Server-Side Request Forgery (SSRF)

Introduction

Server-Side Request Forgery (SSRF) is a critical web security vulnerability that allows attackers to induce server-side applications to make requests to unintended locations. This comprehensive guide explores SSRF across beginner, intermediate, and advanced levels, covering attack vectors, defense mechanisms, and real-world implications. SSRF vulnerabilities have gained significant prominence in recent years, particularly as organizations adopt cloud-based and microservice architectures, making this attack vector increasingly relevant and dangerous.

Beginner Level

Understanding SSRF Fundamentals

What is SSRF?

Definition: Server-Side Request Forgery (SSRF) is a web security vulnerability that allows attackers to induce a server-side application to make HTTP requests to an arbitrary domain chosen by the attacker. The attacker can force the server to connect to internal-only services within the organization's infrastructure or to external systems, potentially leading to unauthorized actions or access to data.

Core Concepts:

- **The vulnerable application makes HTTP requests based on user-supplied input**
- **The attacker manipulates this input to control the destination of these requests**
- **The server acts as a proxy, allowing attackers to reach otherwise inaccessible systems**
- **SSRF bypasses network security controls by leveraging the server's trusted position**

Attack Flow:

1. **Attacker identifies a vulnerable parameter that influences server-side requests**
2. **Attacker modifies this parameter to point to a target system (internal or external)**
3. **The server makes the request to the attacker-specified destination**
4. **The server may return the response data to the attacker or perform unintended actions**

Types of SSRF

Basic SSRF (Direct Response):

- **Server makes the request and returns the response directly to the attacker**
- **Allows direct reading of content from internal systems**
- **Provides immediate feedback about the success of the attack**
- **Example: A website preview feature that returns the content of any URL provided**

Blind SSRF:

- **Server makes the request but doesn't return the response to the attacker**

- **Attacker must infer success based on timing, error messages, or out-of-band techniques**
- **More difficult to exploit but often harder to detect**
- **Example: A webhook feature that sends notifications to user-specified URLs**

Semi-Blind SSRF:

- **Server provides partial information about the responses**
- **May include status codes, response size, or processing time**
- **Provides more context than blind SSRF but less than direct SSRF**
- **Example: An import feature that reports success/failure but not the content**

Common SSRF Vectors

URL Input Fields:

- **Website preview generators**
- **Document/image importers**
- **URL shorteners**
- **API integrations**
- **Webhooks**
- **PDF generators**
- **Screenshot services**

File Inclusion:

- **XML External Entity (XXE) processing**
- **Remote file inclusion parameters**
- **URL-based configurations**
- **Remote template inclusion**

HTTP Headers:

- **Referer header**
- **X-Forwarded-For**
- **Custom headers processed by the application**

Data Formats with External Entity Support:

- **XML with DTD support**
- **SVG images**
- **HTML imports**
- **JSON with URL references**

Impact of SSRF Attacks

Internal Network Reconnaissance:

- **Mapping internal network architecture**
- **Port scanning internal hosts**
- **Identifying internal services and their versions**
- **Discovering hidden assets and shadow IT systems**

Data Exfiltration:

- **Accessing internal databases**
- **Reading local files via file:// protocol**
- **Retrieving metadata from cloud environments**
- **Accessing internal APIs and services**

Remote Code Execution Chains:

- **Leveraging SSRF to exploit vulnerable internal services**
- **Accessing administrative interfaces not intended for external access**
- **Exploiting unpatched internal systems**
- **Chain with other vulnerabilities for greater impact**

Cloud-Specific Risks:

- **Accessing instance metadata services**
- **Stealing cloud credentials and access tokens**
- **Lateral movement between cloud resources**
- **Privilege escalation within cloud environments**

Beginner Detection Examples

Common URLs and IP Addresses to Monitor:

- **Private IP ranges (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)**
- **Localhost references (127.0.0.1, localhost, 0.0.0.0)**
- **Link-local addresses (169.254.0.0/16)**
- **Cloud metadata endpoints (169.254.169.254, fd00:ec2::254)**
- **DNS rebinding attempts (attacker-controlled domains)**
- **Non-HTTP protocols (file://, dict://, gopher://, ftp://)**

Simple Detection Patterns:

- **URLs in unexpected parameters**

- **IP addresses in domain fields**
- **Protocol switching (http:// to file://)**
- **Unusual ports in URL parameters**
- **Localhost references in various formats**

Basic SSRF Prevention for Developers

Input Validation:

- **Validate URL scheme (restrict to http:// and https://)**
- **Validate domain against whitelist**
- **Reject private IP addresses and localhost**
- **Reject reserved IP ranges**
- **Validate port numbers (restrict to standard web ports 80/443)**

Request Control:

- **Use request timeouts to limit SSRF effectiveness**
- **Limit redirects to prevent request chains**
- **Strip authentication headers from forwarded requests**
- **Validate SSL certificates for outbound connections**

Architectural Controls:

- **Implement a URL whitelist instead of a blacklist**
- **Use dedicated service accounts with minimal privileges**
- **Segregate services that need to make external requests**
- **Deploy web applications with no route to internal networks**

Response Handling:

- **Limit response data returned to users**
- **Sanitize error messages to prevent information leakage**
- **Implement response size limits**
- **Consider returning only status codes for certain operations**

Beginner Implementation Examples

Vulnerable Code Example (PHP)

php

// Vulnerable SSRF example

function fetchRemoteContent() {

```
    $url = $_GET['url']; // User-controlled input

    // No validation performed
    $content = file_get_contents($url);

    echo $content; // Returns response directly to user
}
```

Basic Remediation (PHP)

```
php
// Basic remediation with URL validation
function fetchRemoteContent() {
    $url = $_GET['url'];

    // Validate URL format
    if (!filter_var($url, FILTER_VALIDATE_URL)) {
        die("Invalid URL format");
    }

    // Parse the URL
    $parsedUrl = parse_url($url);

    // Ensure only HTTP/HTTPS schemes
    if (!isset($parsedUrl['scheme']) ||
        !in_array($parsedUrl['scheme'], ['http', 'https'])) {
        die("Only HTTP and HTTPS URLs are allowed");
    }

    // Validate against whitelisted domains
    $allowedHosts = ['example.com', 'api.example.org'];
    if (!in_array($parsedUrl['host'], $allowedHosts)) {
        die("This host is not allowed");
```

```
    }

    // Make the request
    $content = file_get_contents($url);

    echo $content;
}
```

Java Vulnerable Code Example

```
java
@GetMapping("/fetch")
public String fetchUrl(@RequestParam String url) {
    try {
        URL resourceUrl = new URL(url);
        HttpURLConnection connection = (HttpURLConnection) resourceUrl.openConnection();
        connection.setRequestMethod("GET");

        BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
        String inputLine;
        StringBuffer content = new StringBuffer();

        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();
        connection.disconnect();

        return content.toString(); // Returns content directly to user
    } catch (Exception e) {
        return "Error: " + e.getMessage();
    }
```

}

Testing for SSRF (Basic)

Manual Testing Steps:

1. **Identify parameters that accept URLs or file paths**
2. **Test with internal IP addresses (127.0.0.1, 192.168.1.1)**
3. **Test with localhost in different formats (localhost, 127.0.0.1, [::1])**
4. **Try accessing common internal services (localhost:3306 for MySQL)**
5. **Test for DNS rebinding using tools like rbndr.us**
6. **Check for time-based responses to detect blind SSRF**

Common Test Payloads:

- **http://127.0.0.1/**
- **http://localhost:8080/admin**
- **http://internal-service/api**
- **file:///etc/passwd**
- **http://169.254.169.254/latest/meta-data/ (AWS)**
- **http://[::1]:22/**
- **http://127.1/**
- **http://0.0.0.0/**
- **http://425.510.425.510/ (Dotted decimal IP variant)**

http://{subdomain}.attacker.com/ (DNS rebinding)