Broken Access Control in Global Gym Configuration Update Endpoint

High **rolandgeider** published **GHSA-xppv-4jrx-qf8m** 2 days ago

Package

Affected versions

Tested on wger 2.5.0a1 (wger/server:latest, image sha256:c80a…b2a4f) and source checkout at commit bf7ed9f57dc70681f1b48ead3a50aa1200888fa5 (git describe 2.4-106-gbf7ed9f57).

Patched versions

>= 2.5

Description

Summary

wger exposes a global configuration edit endpoint at `/config/gym-config/edit` implemented by `GymConfigUpdateView`. The view declares `permission_required = 'config.change_gymconfig'` but does not enforce it because it inherits `WgerFormMixin` (ownership-only checks) instead of the project's permission-enforcing mixin (`WgerPermissionMixin`) .

The edited object is a singleton (`GymConfig(pk=1)`) and the model does not implement `get_owner_object()` , so `WgerFormMixin` skips ownership enforcement. As a result, a low-privileged authenticated user can modify installation-wide configuration and trigger server-side side effects in `GymConfig.save()` .

This is a vertical privilege escalation from a regular user to privileged global configuration control. The application explicitly declares permission_required = 'config.change_gymconfig', demonstrating that the action is intended to be restricted; however, this requirement is never enforced at runtime.

Affected endpoint

The config URLs map as follows.

File: `wger/config/urls.py`

```
patterns_gym_config = [
    path('edit', gym_config.GymConfigUpdateView.as_view(), name='edit'),
]

urlpatterns = [
    path(
        'gym-config/',
        include((patterns_gym_config, 'gym_config'), namespace='gym_config'),
    ),
]
```

This resolves to:

`/config/gym-config/edit`

Root cause

The view declares a permission but does not enforce it

File: `wger/config/views/gym_config.py`

```
class GymConfigUpdateView(WgerFormMixin, UpdateView):
    model = GymConfig
    fields = ('default_gym',)
    permission_required = 'config.change_gymconfig'
    success_url = reverse_lazy('gym:gym:list')
    title = gettext_lazy('Edit')

    def get_object(self):
        return GymConfig.objects.get(pk=1)
```

The permission string exists, but `WgerFormMixin` does not check `permission_required`.

The project’s permission mixin exists but is not used

File: `wger/utils/generic_views.py`

```
class WgerPermissionMixin:
    permission_required = False
    login_required = False
```

```
    def dispatch(self, request, *args, **kwargs):
        if self.login_required or self.permission_required:
            if not request.user.is_authenticated:
                return HttpResponseRedirect(
                    reverse_lazy('core:user:login') + f'?next={request.path}'
                )

            if self.permission_required:
                has_permission = False
                if isinstance(self.permission_required, tuple):
                    for permission in self.permission_required:
                        if request.user.has_perm(permission):
                            has_permission = True
                elif request.user.has_perm(self.permission_required):
                    has_permission = True

                if not has_permission:
                    return HttpResponseForbidden('You are not allowed to access this obj

        return super(WgerPermissionMixin, self).dispatch(request, *args, **kwargs)
```

`GymConfigUpdateView` does not inherit this mixin, so none of the login/permission logic runs.

The mixin that *is* used performs only ownership checks, and `GymConfig` has no owner

File: `wger/utils/generic_views.py`

```
class WgerFormMixin(ModelFormMixin):
    def dispatch(self, request, *args, **kwargs):
        self.kwargs = kwargs
        self.request = request

        if self.owner_object:
            owner_object = self.owner_object['class'].objects.get(pk=kwargs[self.owner_o
        else:
            try:
                owner_object = self.get_object().get_owner_object()
            except AttributeError:
                owner_object = False

        if owner_object and owner_object.user != self.request.user:
            return HttpResponseForbidden('You are not allowed to access this object')

        return super(WgerFormMixin, self).dispatch(request, *args, **kwargs)
```

File: `wger/config/models/gym_config.py`

```
class GymConfig(models.Model):
    default_gym = models.ForeignKey(
        Gym,
```

```
        verbose_name=_('Default gym'),
        # ...
        null=True,
        blank=True,
        on_delete=models.CASCADE,
    )
    # No get_owner_object() method
```

Because `GymConfig` does not implement `get_owner_object()`, `WgerFormMixin` catches `AttributeError` and sets `owner_object = False`, skipping any access restriction.

Security impact

This is not a cosmetic setting: `GymConfig.save()` performs installation-wide side effects.

File: `wger/config/models/gym_config.py`

```
def save(self, *args, **kwargs):
    if self.default_gym:
        UserProfile.objects.filter(gym=None).update(gym=self.default_gym)

        for profile in UserProfile.objects.filter(gym=self.default_gym):
            user = profile.user
            if not is_any_gym_admin(user):
                try:
                    user.gymuserconfig
                except GymUserConfig.DoesNotExist:
                    config = GymUserConfig()
                    config.gym = self.default_gym
                    config.user = user
                    config.save()

    return super(GymConfig, self).save(*args, **kwargs)
```

On deployments with multiple gyms, this allows a low-privileged user to tamper with tenant assignment defaults, affecting new registrations and bulk-updating existing users lacking a gym. This permits unauthorized modification of installation-wide state and bulk updates to other users' records, violating the intended administrative trust boundary.

Proof of concept (local verification)

Environment: local docker compose stack, accessed via `http://127.0.0.1:8088/en/`.

Observed behavior

An unauthenticated user can reach the endpoint via GET; POST requires authentication and redirects to login.
An authenticated low-privileged user can submit the form and change the global singleton. After the save, the application redirects to `success_url = reverse_lazy('gym:gym:list')` (e.g. `/en/gym/list`), which is permission-protected; therefore the browser may display a “Forbidden” page even though the global update already succeeded.

DB evidence (before/after)

Before submission:

```
default_gym_id= None
profiles_gym_null= 1
```

After a low-privileged user submitted the form setting `default_gym` to gym id `1`:

```
default_gym_id= 1
profiles_gym_null= 0
```

Recommended fix

Ensure permission enforcement runs before the form dispatch.

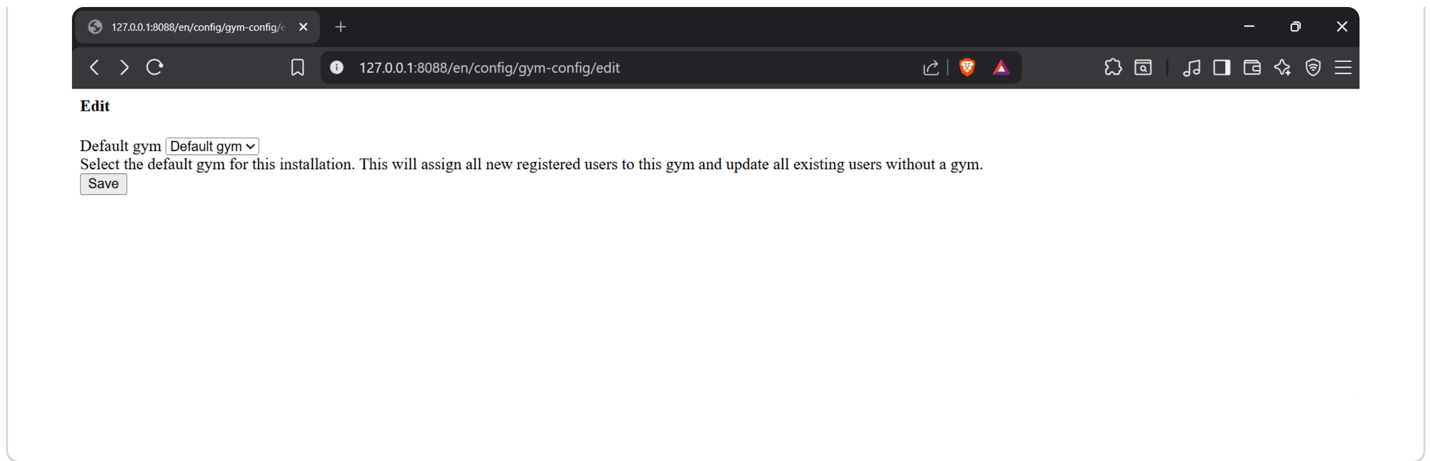
Using the project mixin (order matters):

```
class GymConfigUpdateView(WgerPermissionMixin, WgerFormMixin, UpdateView):
    permission_required = 'config.change_gymconfig'
    login_required = True
```

Alternatively, use Django’s `PermissionRequiredMixin` (and `LoginRequiredMixin`) directly.

Conclusion

The view explicitly declares permission_required = 'config.change_gymconfig', which demonstrates developer intent that this action be restricted. The fact that it is not enforced constitutes improper access control regardless of perceived business impact.



Severity

High 7.6 / 10

CVSS v3 base metrics

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

Learn more about base metrics

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

CVE ID

CVE-2026-40474

Weaknesses

- CWE-284
- CWE-862

Credits

VashuVats Reporter