Entity Access Password - migration and generalization of the Protected Node module

By @Florent_Torre - Drupal France Marseille meetup February 2023

DrupalCon

WARM-UP TOUR

JANVIER - SEPTEMBRE 2023

SMILE

Entity Access Password - migration and generalization of the Protected Node module

By @Florent_Torre - Drupal France Marseille meetup February 2023

SMILE

1. Context
2. Features
3. Architecture
4. Problems encountered
5. Data migration
6. To go beyond
7. Demo

CONTEXT

Protected Node

LMU

Methodology

SMILE

I.T IS OPEN

Context:

Protected Node

- Old module, created in 2007 on Drupal 5!
- Ported in Drupal 6 and then in development version for Drupal 7.
- One of the first modules I contributed to, starting on november 2013 with my mentor izus.

- The main goal of the module is to grant access to nodes given a password the user has to enter in a dedicated form when trying to access a protected node.
- Then around this comes other features more or less exotic like the password fork feature:
  - Given an URL like /protected-nodes?protected_pages=<nid1>,<nid2>,...&back=<url>
  - Depending on the entered password, the user will be redirected to one of the nodes.

Context:

Protected Node

- The main alternative module of Protected Node is Protected Pages.
- The key difference between the two modules is:
  - In Protected Node, the password is associated to a node (or its bundle or a global password)
  - In Protected Pages, the password is associated to a path.

- So with Protected Pages you can also add a password to pages that are not nodes or even content entities like a View page or a path handled by a custom controller.
- But you can’t administer your password directly on your entity form.

Context:

LMU

- End of 2021, I had been contacted by the Ludwig-Maximilians-Universität München, a university of Munich, which was willing to sponsorize the port of Protected Node for Drupal 9.
- In Germany, there is a high level of privacy, especially for children, teenagers and students.
- Use case: as the university provides to their students video recordings of their lessons, the students present in the classroom (may) appear on the videos, so the recording should only be accessible to the appropriate students.

Context:

LMU

University customizations:

- Creation of a system to grant access to user:
  - A professor can enter the students of its course, so those students do not have to enter any password, access is already granted.
- Creation of a mechanism of access inheritance:
  - Based on taxonomy terms, if given access to a taxonomy term, it grants access to all the associated nodes. For example tagging all the nodes related to a specific course for a specific year.
- Store successful password submissions in the database, not in the session.
  - So a user no more have to re-enter the password.
  - And the password of a node can be changed without having to communicate the new password to people who already got the access.
- Bypass password access for certains roles based on taxonomy

Context:

Methodology

Constraint: 20 work days to do the port!

1. Evaluation of the current features, which ones will be kept
2. Evaluation of the LMU features, which ones are enough generic and implementable in the available time.
3. Checking Protected Node issue queue for bugs or legitimate feature requests impossible to do with the current architecture of Protected Node to avoid being blocked again.

FEATURES

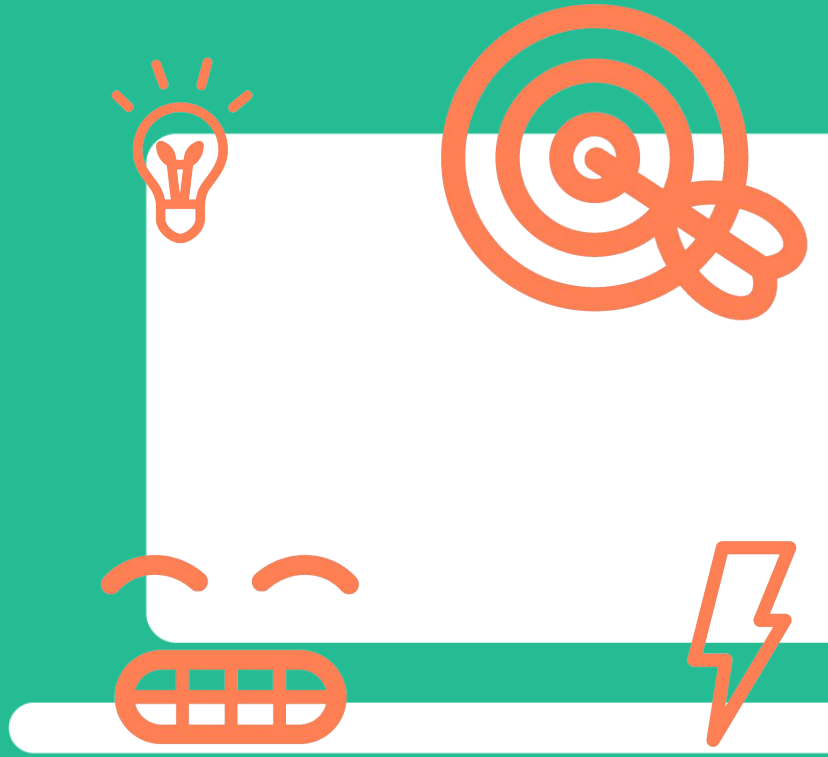
Database table

Global configuration

Per bundle configuration

Other features

LMU features



SMILE

I.T IS OPEN

Features

From custom database table to field storage

Database table field	Field type column
nid	**removed** (automatically handled by the field API)
protected_node_is_protected	is_protected
protected_node_show_title	show_title
protected_node_hint	hint
protected_node_passwd	password
protected_node_passwd_changed	**removed**
protected_node_emails	**removed**

Features

Global configuration

Before	After
Reports	**Removed**
Global password	Preserved
Password inheritance/behavior (global / bundle / entity)	Field instance setting
Show password strength	**Removed** (force enabled)
Show node titles by default	Field widget setting
Allow author to enter a password hint	Field widget setting
Protected Node email support	**Removed**
Generate a random password if necessary	Field widget setting
Protected Node Form: • Always add a cancel link • Other settings	• **Removed** • Field label / Field formatter / view mode usage
Display Suite view modes	Field instance setting
Protected Node actions: • Clear sessions • Other actions	• **Removed** • Use Views Bulk Edit

Features

Per bundle configuration

Before	After
Protected mode for nodes of this type	Field instance default value or field widget setting
A default (global) password for nodes of this type	Field instance setting
How to show the protected node fieldset	Field widget setting

Features

Other features

Before	After
Private attachments support	Preserved with support of Webform submissions by default
Token support	Only one token remaining [entity:protected-label] (for the show title feature)
Flood support	Preserved (using the user flood settings)
Autocomplete disabled	Preserved (by default with Drupal core)
Fork	**Removed**
Rules submodule	**Removed**
Views submodule	**Removed** (no more needed as the module is field based)
Search support (hiding info on indexation)	**Removed** (you can configure your view modes as you want and index the fields you want with Search API, so you can index fields to have search results and choose how to display them in other view mode, protected or not)

Features

LMU features

Before	After
Access inheritance	To do in custom development
Store access in user data	Implemented
Allow to grant/revoke access	Implemented
Bypass password access for certains roles based on taxonomy	To do in custom development

ARCHITECTURE

Architecture:

Settings and password storage

To have a generic module, it should:

1. not be limited anymore to nodes but become usable for any content entities
2. provide great flexibility/granularity

=> this should be managed by a new field type.

That way, it benefits of:

- Field storage
  - No more need of a custom table to store the data
- Field settings and field widgets plugins
  - No more need to alter node type and node forms
- Field formatters plugins
  - Display of the password form
  - Views integration

Architecture:

Access management

A first draft had been proposed by a community member but based on the hook_entity_access.

Problems:

- The hook_entity_access prevents the access to the content, but then also content in lists will not be present, so how does the visitor can have links to point to the content?
- This supposes to provide the password form in a dedicated URL or another system.

We want the content to be displayed normally except for some selected view modes where the user has to enter the password to see the content normally also in those view modes.

Architecture:

Access management

The Display Suite module has a feature that allows to select a view mode to display a content in this view mode instead of the “full” view mode when displaying the full page.

Knowing that, I checked how it was implemented and found out the hook_entity_view_mode_alter which allows to programmatically change the view mode a content entity is going to be displayed into.

So the access check can happen in this hook, to display either the expected view mode or the ‘password_protected’ view mode.

Architecture:

Access checking

Regarding the limitations of the Protected Node module, the access checking part had to be highly flexible to have a clean default implementation and letting developers able to override it for custom needs.

Currently needing to have a check for access stored in session but also in user's data.

=> this should be done using services that can be stacked to not override each other in a cumulative way.

Architecture:

Access checking

I took inspiration from core Breadcrumb system:

- Service tags and service collector
  - Multiple services, only one is triggered depending on conditions. Or all are triggered.
  - Order of execution controlled by the weight of the services.

=> this results in:

- One service tag to validate the password, the first service to validate the password grants the access
  - That way it is possible for example to validate the password against anything like a regex
- One service tag to store that the access had been granted, all services can be triggered to store that it is ok.
- One service tag to check if the user has access, the first service to grant the access stops the verification process.

=> Password validation, access checking and access storage are decorrelated.

PROBLEMS ENCOUNTERED

SMILE

IT IS OPEN

Problems encountered:

Field API

Creating a new field type is time consuming. It depends on the complexity of your field type, its settings. It can become tedious.

Because the field widget you are going to create will also be used in the field settings to configure the default value.

So you have to check that your field widget works:

- On the entity form
- On the field settings form
- (On the Views Bulk Edit form)

Problems encountered:

Field API

Second difficulty proper to the module: It is handling passwords… and passwords are saved hashed, so not possible to retrieve the password when returning on the form and populate default values of form elements.

=> Need to check if empty or not to override already saved password.

Separate logic of handling values between:

- massageFormValues() method of the widget
- preSave() method of the field type

Problems encountered:

Form API

The password confirm form element is a composite form element. Composed of 2 password form elements.

On the settings form or on an entity form, sometimes we want to hide this form element, for example when wanting to generate a random password.

Form API states features allows (among other stuff) to hide form elements based on conditions, but there is a core bug on password element.

=> The solution was to wrap the password confirm element in a container and to apply the states on it.

Problems encountered:

Form API

When displaying the same form multiple times on the same page, even when having different form objects with different arguments value, when submitting a form, if there is an error, all the forms are marked in error.

This is due to a core bug which does not generate different form tokens and cache based on the arguments, only based on the form ID.

=> The solution was to have a form ID dependent of the entity the form was for, that way Drupal may distinguish the form that is submitted.

This required to declare the form has a service to be able to have the form object and manipulate it before passing it to the form builder service.

Problems encountered:

Cache

To preserve Drupal cache mechanism and avoid performance loss, a new cache context had been created,

It allows to check if it is the protected version of an entity that should be displayed for a view mode.

It has 2 values, 0 or 1, to avoid cache duplication.

The value is based on:

1. If the entity is password protected,
2. If the view mode being viewed is protected,
3. If the user has access to this entity.

That way even the protected version of an entity is cacheable.

Problems encountered:

Cache

But the problem of displaying a form is that it prevents anonymous cache (Page cache core module).

Fortunately Drupal has a mechanism to avoid this problem: the lazy builder

It allows to delay the rendering of an element of the page. Drupal in a first time put a placeholder instead and render that in a second time.

That way the page is cacheable, and when rendering only the placeholders are recomputed if needed. They also can benefit from the cache mechanism.

Problems encountered:

User data sub-module

The creation of the service to store the access in the user's data was straightforward.

It was the creation of the UI to administer those data that was long:

- Forms to view, grant or revoke access:
  - globally
  - per bundle
  - per entity
  - from the user profile
- Access to those forms:
  - routes
  - menu links
  - menu tasks

Problems encountered:

Show title feature

When preventing access to content, you may want to also avoid to display the title of the content which can contain private data.

And this… is a nightmare!

Because the title of a content is handled separately multiple times during a page rendering:

- Breadcrumb
- Entity template
  - not manageable in the view mode configuration, natively in the Twig template
- HTML title of the page / Metatag
- Page title block

Problems encountered:

Show title feature

Solutions:

- Breadcrumb
  - Often on projects additional contrib modules manipulating the breadcrumb are added so not possible to handle that in a generic way
  - Do in custom code!
- Entity template
  - Hook_preprocess_node
  - Hook_preprocess_taxonomy_term
  - No hook needed for media entity because no hardcoded label in Twig template \o/
  - For other entity types, implements the appropriate hooks
- HTML title of the page / Metatag
  - Hook_preprocess_html
  - If there is Metatag, use the provided token that will handle the logic of displaying the title or not
- Page title block
  - Hook_preprocess_page_title

DATA MIGRATION

SMILE

I.T IS OPEN

Data migration: General notes

There is no automatic migration path provided from Protected Node.

As a migration is frequently the moment used to reorganize content structure and as it can be used to cleanup content and provide new passwords, it has been decided to not spend too much time on this point.

But:

- Examples of how to do a migration had been implemented and documented.
- As an hashed D7 password is still valid on D9, a change had been done in Protected Node to use the same hash mechanism as user account passwords. So by keeping the hashed password intact in your migration you can “preserve” your passwords.

Data migration: LMU specificities

LMU had already implemented a D7/D9 migration with the constraint of preserving the node IDs to keep some relationships.

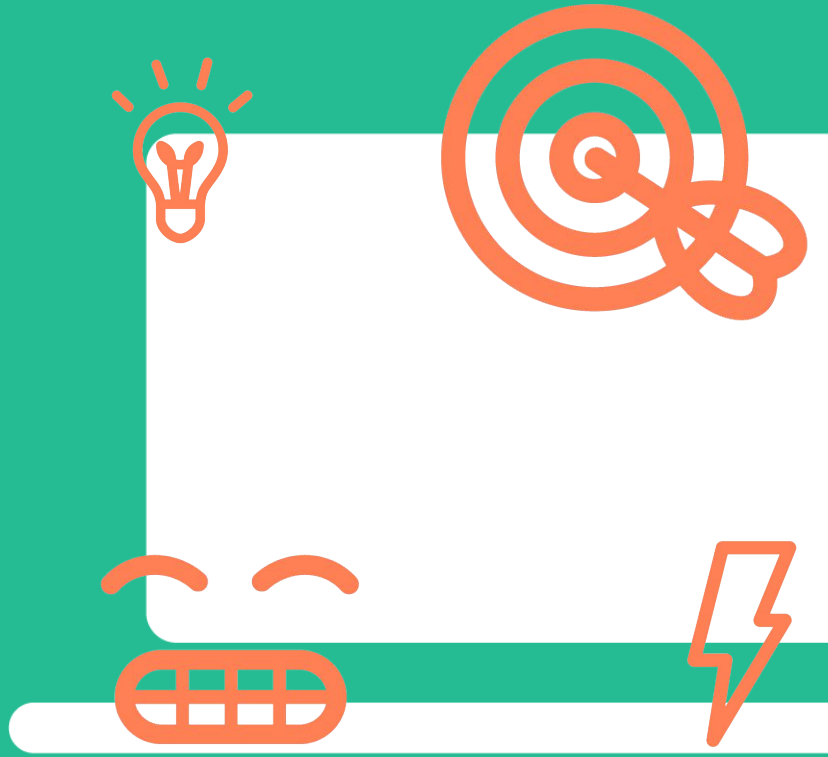
Therefore in the sub-module dedicated to user data storage, the storage keys are obtained through dedicated methods on the storage service.

That way by default, keys related to entities use the entity UUID, but it is possible for LMU to override this service and use the entity ID.

TO GO BEYOND

Next steps

Some thoughts



SMILE

I.T IS OPEN

To go beyond:

Next steps

None!

Entity Access Password is:

- Stable
- More generic
- Extensible
- Overridable

=> Maintenance only!

You can still provide features by providing MR with proper test coverage.

Or add features in separated modules extending Entity Access Password.

One drawback though: the hardcoded view mode machine name “password_protected”

A settings form should be introduced to get rid of this hardcoded configuration.

To go beyond:

Some thoughts

With the user data backend sub-module, you can grant/revoke access to users without having them to enter a password.

This raises the concern of a more wider access API around access control lists without being related to password.

Also another point to explore is if the granularity/inheritance (global/bundle/entity) password could have been handle with a plugin system, but during the implementation I had the feeling that it would have been over engineering.

DEMO

SMILE

I.T IS OPEN

Demo

1. Installation and configuration on a node
    a. Global settings
    b. Field settings
    c. Field widget settings
    d. View mode & Field formatter settings
2. Session backend
3. User data backend

Thank you for your attention!

Do you have any questions?

I.T IS OPEN