GitHub Advisory Database / GitHub Reviewed / CVE-2025-25194

# Server-Side Request Forgery (SSRF) in activitypub_federation

Moderate severity | GitHub Reviewed | Published on Feb 10, 2025 in **LemmyNet/lemmy** • Updated on Feb 11, 2025

Vulnerability details | Dependabot alerts 0

**Package**

**activitypub_federation** (Rust)

| Affected versions | Patched versions |
|---|---|
| <= 0.6.2 | None |

## Description

### Summary

This vulnerability allows a user to bypass any predefined hardcoded URL path or security anti-Localhost mechanism and perform an arbitrary GET request to any Host, Port and URL using a Webfinger Request.

### Details

The Webfinger endpoint takes a remote domain for checking accounts as a feature, however, as per the ActivityPub spec (https://www.w3.org/TR/activitypub/#security-considerations), on the security considerations section at B.3, access to Localhost services should be prevented while running in production.
The library attempts to prevent Localhost access using the following mechanism (/src/config.rs):

```
pub(crate) async fn verify_url_valid(&self, url: &Url) -> Result<(), Error> {
        match url.scheme() {
            "https" => {}
            "http" => {
                if !self.allow_http_urls {
                    return Err(Error::UrlVerificationError(
                        "Http urls are only allowed in debug mode",
                    ));
                }
            }
            _ => return Err(Error::UrlVerificationError("Invalid url scheme")),
```

```
        };

        // Urls which use our local domain are not a security risk, no further ve
        if self.is_local_url(url) {
            return Ok(());
        }

        if url.domain().is_none() {
            return Err(Error::UrlVerificationError("Url must have a domain"));
        }

        if url.domain() == Some("localhost") && !self.debug {
            return Err(Error::UrlVerificationError(
                "Localhost is only allowed in debug mode",
            ));
        }

        self.url_verifier.verify(url).await?;

        Ok(())
    }
```

There are multiple issues with the current anti-Localhost implementation:

1. It does not resolve the domain address supplied by the user.
2. The Localhost check is using only a simple comparison method while ignoring more complex malicious tampering attempts.
3. It filters only localhost domains, without any regard for alternative local IP domains or other sensitive domains, such internal network or cloud metadata domains.

We can reach the verify_url_valid function while sending a Webfinger request to lookup a user's account (/src/fetch/webfinger.rs):

```
pub async fn webfinger_resolve_actor<T: Clone, Kind>(
    identifier: &str,
    data: &Data<T>,
) -> Result<Kind, <Kind as Object>::Error>
where
    Kind: Object + Actor + Send + 'static + Object<DataType = T>,
    for<'de2> <Kind as Object>::Kind: serde::Deserialize<'de2>,
    <Kind as Object>::Error: From<crate::error::Error> + Send + Sync + Display,
{
    let (_, domain) = identifier
        .splitn(2, '@')
        .collect_tuple()
        .ok_or(WebFingerError::WrongFormat.into_crate_error())?;
    let protocol = if data.config.debug { "http" } else { "https" };
    let fetch_url =
        format!("{protocol}://{domain}/.well-known/webfinger?resource=acct:{ident
    debug!("Fetching webfinger url: {}", &fetch_url);

    let res: Webfinger = fetch_object_http_with_accept(
        &Url::parse(&fetch_url).map_err(Error::UrlParse)?,
```

```
            data,
            &WEBFINGER_CONTENT_TYPE,
        )
        .await?
        .object;

        debug_assert_eq!(res.subject, format!("acct:{identifier}"));
        let links: Vec<Url> = res
            .links
            .iter()
            .filter(|link| {
                if let Some(type_) = &link.kind {
                    type_.starts_with("application/")
                } else {
                    false
                }
            })
            .filter_map(|l| l.href.clone())
            .collect();

        for l in links {
            let object = ObjectId::<Kind>::from(l).dereference(data).await;
            match object {
                Ok(obj) => return Ok(obj),
                Err(error) => debug!(%error, "Failed to dereference link"),
            }
        }
        Err(WebFingerError::NoValidLink.into_crate_error().into())
    }
```

The Webfinger logic takes the user account from the GET parameter “resource” and sinks the domain directly into the hardcoded Webfinger URL (“{protocol}://{domain}/.well-known/webfinger?resource=acct:{identifier}”) without any additional checks.
Afterwards the user domain input will pass into the “fetch_object_http_with_accept” function and finally into the security check on “verify_url_valid” function, again, without any form of sanitizing or input validation.
An adversary can cause unwanted behaviours using multiple techniques:

1. ***Gaining control over the query’s path:***
   An adversary can manipulate the Webfinger hard-coded URL, gaining full control over the GET request domain, path and port by submitting malicious input like: hacker@hacker_host:1337/hacker_path?hacker_param#, which in turn will result in the following string: http[s]://hacker_host:1337/hacker_path?hacker_param#/.well-known/webfinger?resource=acct:{identifier}, directing the URL into another domain and path without any issues as the hash character renders the rest of the URL path unrecognized by the webserver.

2. ***Bypassing the domain's restriction using DNS resolving mechanism:***
   An adversary can manipulate the security check and force it to look for internal services regardless the Localhost check by using a domain name that resolves into a local IP (such as: localh.st, for example), as the security check does not verify the resolved IP at all - any service under the Localhost domain can be reached.

3. ***Bypassing the domain's restriction using official Fully Qualified Domain Names (FQDNs):***
   In the official DNS specifications, a fully qualified domain name actually should end with a dot.
   While most of the time a domain name is presented without any trailing dot, the resolver will assume it exists, however - it is still possible to use a domain name with a trailing dot which will resolve correctly.
   As the Localhost check is mainly a simple comparison check - if we register a "hacker@localhost." domain it will pass the test as "localhost" is not equal to "localhost.", however the domain will be valid (Using this mechanism it is also possible to bypass any domain blocklist mechanism).

## PoC

1. Activate a local HTTP server listening to port 1234 with a "secret.txt" file:
   `python3 -m http.server 1234`
2. Open the "main.rs" file inside the "example" folder on the activitypub-federated-rust project, and modify the "beta@localhost" string into "hacker@localh.st:1234/secret.txt?something=1#".
3. Run the example using the following command:
   `cargo run --example local_federation axum`
4. View the console of the Python's HTTP server and see that a request for a "secret.txt" file was performed.

This proves that we can redirect the URL to any domain and path we choose.
Now on the next steps we will prove that the security checks of Localhost and blocked domains can be easily bypassed (both checks use the same comparison mechanism).

1. Now open the "instance.rs" file inside the "example" folder and view that the domain "malicious.com" is blocked (you can switch it to any desired domain address).
2. Change the same "beta@localhost" string into "hacker@malicious.com" and run the example command to see that the malicious domain blocking mechanism is working as expected.
3. Now change the "hacker@malicious.com" string into "hacker@malicious.com." string and re-initiate the example, view now that the check passed successfully.
4. You can combine both methods on "localhost." domain (or any other domain) to verify that the FQDNs resolving is indeed successful.

## Impact

Due to this issue, any user can cause the server to send GET requests with controlled path and port in an attempt to query services running on the instance's host, and attempt to execute a Blind-SSRF gadget in hope of targeting a known vulnerable local service running on the victim's machine.

## Fix Suggestion

Modify the domain validation mechanism and implement the following checks:

1. Resolve the domain and validate it is not using any invalid IP address (internal, or cloud metadata IPs) using regexes of both IPv4 and IPv6 addresses.
   For Implementation example of a good SSRF prevention practice you can review a similiar project such as "Fedify" (https://github.com/dahlia/fedify/blob/main/src/runtime/url.ts) which handles external URL resource correctly.
   Note that it is still needed to remove unwanted characters from the URL.
2. Filter the user's input for any unwanted characters that should not be present on a domain name, such as #,?,/, etc.
3. Perform checks that make sure the desired request path is the executed path with the same port.
4. Disable automatic HTTP redirect follows on the implemented client, as redirects can be used for security mechanisms circumvention.

## References

- GHSA-7723-35v7-qcxw
- https://nvd.nist.gov/vuln/detail/CVE-2025-25194

**dessalines** published to LemmyNet/lemmy on Feb 10, 2025

Published to the GitHub Advisory Database on Feb 10, 2025

Reviewed on Feb 10, 2025

Published by the National Vulnerability Database on Feb 10, 2025

Last updated on Feb 11, 2025

**Severity**

Moderate 4.0 / 10

**CVSS v3 base metrics**

| | |
|---|---|
| Attack vector | Network |
| Attack complexity | High |
| Privileges required | None |
| User interaction | None |
| Scope | Changed |
| Confidentiality | Low |
| Integrity | None |
| Availability | None |

Learn more about base metrics

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:L/I:N/A:N

**EPSS score**

0.035% (10th percentile)

**Weaknesses**

▶ CWE-918

**CVE ID**

CVE-2025-25194

**GHSA ID**

GHSA-7723-35v7-qcxw

**Source code**

LemmyNet/activitypub-federation-rust

**Credits**

 **nnfrog** Reporter

This advisory has been edited. See History.

See something to contribute? Suggest improvements for this vulnerability.