Publish Reader Can Arbitrarily Delete Attribute View Files via `/api/av/removeUnusedAttributeView`

High 88250 published GHSA-7m5h-w69j-qggg last week

Package

siyuan (Go)

Affected versions

<=v3.6.3

Patched versions

v3.6.4

Description

Summary

An authenticated publish-service reader can invoke `/api/av/removeUnusedAttributeView` and cause persistent deletion of arbitrary attribute view (`AV`) definition files from the workspace.

The route is protected only by generic `CheckAuth`, which accepts publish `RoleReader` requests. The handler forwards a caller-controlled `id` directly into a model function that deletes `data/storage/av/<id>.json` without verifying either:

- that the caller is allowed to perform write/destructive actions; or
- that the target AV is actually unused.

This is a persistent integrity and availability issue reachable from the publish surface.

Root Cause

1. Publish users are issued a `RoleReader` JWT

- kernel/model/auth.go

```
ClaimsKeyRole: RoleReader,
```

2. The publish reverse proxy forwards that token upstream

- kernel/server/proxy/publish.go
- kernel/server/proxy/publish.go

3. `CheckAuth` accepts `RoleReader`

- kernel/model/session.go

```
if role := GetGinContextRole(c); IsValidRole(role, []Role{
    RoleAdministrator,
    RoleEditor,
    RoleReader,
}) {
    c.Next()
    return
}
```

4. The route is exposed with `CheckAuth` only

- kernel/api/router.go

```
ginServer.Handle("POST", "/api/av/removeUnusedAttributeView", model.CheckAuth, rem u
```

There is no `CheckAdminRole` and no `CheckReadonly`.

5. The handler forwards attacker-controlled `id` directly to the delete sink

- kernel/api/av.go

```
avID := arg["id"].(string)
model.RemoveUnusedAttributeView(avID)
```

6. The model deletes the AV file unconditionally

- kernel/model/attribute_view.go

```
func RemoveUnusedAttributeView(id string) {
    absPath := filepath.Join(util.DataDir, "storage", "av", id+".json")
    if !filelock.IsExist(absPath) {
        return
    }
    ...
```

```
    if err = filelock.RemoveWithoutFatal(absPath); err != nil {
        ...
        return
    }
    IncSync()
}
```

Crucially, this function does **not** verify that the supplied AV is actually unused. The name of the function suggests a cleanup helper, but the implementation is really "delete AV file by id if it exists".

Attack Prerequisites

- Publish service enabled
- Attacker can access the publish service
- If publish auth is enabled, attacker has valid publish-reader credentials
- Attacker knows an `avID`

Obtaining `avID`

`avID` is not secret. It is exposed extensively in frontend markup as `data-av-id`.

Examples:

- app/src/protyle/render/av/render.ts
- app/src/protyle/render/av/layout.ts
- app/src/protyle/render/av/groups.ts

Any publish-visible database/attribute view can therefore disclose a valid `avID` to the attacker.

Exploit Path

1. Attacker browses published content containing an attribute view.
2. Attacker extracts the `data-av-id` value from the page/DOM.
3. Attacker sends a POST request to `/api/av/removeUnusedAttributeView` through the publish service.
4. Publish proxy injects a valid `RoleReader` token.
5. `CheckAuth` accepts the request.
6. The handler passes the attacker-controlled `avID` to `model.RemoveUnusedAttributeView`.
7. The backend deletes `data/storage/av/<avID>.json`.

Proof of Concept

Request:

```
POST /api/av/removeUnusedAttributeView HTTP/1.1
Host: <publish-host>:<publish-port>
Content-Type: application/json
Authorization: Basic <publish-account-creds-if-enabled>

{
  "id": "<exposed-data-av-id>"
}
```

Expected result:

- HTTP 200
- backend increments sync state
- the target attribute view file is removed from `data/storage/av/`
- published and local workspace behavior for that AV becomes broken until restored from history or recreated

Impact

This gives a low-privileged publish reader a destructive persistent write primitive against workspace data.

Practical consequences include:

- deletion of live attribute view definitions
- corruption/breakage of published database views
- breakage of local workspace rendering and AV-backed relationships
- operational disruption until restore or manual repair

The bug affects integrity and availability, not merely UI state.

Recommended Fix

At minimum:

1. Block publish/read-only users from this route.
2. Require admin/write authorization.
3. Re-validate that the target AV is actually unused before deletion.

Safe router fix:

```
ginServer.Handle("POST", "/api/av/removeUnusedAttributeView",
    model.CheckAuth,
    model.CheckAdminRole,
    model.CheckReadonly,
```

```
    removeUnusedAttributeView,
)
```

And inside the model or handler, reject deletion unless the target `id` is present in `UnusedAttributeViews(...)`.

Severity

High 8.1 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	Low
User interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	High
Availability	High

Learn more about base metrics

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

CVE ID

CVE-2026-40259

Weaknesses

▶ CWE-285

Credits

ch1nhpd Reporter