Server-Side Request Forgery (SSRF) Vulnerability

Taha Zouggari

June 2025

Goal of the Attack

The goal of a Server-Side Request Forgery (SSRF) attack is to abuse a functionality of a web server that fetches URLs on behalf of the user. By manipulating this behavior, an attacker can:

- Make the server send requests to internal services (e.g., `127.0.0.1` or cloud metadata endpoints).
- Access otherwise protected or unreachable resources.
- Enumerate internal networks and services.
- Potentially gain access to sensitive information or trigger backend actions.

This makes SSRF a powerful primitive for pivoting inside a target's internal network, even when no direct access is allowed from the attacker's machine.

Overview

Server-Side Request Forgery (SSRF) is a type of vulnerability that allows an attacker to make the vulnerable server perform HTTP or other protocol-based requests to arbitrary destinations on behalf of the attacker. This can include internal systems that are otherwise not accessible from outside the network.

Impact

Exploiting SSRF can lead to:

- Accessing internal-only services or applications.
- Enumerating internal infrastructure.
- Triggering unintended behaviors in backend systems.
- Potentially chaining with other vulnerabilities for further compromise.

Practical Example

In this scenario, the web application includes a functionality that accepts a `url` parameter and fetches content from it. The server hosting the application can reach internal resources, even though the attacker cannot directly access them.

Target

An internal web service is running on TCP port `1234` on the same server (or within the same internal network). The goal is to retrieve the contents of the webroot (e.g., `/`) of this internal service.

Exploit Strategy

By modifying the `url` parameter of the vulnerable functionality, the attacker can make the server fetch an internal URL. Example payload:

```
http://vulnerable-website.com/fetch?url=http://127.0.0.1:1234/
```

Here:

- `127.0.0.1` refers to the local host from the server's perspective.
- `:1234` specifies the internal service's port.
- `/` requests the root path of the internal service.

Bypassing Filters on `127.0.0.1`

Developers may attempt to block direct access to `127.0.0.1`. However, there are several alternate representations and IP encodings that resolve to the same loopback interface:

- `localhost`
- `0.0.0.0`
- `2130706433` (decimal representation of 127.0.0.1)
- `0x7f000001` (hexadecimal)
- `017700000001` (octal)
- `127.1`, `127.0.1.1`, etc. (other valid loopback IPs)
- `[::1]` (IPv6 loopback address)

Example payload using hexadecimal:

```
http://vulnerable-website.com/fetch?url=http://0x7f000001:1234/
```

DNS-Based SSRF Bypass

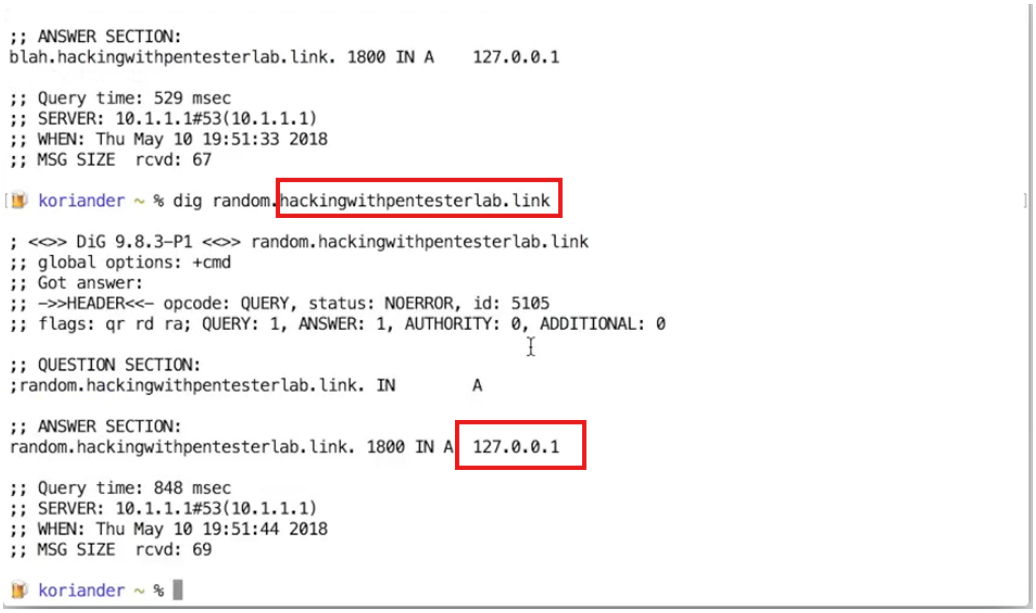
In some cases, filters block raw IP addresses like `127.0.0.1`. However, attackers can register a domain name that resolves to this loopback address, effectively bypassing the filter.

Example Case:

In the screenshot below, a DNS query is performed for the domain:

`random.hackingwithpentesterlab.link`

This domain is configured to resolve to `127.0.0.1`. The server sees it as a normal domain name and may allow the request, but it ultimately reaches a service running on the loopback interface.



Explanation

- The attacker controls the DNS record for `*.hackingwithpentesterlab.link`.
- They create a subdomain like `random.hackingwithpentesterlab.link` and point it to `127.0.0.1`.
- The vulnerable server queries this domain and is tricked into contacting its own local interface.

This technique bypasses naive blacklist-based filters that only look for the literal IP `127.0.0.1` or the keyword `localhost`.

Mitigations

To prevent SSRF, developers should:

- Validate and sanitize user-supplied URLs strictly.
- Resolve the final destination IP and compare it against a deny-list of internal ranges (e.g., 127.0.0.0/8, 10.0.0.0/8).
- Restrict server-side outbound traffic via firewall or allow-list.
- Disable unnecessary internal services or expose them only over secure VPNs.