0-DAY NO CVE HIGH PUBLIC

Prometheus v3.15.0, remote_write Bearer on cross-host 3xx

prometheus/common #901 strips scrape credentials on cross-host redirects. remote_write azuread and google_iam wrap the client after that strip and put Bearer back on every hop. Labbed v3.15.0. No CVE yet.

NAME Prometheus v3.15.0, remote_write Bearer on cross-host 3xx	TYPE 0-day analysis
CVE n/a	CVE RISK high
DISCLOSURE STATUS public	VENDOR Prometheus
AFFECTED Prometheus v3.15.0 (5241a27) remote_write with azuread: or google_iam:. Default prometheus.yml has no remote_write. Scrape oauth2 on the same pin does not forward.	PUBLISHED 12 Oct 2026
UPDATED 12 Oct 2026	TAGS 0-day, prometheus, remote-write, credential-leak, cwe-201, google-iam, azuread

TL;DR

Lab files and proof-of-concept here.

01 The wrappers sit outside the strip

I am @abraxas_null. The proof of concept is on GitHub: **abraxas/prometheus-bearer-redirect** (loopback client). The lab stack is `lab/`: docker-compose.yml, prometheus.yml, run.sh, poc.py. Authorized lab only. It talks to loopback.

This is **Prometheus v3.15.0** (`5241a27fe3c6983549fccc32f6e65917408c63cd`). Image `prom/prometheus:v3.15.0`. No CVE yet. CWE-201. 8.6 High (`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N`). Remote credential leak. Config-gated `remote_write` with `azuread:` or `google_iam:`.

prometheus/common v0.71.0 already strips scrape / HTTPClientConfig credentials on a cross-host redirect. That was #901. `NewWriteClient` builds that client first, then wraps Azure AD or Google IAM outside it. `follow_redirects` still defaults true. Go already dropped the copied `Authorization` header on a 307 to another host. These RoundTrippers set it again.

Default `prometheus.yml` has no `remote_write`. This is not a default-install High. It is the leftover of a fix that stopped at scrape.

Not CVE-2026-42151. That was `client_secret` on `/-/config`. This pin's field is typed `Secret`.

02 Why Prometheus

I had just finished unpublished Crit/High hunts of Grafana OSS. Prometheus is the other half of that monitoring pair: a large Go tree, a documented unauth HTTP model,

and a 3.11/3.12 secret-redaction wave that is a good place to look for wrappers that sat outside the strip.

I only wanted Critical or High. Local is in-scope. Medium stays on the floor unless it chains. Alertmanager, node_exporter, blackbox, pushgateway, and client_golang are other trees.

The vendor security model is honest. Unauthenticated PromQL, `/metrics`, federation, and scrape of operator-configured targets are intended. A finding needed auth bypass when `--web.config.file` is set, unauth mutating admin/lifecycle when those flags are off, a `config.Secret` leak, RCE, or a leftover of a published credential-strip that still fires.

03 Pinning v3.15.0

Latest GA. Tag `v3.15.0`, commit `5241a27`. grpc is already `v1.83.2` (GO-2026-6348 patched). common is v0.71.0. The 3.15 FEATURE line also added Unix domain socket scrape, which is a sibling leftover and a different pack.

Intake is maintainers plus CC security@prometheus.io. GitHub private advisory. Unpaid CNCF volunteer project.

04 The skip list is half the hunt

On this pin the interesting ones were already closed:

- GHSA-fw8g / CVE-2026-44903 old-ui heatmap XSS. Medium. Patched.
- GHSA-8rm2 / CVE-2026-42154 remote-read snappy DoS. 32 MiB cap.
- GHSA-wg65 / CVE-2026-42151 Azure AD `client_secret` via `/-/config`. Field is `Secret` here.

- GHSA-vffh / CVE-2026-40179 metric/label stored XSS. `escapeHTML` on this pin.
- GHSA-4v48 / CVE-2022-46146 toolkit basic-auth cache poison. toolkit v0.19.0.
- GHSA-vx57 / CVE-2021-29622 `/new` open redirect. Route gone.
- STACKIT SD secrets on `/-/config`. `config.Secret`.
- 3.12.0 remote-write snappy declared length >32 MiB.
- 3.13.4 grpc HTTP/2 DATA fragmentation.

I did not re-file any of that.

05 Nineteen surfaces, two leftovers

I split the tree. Web/auth/admin, remote write/read, `/-/config` secrets, UI XSS, scrape/UDS, service discovery, PromQL, TSDB, federation, notifier/templates, snappy/zstd/gzip, lifecycle/reload, k8s SD, HTTP client, OTLP/plugins, rules API, Azure/GCP auth, query API, 3.15 FEATURE surfaces.

Most of that is Prometheus doing what Prometheus said it would do.

Admin API and lifecycle HTTP default off. Remote-write receiver and OTLP receiver default off. Dedicated credential fields marshal as `<secret>`. UI graph tooltips `escapeHTML`. Remote-read snappy is `LimitReader` + `DecodedLen` vs 32 MiB. Unauth PromQL is the product. Pure DoS is not a private security report.

Default OSS v3.15.0 is NONE unpublished Crit/High. Stock image scrapes `localhost:9090` over TCP.

The leftovers that survived are config-gated. This pack is G1: the remote_write wrappers. G2 is unsanitized `__unix_socket__` Dial, a different post.

06 Common #901 was supposed to be a lesson

Scrape, HTTP SD, and HTTPClientConfig oauth2 / basic / bearer already skip `Authorization` on a cross-host redirect. `isCrossHostRedirect` lives in prometheus/common. v0.69.0 shipped it. v0.71.0 is on this pin.

`NewWriteClient` builds that client first, then wraps:

```go
if conf.AzureADConfig != nil {
    t, err = azuread.NewAzureADRoundTripper(conf.AzureADConfig, t)
}
if conf.GoogleIAMConfig != nil {
    t, err = googleiam.NewRoundTripper(conf.GoogleIAMConfig, t)
}
```

`azureADRoundTripper.RoundTrip` always `Header.Set("Authorization", "Bearer "+token)` with no `isCrossHostRedirect`. `googleiam.NewRoundTripper` is `google.golang.org/api/transport/http` → `oauth2.Transport`, which `SetAuthHeader`s on every hop.

Go's client already dropped the copied header on a cross-host 3xx. These wrappers put it back. `follow_redirects` defaults true (`DefaultRemoteWriteHTTPClientConfig`).

If the configured remote_write URL, or a hop it 301/302/307s to, is a host the attacker controls, that host gets a live Azure AD or GCP `monitoring.write` Bearer.

Marathon SD `authTokenRoundTripper` is the same class and not a separate default finding. SD URL is operator config.

07 What I almost filed instead

Unauth PromQL. Documented.

`/-/config` dumping `client_secret`. CVE-2026-42151. Typed `Secret` on this pin.

Scrape oauth2 still forwarding Bearer. It does not. The lab's negative control is the same process scraping hop `/metrics` with oauth2, following a 302 to the same oracle, and sending no Authorization.

Calling this a default-install High. `remote_write` is empty until you turn it on. The leftover is the wrap, not the empty YAML.

`remote_write.headers` setting `Authorization`. Rejected. Docs already say do not put secrets in non-Secret fields.

08 The lab

Image `prom/prometheus:v3.15.0`. Loopback Prometheus `http://127.0.0.1:19380`, oracle `http://127.0.0.1:19381`. `run.sh` writes a lab `gcp-sa.json` with `token_uri: http://hop:8080/token` so google_iam never talks to Google.

google_iam remote_write POSTed to hop:8080/write, which 307d to oracle:8080/write. The oracle received `Authorization: Bearer lab-gcp-PROMETHEUS-BEARER-REDIRECT-WITNESS`. The same process scraped hop:8080/metrics with oauth2, followed a 302 to the same oracle, and did not forward Bearer.

Replay:

PLAIN TEXT Copy

```
git clone https://github.com/abraxas/prometheus-bearer-redirect
cd prometheus-bearer-redirect/lab
```

```
./run.sh
```

Bind it to loopback. The loopback client is `lab/run.sh` with a banner on it.

PLAIN TEXT Copy

```
SUCCESS prometheus-bearer-redirect how=live rw-method=POST rw-bearer=yes token-ha
```

09 Wrong turns, in the order I made them

Re-filing CVE-2026-42151. `/-/config` is redacted. The leftover is outbound Bearer on 3xx.

Treating scrape oauth2 as still leaky. Common #901 holds. The wrappers sit outside that client.

Azure first. azuread is the same wrap site. google_iam is easier to stub without calling Microsoft. The lab used google_iam. azuread is inferred from the pin source.

Default YAML as the bug. Empty `remote_write` is not this leftover. The High is gated on `azuread:` or `google_iam:`.

A reverse shell. Theatre. The oracle is a Bearer on the redirected POST.

10 What an attacker can do

Operator has `remote_write` with `azuread:` or `google_iam:`. The RW URL, or a hop it 3xxs to, is a host the attacker controls.

PRIVILEGE	METHOD / SINK
Read the live GCP `monitoring.write` Bearer on a cross-host 3xx	`google_iam` rem 307 to another h
Same leak for `azuread`	`azureADRoundTr` always `Header.Set("Au` `"Bearer "+token` `isCrossHostRed`
Replay that token against Azure Monitor / GCP Monitoring as that identity	Token is a real cl when those auth configured
Scrape / oauth2 / basic / bearer on the same pin	HTTPClientConfi from common v0
Default OSS	`remote_write` e
`remote_write.headers` **setting** `Authorization`	rejected

PRIVILEGE	METHOD / SINK
`/-/config` **dumping** `client_secret`	CVE-2026-42151 `Secret` on this p
Prometheus admin / lifecycle	flags default off
RCE / a shell	not this leftover

11 The fix

Skip attaching Azure AD / Google IAM Bearer when `isCrossHostRedirect(req)`, same as common #901. Do not re-set Authorization after Go dropped it on a cross-host 3xx.

Until then, do not point `remote_write` with `azuread:` or `google_iam:` at a URL that 3xxs, and do not let anyone you do not already trust with that cloud identity own a hop on that path. `follow_redirects: false` is a workaround, not a fix.

I am not going to print a remote_write YAML you can paste at someone else's Prometheus. The missing `isCrossHostRedirect` is the useful part. If you own the process, run the proof of concept against loopback.

12 References

- Proof of concept: abraxas/prometheus-bearer-redirect · prometheus-bearer-redirect-Abraxas-Labs.py
- Lab: `lab/` · docker-compose.yml · prometheus.yml · run.sh · poc.py
- @abraxas_null · github.com/abraxas · abraxaslabs.tech ·

abraxas.null@proton.me

- CWE-201
- Tree: prometheus/prometheus v3.15.0 (`5241a27fe3c6983549fccc32f6e65917408c63cd`) · `storage/remote/client.go` · `azuread.go` · `googleiam.go`
- prometheus/common #901
- Nearby patched (not this wrap): CVE-2026-42151
- Sibling leftover: prometheus-unix-socket-dial

Abraxas Labs

Independent security research by abraxas.

Analysis of 0-days, n-days, malware, and the leftovers that still matter.

GitHub **@abraxas_null** **abraxas.null@proton.me** **abraxaslabs.tech**

Independent research. Private analysis by arrangement. No ads. abraxas.null@proton.me · For defensive awareness and authorized testing only.

© 2026 Abraxas Labs