OAuth Authorization Code Theft via Unvalidated Host Header in OIDC Flow

Moderate **tkurki** published **GHSA-cxj8-ggf2-p57c** 4 days ago

Package

signalk-server (npm)

Affected versions	Patched versions
>=v2.20.0	v2.24.0

Description

Summary

SignalK Server contains a code-level vulnerability in its OIDC login and logout handlers where the unvalidated HTTP Host header is used to construct the OAuth2 redirect_uri. Because the redirectUri configuration is silently unset by default, an **attacker spoof the Host header** to steal OAuth authorization codes and hijack user sessions in realistic deployments as The OIDC provider will then send the authorization code to whatever domain was injected.
The OIDC specification requires redirect_uri to be pre-registered and not derived from untrusted input. Constructing it from the Host header violates this requirement and introduces a trust boundary break.
This risk is actively amplified by SignalK's official documentation, which instructs administrators to deploy an Nginx configuration that forwards the vulnerable Host header, exposing production environments.

Vulnerability Root Cause

Two factors combine to create this vulnerability:

Factor 1: redirectUri is optional with an unsafe fallback
In types.ts:30, redirectUri is declared as optional

```
export interface OIDCConfig {
  // ...
  redirectUri?: string   // ← Optional, no default value
```

```
  // ...
}
```

The defaults in types.ts:175-185 do not include a redirectUri: never checks or warns about a missing redirectUri. This means a fully "valid" OIDC configuration can exist without redirectUri, silently activating the vulnerable fallback path.

```
export const OIDC_DEFAULTS: Omit<OIDCConfig, 'issuer' | 'clientId' | 'clientSecret
= {
  enabled: false,
  scope: 'openid email profile',
  defaultPermission: 'readonly',
  autoCreateUsers: true,
  providerName: 'SSO Login',
  autoLogin: false
  // ← No redirectUri default
}
```

Factor 2: Unsafe Host header usage in two locations
Location 1 — Login handler in oidc-auth.ts:278-282:

```
const protocol = req.secure ? 'https' : 'http'
const host = req.get('host')                          // ← Attacker-controlled
const redirectUri =
  oidcConfig.redirectUri ||                            // ← Only safe if explicitly
set
  `${protocol}://${host}${skAuthPrefix}/oidc/callback` // ← Uses attacker's Host
```

This redirectUri flows into createAuthState() → buildAuthorizationUrl() → OIDC provider's redirect_uri parameter. The OIDC provider will then send the authorization code to whatever domain was injected.

Location 2 — Logout handler in oidc-auth.ts:513-515:

```
const protocol = req.secure ? 'https' : 'http'
const host = req.get('host')                            // ← Same pattern
const fullPostLogoutUri = `${protocol}://${host}${postLogoutRedirect}`
```

This constructs the post_logout_redirect_uri sent to the OIDC provider's end_session_endpoint, allowing an attacker to redirect the user to an attacker controlled domain after logout.

Official Documentation Enables the Attack

SignalK's own security documentation at docs/security.md:222-228 provides the recommended nginx reverse proxy configuration:
The proxy_set_header Host $host; directive forwards the client-supplied Host header to the backend unmodified. Without this directive, nginx would replace the Host header with the upstream address (localhost:3000), which would neutralize the injection.

```
location / {
    proxy_pass http://localhost:3000;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header Host $host;   # ← Forwards client's Host header to SignalK
}
```

Administrators who follow the official documentation are directly enabling this vulnerability behind their reverse proxy.

Proof of Concept

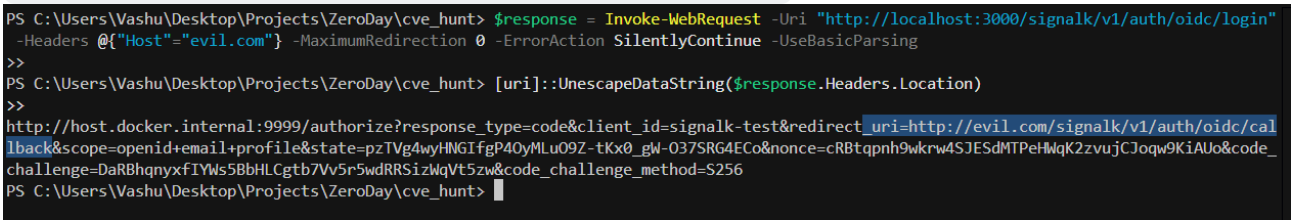
Tested against SignalK Server v2.23.0 in Docker with OIDC enabled .

Step 1 — Send login request with injected Host header:

`$response = Invoke-WebRequest -Uri "http://localhost:3000/signalk/v1/auth/oidc/login" -Headers @{"Host"="evil.com"} -MaximumRedirection 0 -ErrorAction SilentlyContinue -UseBasicParsing`

Step 2: Decode and print the injected redirect URL

`[uri]::UnescapeDataString($response.Headers.Location)`



Impact

- **Authorization Code Theft:** The OIDC provider sends the OAuth authorization code to the attacker's domain instead of the legitimate server.
- **Session Hijack:** The attacker can exchange the stolen code for tokens and create a session as the victim user.
- **Logout Redirect Hijack:** The logout handler has the same pattern, allowing post-logout redirection to an attacker domain (phishing opportunity).

Severity

Moderate 6.1 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	Required
Scope	Changed
Confidentiality	Low
Integrity	Low
Availability	None

Learn more about base metrics

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

CVE ID

CVE-2026-34083

Weaknesses

- CWE-346
- CWE-601

Credits

VashuVats Reporter