# Research Report: About SSRF Outcome:

# What is SSRF?

In an SSRF attack, the attacker typically manipulates input that is used to specify the target URL for a server-side HTTP request. This can result when an application does not validate or sanitize a URL input by a user before pulling data from a remote resource. The attacker can make the targeted server perform requests to arbitrary destinations, potentially leading to various security risks.

These attacks can also result if the targeted resource has trust relationships with other systems, such as a cloud metadata service or backend APIs, allowing an attacker to make requests to those trusted services and extract sensitive information or perform unauthorized actions.

As modern web applications provide end-users with convenient features, fetching a URL has become a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF increases as APIs and cloud services have become more prevalent and computing architectures more de-centralized and complex.

SSRF is one of the OWASP Top 10 Application Security Risks, a widely recognized compilation of the most critical web application security risks. SSRF is also one of the OWASP Top Ten security risks that are common to both apps and API**s**, and bears special consideration when implementing security solutions.

Here's how an SSRF attack works:

- The application allows user input to determine the target URL or resource for a server-side request. This input might come from parameters in a URL, user inputs from form fields, or other data sources.
- The attacker submits a specially crafted request, manipulating the input to point to a resource the attacker wants to access or exploit. This resource could be an internal server, a backend service, an API, or other internal systems that are behind firewalls and are not accessible from the external network.
- The server processes the user's malicious input and constructs a request to the specified URL. This request is made from the server's perspective, not the user's browser, making it a server-side request.
- If the targeted server is within an internal network, the attacker may attempt to access and retrieve sensitive information from internal resources and then exfiltrate that data to an external location controlled by the attacker. SSRF can also be used to scan ports on internal systems, helping the attacker identify potential weaknesses and vulnerabilities.
- SSRF attacks are particularly concerning in cloud environments where instances might have access to sensitive metadata services.
- The impact of an SSRF attack can be severe, ranging from unauthorized access to sensitive data and services to the exploitation of internal systems and even compromise of entire cloud environments.

**Example of an SSRF Attack:**

Here’s how an SSRF attack can play out in real life.

Imagine an unsecured web application that allows users to upload images for processing, with a feature that lets users provide a URL to an image they want to process. However, instead of providing a legitimate image URL, the attacker submits a carefully crafted input containing a malicious URL that points to an internal resource on the application server, such as an internal file or a backend database. The application server blindly processes the malicious URL, and following the attacker's input, makes a request to the internal resource specified by the URL to fetch sensitive data or query a database and return the data to the attacker.

Once the server is compromised, the attacker can also make requests to external systems, probe for vulnerabilities, or interact with services that the server has access to. The impact of an SSRF attack can vary based on the system's architecture and the permissions of the compromised server. Attacks can lead to unauthorized data access, service disruption, or compromise of internal systems.

**Three Basic Types of SSRF Attacks:**

SSRF attacks can be classified into different types based on how the attacker interacts with the server and extracts information.

1. **Standard SSRF Attack**

In a standard SSRF attack, the attacker injects a malicious URL as part of the user input, triggering the server to make a request to the specified resource. The attacker can directly observe the response from the server and gather information about the internal network, such as how to retrieve data or identify accessible services.

2. **Blind SSRF Attack**

In blind SSRF attacks, the attacker does not directly receive the response from the server. Instead, the attacker indirectly confirms the success or failure of the SSRF attack by observing changes in the application's behavior.

The attacker submits a crafted input, forcing the server to make a request to an external or internal resource. The attacker looks for observable changes in the application's activity, such as differences in error messages, response times, or other side effects. This information helps the attacker infer whether the SSRF was successful, even though the attacker doesn't directly see the response.

3. **Time-Based Blind SSRF Attack**

Time-based blind SSRF attacks involve exploiting delays in the server's responses to infer the success or failure of the SSRF without directly seeing the response.

As in other SSRF attacks, the attacker submits a malicious input, causing the server to make a request to an external or internal resource. The attacker observes the time it takes for the application to respond. Delays in response time may indicate that the SSRF was successful.

The attacker iteratively adjusts the payload and monitors the time delays to extract information about the internal network or resources.

**How to Protect Against SSRF Attacks**

To protect against SSRF attacks, implement a combination of preventative cybersecurity measures including:

- **Input validation.** Validate and sanitize user-supplied input rigorously, especially when the application is making requests to external resources. Implement URL allowlisting, allowing only trusted domains or specific URLs that the application needs to access. Ensure that only the expected protocols, such as http and https, are allowed and avoid allowing potentially risky protocols such as file:// or ftp://, which are often exploited in SSRF attacks. Also, validate that URLs provided by users adhere to expected patterns or formats, reducing the likelihood of injecting malicious URLs.
- **Allowlists for hosts and IP addresses.** Maintain an allowlist of allowed hosts and IP addresses that the application is permitted to interact with. This can help restrict requests to trusted and intended destinations, reducing the attack surface for SSRF.
- **Restrict access to internal resources.** Implement network segmentation to limit the exposure of internal resources. Ensure that external-facing servers have minimal access to internal systems and services. Configure web application firewalls (WAFs) and access controls to restrict outbound connections from the application server. Only allow necessary and predefined connections and block unnecessary outbound traffic. Deploy a reverse proxy to act as an intermediary between the application and external resources. Configure the proxy to control which external resources the application can access, adding an additional layer of control.

How do these preventative measures help prevent SSRF attacks? Let’s look at a hypothetical scenario.

An online content management system allows users to input URLs to embed external images into their posts. This system processes user-supplied URLs without proper input validation. An attacker, aware of the lack of input validation, attempts to exploit the system by providing a malicious URL pointing to an internal resource, such as an internal API endpoint or a database server. However, the application’s security team enhances the system's input validation to enforce strict rules on accepted URLs. The system begins validating that the URLs provided by users adhere to expected patterns, and the team implements a whitelist of allowed domains for external resources.

The attacker submits the post with a crafted URL intended to exploit the SSRF flaw, but the input validation now detects the malicious URL. The input validation rejects the malicious URL during processing, preventing the application from making a request to the internal resource specified by the attacker. The SSRF attack is thwarted by the enhanced input validation: The system does not allow unauthorized requests to internal resources, and the integrity and security of internal systems are preserved.

By implementing input validation, the application can reject or sanitize malicious input, preventing unauthorized requests and mitigating the risk of SSRF attacks.

# Impacts of SSRF:

The impacts of **Server-Side Request Forgery (SSRF)** vulnerabilities can be significant, especially in environments where the server has access to internal systems or sensitive resources. Below are the key impacts categorized by their severity and the potential risks they pose:

### 1. Information Disclosure

SSRF can allow attackers to access internal or restricted information that would otherwise be inaccessible from outside the server's network.

- **Access to Internal Services**: Attackers can query internal APIs or databases, gaining information such as server configurations, user data, or operational details.
- **Cloud Metadata Access**: Many cloud providers (e.g., AWS, Google Cloud, Azure) use metadata endpoints to provide instance-related information. SSRF can expose sensitive details like access tokens or credentials.

### 2. Unauthorized Internal Network Access

SSRF allows attackers to bypass network access controls and interact with internal services that are not exposed to the public.

- **Network Scanning**: Attackers can map internal networks by making requests to different IPs and ports, identifying active services.
- **Interacting with Private Systems**: Attackers can interact with services like Redis, Elasticsearch, or other internal APIs that lack proper authentication, potentially leading to further exploits.

### 3. Exploitation of Internal Services

If internal services are vulnerable, SSRF can lead to further exploitation.

- **Remote Code Execution (RCE)**: Attackers might exploit vulnerable internal systems by sending malicious payloads via SSRF.
- **Database Manipulation**: If SSRF accesses services like GraphQL or SQL-based APIs, it could lead to injection attacks or unauthorized data access.

### 4. Privilege Escalation

In environments where servers have elevated permissions (e.g., accessing privileged networks or systems), SSRF can lead to privilege escalation.

- **Accessing Admin Panels**: If internal admin panels or management systems are accessible via SSRF, attackers can perform administrative actions.
- **Gaining Credentials**: Access to metadata services might expose temporary tokens or credentials that can be used to escalate privileges in cloud environments.

### 5. Data Exfiltration

SSRF can be exploited to send sensitive data from the server to an external attacker-controlled endpoint.

- **DNS-Based Data Leakage**: Attackers can encode sensitive data into DNS queries sent to their servers.
- **API Responses**: SSRF can be used to retrieve and forward responses from internal APIs to external systems.

**6. Denial of Service (DoS)**

Excessive or malformed SSRF requests can overwhelm internal services, leading to Denial of Service (DoS).

- **Resource Exhaustion**: Attackers can flood internal services with repeated requests.
- **Service Disruption**: If SSRF targets services critical to application functionality, it could render the application unusable.

**7. Financial and Reputational Damage**

SSRF exploits can have wide-ranging business impacts, including:

- **Financial Loss**: Unauthorized access to sensitive information can lead to financial penalties (e.g., fines under GDPR, CCPA).
- **Reputation Damage**: A data breach due to SSRF can tarnish the organization's reputation and erode customer trust.
- **Infrastructure Costs**: Malicious SSRF traffic could result in increased costs, especially in cloud environments where outgoing requests are metered.

**8. Blind SSRF Impacts**

In blind SSRF scenarios (where the attacker doesn't directly see the response), the attacker can still achieve significant impacts by leveraging out-of-band (OOB) interactions:

- **Trigger External Webhooks**: Exploiting SSRF to trigger sensitive or unauthorized workflows in other systems.
- **Covert Communications**: Using the server as a proxy for malicious activities.

**9. Chained Exploits**

SSRF is often used as an entry point to launch more sophisticated attacks:

- **Combined with XSS or CSRF**: SSRF can be paired with Cross-Site Scripting (XSS) or Cross-Site Request Forgery (CSRF) to exploit both client-side and server-side vulnerabilities.
- **Pivoting Attacks**: Attackers can use SSRF to move laterally within a network, attacking other services.

# Mitigation of SSRF Vulnerabilities:

Server-Side Request Forgery (SSRF) vulnerabilities can lead to severe security risks, including unauthorized access to internal systems, sensitive data exposure, and privilege escalation. To address these risks, organizations must adopt a combination of secure coding practices, network configurations, and monitoring mechanisms. Below are key mitigation strategies:

### 1. Input Validation and URL Filtering

Allowlist Trusted Domains:

- Restrict outgoing requests to specific, trusted domains or IP addresses.

Block Private and Localhost Addresses:

- Disallow requests to internal/private IP ranges (e.g., 127.0.0.1, 10.0.0.0/8) to prevent unauthorized access to internal services.

### Validate URL Structure

- Ensure user-supplied URLs follow the correct format and use only permitted schemes (e.g., http or https).

### 2. Network Configuration

### Restrict Outgoing Requests

- Implement firewall rules to limit the server's ability to make outgoing requests to untrusted destinations.

### Isolate Critical Systems

- Use network segmentation to separate web-facing servers from internal systems and sensitive services.

### Cloud Metadata Protection

- For cloud environments, configure access controls to restrict access to metadata endpoints (e.g., AWS 169.254.169.254):
    - Enable **IMDSv2** in AWS to enforce session-based metadata access.

### 3. Secure Request Libraries

### Control Request Behavior

- Disable automatic redirections in request-handling libraries to avoid unexpected responses.
- Enforce timeouts and limit the size of responses to prevent resource exhaustion.

### 4. Authentication and Access Control

#### Authenticate Internal Services

- Use authentication mechanisms (e.g., API keys, tokens) for internal APIs and services.

#### Role-Based Access

- Restrict server privileges to minimize the impact of potential SSRF attacks.

### 5. Monitoring and Detection

#### Log and Monitor Outgoing Requests

- Maintain detailed logs of outgoing HTTP requests, including destination URLs, IPs, and response statuses.

#### Anomaly Detection

- Use Intrusion Detection Systems (IDS) or Web Application Firewalls (WAF) to detect and block suspicious traffic patterns.

#### DNS Monitoring

- Monitor DNS queries for unauthorized domain lookups that may indicate SSRF exploitation.

### 6. Avoid Direct User Input for Requests

- Avoid directly using user-supplied URLs for making server-side requests. Instead, implement an indirect mapping approach:
    - Use database IDs to reference pre-approved URLs.

### 7. Conduct Regular Security Testing

#### Penetration Testing

- Test applications for SSRF vulnerabilities using tools like **Burp Suite** or **OWASP ZAP**.

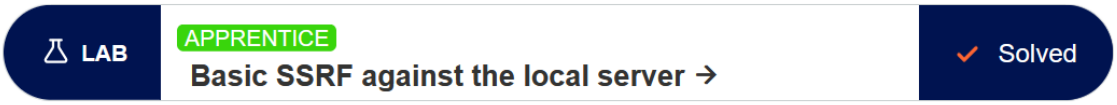
#### Code Reviews

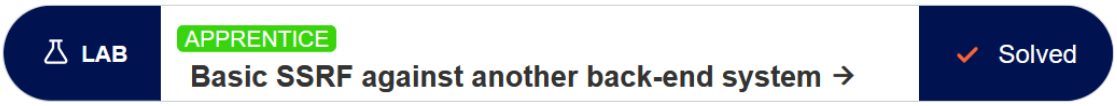
- Perform code reviews to identify insecure handling of user input and outgoing requests.

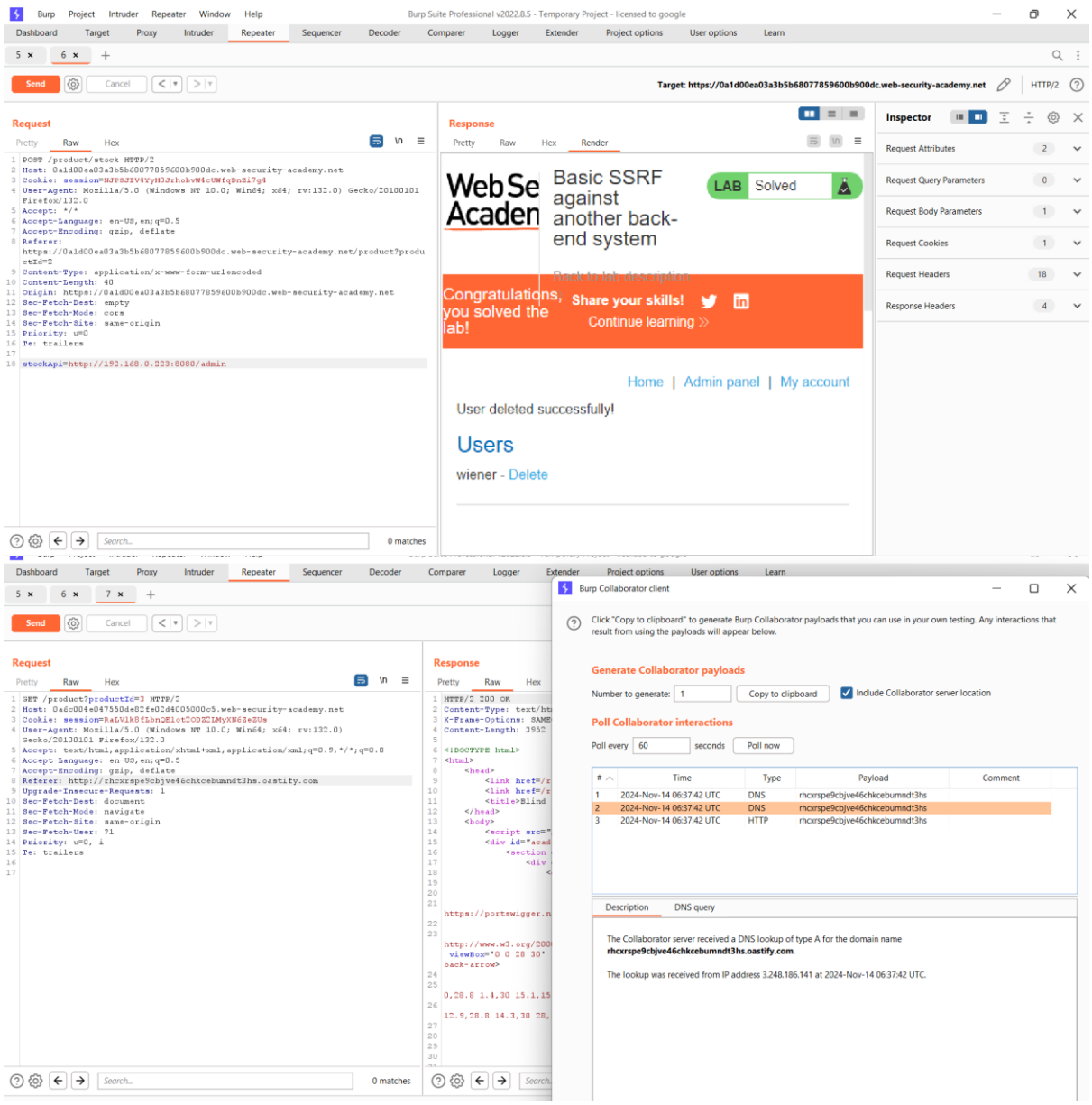
#### Automated Scanning

- Integrate automated vulnerability scanners to identify SSRF risks during development.

# Practical:


LAB
APPRENTICE
Basic SSRF against the local server →
Solved


LAB
APPRENTICE
Basic SSRF against another back-end system →
Solved


Burp Project Intruder Repeater Window Help
Burp Suite Professional v2022.8.5 - Temporary Project - licensed to google
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Send Cancel
Target: https://0a1d00ea03a3b5b68077859600b900dc.web-security-academy.net
HTTP/2
Request
Response
Inspector
Request Attributes
Request Query Parameters
Request Body Parameters
Request Cookies
Request Headers
Response Headers
Basic SSRF against another back-end system
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | Admin panel | My account
User deleted successfully!
Users
wiener - Delete
Burp Collaborator client
Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.
Generate Collaborator payloads
Number to generate: 1
Copy to clipboard
Include Collaborator server location
Poll Collaborator interactions
Poll every 60 seconds
Poll now
Time
Type
Payload
Comment
2024-Nov-14 06:37:42 UTC DNS rhcxrspe9cbjve46chkcebumndt3hs
2024-Nov-14 06:37:42 UTC HTTP rhcxrspe9cbjve46chkcebumndt3hs
Description
DNS query
The Collaborator server received a DNS lookup of type A for the domain name rhcxrspe9cbjve46chkcebumndt3hs.oastify.com.
The lookup was received from IP address 3.248.186.141 at 2024-Nov-14 06:37:42 UTC.

References:

1. https://owasp.org/www-community/attacks/Server_Side_Request_Forgery
2. https://www.f5.com/glossary/ssrf
3. https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/
4. https://portswigger.net/web-security/all-labs#server-side-request-forgery-ssrf