SSRF)

Server-Side Request Forgery (SSRF) is a security vulnerability that occurs when an attacker can trick a server into making unintended requests to other internal or external services. This can lead to unauthorised access to internal resources, sensitive data leakage, or additional attacks.

Key Characteristics of SSRF:

1. **Exploitation of Server-Side Requests**:
    - The attacker manipulates server-side requests to target internal systems that are not directly accessible from the external network.
2. **Common Targets**:
    - Internal services (e.g., databases, other web applications, internal APIs)
    - Metadata services in cloud environments (e.g., AWS EC2 metadata)
    - Localhost services (127.0.0.1)
3. **Attack Vectors**:
    - URL parameters
    - HTTP headers
    - Form fields
    - Any input field that the server uses to generate a request

Example of SSRF:

Consider a web application that fetches data from a URL provided by the user. An attacker might supply a malicious URL:

http
Copy code
```
http://vulnerable-app.com/fetch?url=http://internal-service.local/admin
```

If the application blindly trusts and fetches this URL, it could allow the attacker to access the internal admin interface.

Potential Impacts of SSRF:

- **Information Disclosure**: Access to sensitive data stored on internal systems.
- **Network Scanning**: Discovery of internal network structure and services.
- **Remote Code Execution**: If internal services are vulnerable, SSRF can be used to exploit them.
- **Denial of Service**: Overwhelming internal services with requests.

Mitigation Strategies:

1. **Input Validation and Sanitization**:
    - Validate and sanitize user input to ensure it adheres to expected patterns and formats.
2. **Allowlist of URLs**:
    - Implement a strict allowlist of acceptable URLs or IP addresses that the application can access.
3. **Deny Access to Internal Networks**:
    - Restrict the server from making requests to internal IP ranges (e.g., 127.0.0.1, 10.0.0.0/8).
4. **Authentication and Authorization**:
    - Ensure that any internal services accessed via SSRF require proper authentication and authorization.
5. **Network Segmentation**:
    - Separate internal services from the web-facing application to limit exposure.
6. **Monitoring and Logging**:
    - Implement robust logging and monitoring to detect and respond to unusual request patterns.
7. **Use Security Libraries and Frameworks**:
    - Leverage security libraries and frameworks that provide protection against common vulnerabilities, including SSRF.

By understanding and mitigating SSRF vulnerabilities, developers can better protect their applications and internal services from malicious attacks.