Lemmy: Risoluzione del filtraggio anti-SSRF e gestione media

Lemmy: Risoluzione del filtraggio anti-SSRF e gestione media

Il deployment di un'istanza federata Lemmy in ambienti di rete isolati o instradati richiede una configurazione accurata del routing tra container e proxy.

In scenari con domini interni o instradamenti specifici, l'interazione tra i servizi containerizzati e le policy di sicurezza di Lemmy può generare criticità, in particolare durante la validazione e il salvataggio degli avatar utente gestiti dal microservizio Pictrs. Di seguito viene esaminata l'architettura dell'infrastruttura e la mitigazione del blocco indotto dalla protezione anti-SSRF.

L'Architettura Base

Il setup ideale prevede che i servizi applicativi (Lemmy, Lemmy-UI, Pictrs, PostgreSQL) operino all'interno di una rete Docker dedicata, con un reverse proxy Nginx a gestire il traffico in ingresso.

L'instradamento avviene attraverso un reverse proxy di frontiera:

1. Il servizio d'ascolto espone il servizio sulla porta 80 e lo inoltra verso `127.0.0.1:10080`.
2. Il file `docker-compose.yml` pubblica la porta del container Nginx su `127.0.0.1:10080`.
3. Nginx all'interno della rete Docker riceve il traffico e lo instrada: le chiamate API verso `lemmy:8536`, il frontend verso `lemmy-ui:1234`.

Aggiungiamo qualche parametro di configurazione (como definire `LEMMY_UI_LEMMY_EXTERNAL_HOST` e l'`hostname` nel file hjson con il nostro magico indirizzo `.onion`) e il gioco sembra fatto. L'istanza è online, i post funzionano, federiamo (se lo abbiamo configurato). Tutto bellissimo.

Fino a quando un utente non decide di cambiarsi l'avatar.

Il Bug dell'Avatar Fantasma (InvalidUrl)

L'utente carica la foto, preme "Salva", il server risponde picche. Se andate a spulciare i log del container di Lemmy, troverete un simpatico errore `400 Bad Request` seguito da un poco lusinghiero `InvalidUrl` generato dalla funzione `save_user_settings`.

Cosa diavolo sta succedendo?

Quando si salva un avatar, Lemmy non riceve il file, ma l'URL dell'immagine generato da Pictrs. Il backend di Lemmy prende questo URL (`http://vostro-dominio.onion/pictrs/image/...`) e cerca di connettersi per validarlo.

Ma il container di Lemmy non ha accesso nativo alla rete Tor per risolvere i domini `.onion`. Se siamo stati furbi, avremo aggiunto un "alias di rete" nel docker-compose per fare in modo che il nostro dominio `.onion` risolva internamente verso l'IP del container Nginx. Problema risolto, no?

No. Qui entra in gioco la sicurezza anti-SSRF (Server-Side Request Forgery) di Lemmy. Lemmy usa una funzione interna chiamata `validate_link_ip()`. Questa funzione prende l'IP appena risolto dal dominio e controlla se è un indirizzo IP privato (10.x.x.x, 192.168.x.x, 172.16.x.x). Poiché le reti interne di Docker usano tipicamente range come `172.28.0.x`, Lemmy vede un IP privato, alza i proverbiali scudi e sputa fuori un `InvalidUrl` rifiutandosi di processare l'immagine. Lemmy si sta letteralmente proteggendo da sé stesso.

Non esiste una variabile d'ambiente per disattivare questo controllo. Non c'è una whitelist. O cambiamo il codice sorgente di Rust e ricompiliamo tutto, o siamo fregati.

L'Hack del CGNAT: Come fregare il guardiano

La crittografia e la sicurezza ci insegnano che quando non possiamo sfondare la porta, dobbiamo capire come ragiona la serratura.

Siamo andati a spulciare il codice sorgente di Lemmy, specificamente la funzione `v4_is_invalid()` usata per bloccare gli IP. Questa funzione controlla che l'IP non sia privato (`is_private()`), di loopback, multicast, ecc.

Ma sapete cosa si sono dimenticati di includere nella lista nera? **Lo spazio CGNAT (Carrier-Grade NAT)** **`100.64.0.0/10`**.

Questo spazio di indirizzi è solitamente usato dagli ISP per nattare i clienti. Per la funzione Rust di Lemmy, un indirizzo `100.64.x.x` è a tutti gli effetti un normalissimo indirizzo IP *pubblico* e perfettamente valido.

La soluzione è di un'eleganza assoluta: basta forzare Docker a usare questo spazio di indirizzi per la rete del nostro stack.

Nel nostro `docker-compose.yml`, invece di lasciare che Docker scelga subnet a caso nella famiglia 172.x.x.x, definiamo la rete `default` in fondo al file in questo modo:

```
networks:
  default:
    ipam:
      config:
        - subnet: 100.64.0.0/24
```

Riavviate lo stack con `docker compose down` e `docker compose up -d`.

Cosa succede ora?

1. Nginx (il nostro proxy interno con l'alias `.onion`) riceverà un IP come `100.64.0.2`.
2. Lemmy Backend cerca di salvare l'avatar, risolve il nostro dominio `.onion` usando il DNS di Docker.
3. Il DNS di Docker risponde: "`100.64.0.2`".
4. Il guardiano anti-SSRF di Lemmy controlla l'IP, non lo trova tra quelli privati, e dice: "*Oh, questo è un IP pubblico! Nessun pericolo SSRF. Procedere!*"
5. Lemmy si connette a Nginx, valida l'immagine e l'avatar viene salvato magicamente.

Abbiamo ingannato il sistema di sicurezza usando una sua stessa svista di implementazione, senza toccare mezza riga di codice Rust e mantenendo l'istanza isolata e sicura dietro Tor.

Semplice, pulito, hacker-style.

[Che il talpone sia con voi]

[EOF]

Fluxer Icon

Canale di discussione e supporto

Canale di discussione e supporto. Per confrontarti su configurazioni o altro entra su fluxer.dreadful.work (https://fluxer.dreadful.work/invite/dreadful).