Commit 9a6e913

garvinhicking authored and **ohader** committed 2 hours ago

```
[SECURITY] Do not store password in serialized user settings

The new mechanism of using serialized JSON data for storing
backend user settings since TYPO3 14.2 has introduced a vulnerability
that stored the "password" and "verify password" input data
when changing a user's password inside the serialized user
settings representation.

The root cause was that the setup module controller merged virtual
fields stored in BE_USER->uc with regular fields of the database
table be_users into a single flat submission array. This made it
impossible to distinguish which fields belong to the database record
and which belong to the user settings, causing password fields to
be inadvertently written into the serialized user settings as well.

This is addressed by separating concerns in the backend form view
and how submitted data is handled in the controller. Two separate
partitions are introduced: 'be_users' for everything related to
the database table, and 'user_settings' for everything related to
BE_USER->uc. Field names in the virtual TCA form are now prefixed
accordingly (e.g. be_users__password, user_settings__titleLen) via
UserSettingsSchema::getTcaFieldName(), and the controller splits
submitted values per partition using extractPartitionData() before
processing them.

These passwords are no longer stored in the database columns
`be_users.uc` and `be_users.user_settings` anymore,
but may exist in database records during the period where
TYPO3 v14.2 was used.

An upgrade wizard has been added that will remove these credentials
from the serialized representation.

This upgrade wizard will detect possible records that contain
the string `"password` or `:"password` and then unserialize
the data, remove the two fields and re-serialize the data. It
is important to execute this wizard for safety. If the wizard
does not show up, no serialized credential data is found.

Resolves: #109585
Related: #109638
Related: #108880
Releases: main
```

```
Security-References: CVE-2026-6553
Security-Bulletin: TYPO3-CORE-SA-2026-005
Change-Id: I94bf02311a6cd642b540bb7e538eef5a5af9d6a5
Reviewed-on: https://review.typo3.org/c/Packages/TYPO3.CMS/+/93758
Reviewed-by: Oliver Hader <oliver.hader@typo3.org>
Reviewed-by: Garvin Hicking <garvin@hick.ing>
Tested-by: Benni Mack <benni@typo3.org>
Reviewed-by: Benni Mack <benni@typo3.org>
Tested-by: Garvin Hicking <garvin@hick.ing>
Tested-by: Oliver Hader <oliver.hader@typo3.org>
```

main · v14.3.0

1 parent d02c892 commit 9a6e913

6 files changed +580 -34 lines changed

Top

Filter files…

- typo3/sysext
  - backend
    - Classes
      - Controller
        - SetupModuleController.php
      - Form/FormDataProvider
        - UserSettingsDatabaseEditRow.php
      - Upgrades
        - UserSettingsScrubbingMigration.php
    - Tests/Functional/Upgrades
      - UserSettingsScrubbingMigrationTest.php
  - core
    - Classes/Authentication
      - UserSettingsSchema.php
    - Documentation/Changelog/14.3
      - Important-109585-SerializedCredentialDataInBeUsersDatabaseTable.rst

6 files changed +580 -34 lines changed

Search within code

...lasses/Controller/SetupModuleController.php

```
@@ -72,6 +72,8 @@
72 72  #[AsController]
73 73  class SetupModuleController
74 74  {
   75 +      protected const DISALLOWED_FIELD_NAMES = ['password', 'password2', 'email', 'realName', 'admin', 'avatar'];
   76 +
75 77       protected const PASSWORD_NOT_UPDATED = 0;
76 78       protected const PASSWORD_UPDATED = 1;
77 79       protected const PASSWORD_NOT_THE_SAME = 2;
@@ -272,46 +274,52 @@ protected function storeIncomingData(ServerRequestInterface $request): void
272 274          }
273 275
274 276          $formProtection = $this->formProtectionFactory->createFromRequest($request);
275    -        // First check if something is submitted in the data-array from POST vars
276    -        $d = $postData['data']['be_users_settings'][(int)$this->getBackendUser()->user['uid']] ?? null;
   277 +        // Separate submitted data into corresponding partitions
   278 +        $backendUserId = (int)$this->getBackendUser()->user['uid'];
   279 +        $beUsersSubmission = $this->extractPartitionData($postData['data']['be_users_settings'][$backendUserId] ?? [], 'be_users');
   280 +        $userSettingsSubmission = $this->extractPartitionData($postData['data']['be_users_settings'][$backendUserId] ?? [], 'user_settings');
277 281          $columns = $this->userSettingsSchema->getColumns();
278 282          $backendUser = $this->getBackendUser();
279 283          $beUserId = (int)$backendUser->user['uid'];
280 284          $storeRec = [];
281 285          $doSaveData = false;
282 286          $fieldList = $this->getFieldsFromShowItem();
283    -        if (is_array($d) && $formProtection->validateToken((string)($postData['formToken'] ?? ''), 'BE user setup', 'edit')) {
   287 +
   288 +        if ($beUsersSubmission !== []
```

```
                && $userSettingsSubmission !== []
                && $formProtection->validateToken((string)($postData['formToken'] ?? ''), 'BE user setup', 'edit')
            ) {
                // UC hashed before applying changes
                $save_before = md5(serialize($backendUser->uc));
                // PUT SETTINGS into the ->uc array:
                // Reload left frame when switching BE language
-               if (isset($d['lang']) && $d['lang'] !== $backendUser->user['lang']) {
+               if (isset($beUsersSubmission['lang']) && $beUsersSubmission['lang'] !== $backendUser->user['lang']) {
                    $this->languageUpdate = true;
                }
                // Reload pagetree if the title length is changed
-               if (isset($d['titleLen']) && $d['titleLen'] !== $backendUser->uc['titleLen']) {
+               if (isset($userSettingsSubmission['titleLen']) && $userSettingsSubmission['titleLen'] !== $backendUser->uc['titleLen']) {
                    $this->pagetreeNeedsRefresh = true;
                }
-               if (isset($d['colorScheme']) && $d['colorScheme'] !== ($backendUser->uc['colorScheme'] ?? null)) {
+               if (isset($userSettingsSubmission['colorScheme']) && $userSettingsSubmission['colorScheme'] !== ($backendUser->uc['colorScheme'] ?? null)) {
                    $this->colorSchemeChanged = true;
                }
-               if (isset($d['theme']) && $d['theme'] !== ($backendUser->uc['theme'] ?? null)) {
+               if (isset($userSettingsSubmission['theme']) && $userSettingsSubmission['theme'] !== ($backendUser->uc['theme'] ?? null)) {
                    $this->themeChanged = true;
                }
-               if (isset($d['backendTitleFormat']) && $d['backendTitleFormat'] !== ($backendUser->uc['backendTitleFormat'] ?? null)) {
+               if (isset($userSettingsSubmission['backendTitleFormat']) && $userSettingsSubmission['backendTitleFormat'] !== ($backendUser->uc['backendTitleFormat'] ?? null)) {
                    $this->backendTitleFormatChanged = true;
```

```
303  311                 }
304       -              if (isset($d['dateTimeFirstDayOfWeek']) &&
          $d['dateTimeFirstDayOfWeek'] !== ($backendUser->uc['dateTimeFirstDayOfWeek'] ??
          null)) {
     312  +              if (isset($userSettingsSubmission['dateTimeFirstDayOfWeek']) &&
          $userSettingsSubmission['dateTimeFirstDayOfWeek'] !== ($backendUser-
          >uc['dateTimeFirstDayOfWeek'] ?? null)) {
305  313                     $this->dateTimeFirstDayOfWeekChanged = true;
306  314                     $this->persistentUpdate[] = [
307  315                         'fieldName' => 'dateTimeFirstDayOfWeek',
308       -                     'value' => $d['dateTimeFirstDayOfWeek'],
     316  +                     'value' =>
          $userSettingsSubmission['dateTimeFirstDayOfWeek'],
309  317                     ];
310  318                 }
311  319                 // Options which should trigger direct JS persistent update,
          because
312  320                 // their new state needs to be available in JS components right
          away.
313  321                 foreach ($this->userSettingsSchema->getPersistentUpdateFieldNames()
          as $fieldName) {
314       -                 $fieldValue = ((int)($d[$fieldName] ?? 0)) ? 'on' : 0;
     322  +                 $fieldValue = ((int)($userSettingsSubmission[$fieldName] ?? 0))
          ? 1 : 0;
315  323                     if ($fieldValue !== ($backendUser->uc[$fieldName] ?? null)) {
316  324                         $this->persistentUpdate[] = [
317  325                             'fieldName' => $fieldName,
          @@ -320,45 +328,53 @@ protected function
          storeIncomingData(ServerRequestInterface $request): void
320  328                     }
321  329                 }
322  330
323       -             if ($d['setValuesToDefault'] ?? $postData['data']
          ['setValuesToDefault'] ?? false) {
     331  +             if ($postData['data']['setValuesToDefault'] ?? false) {
324  332                     // If every value should be default
325  333                     $backendUser->resetUC();
326  334                     $this->settingsAreResetToDefault = true;
327       -             } elseif ($d['save'] ?? $postData['data']['save'] ?? false) {
     335  +             } elseif ($postData['data']['save'] ?? false) {
```

```
328  336                     foreach ($columns as $field => $config) {
329  337                         if (!in_array($field, $fieldList, true)) {
330  338                             continue;
331  339                         }
     340 +                       // Skip any disallowed field name, not matter if it's in be_users or user_settings partition
     341 +                       if (in_array($field, self::DISALLOWED_FIELD_NAMES, true)) {
     342 +                           continue;
     343 +                       }
332  344                         $isBeUsersField = ($config['table'] ?? '') === 'be_users';
333  345                         $fieldType = $config['type'] ?? 'text';
334      -                       if ($isBeUsersField && !in_array($field, ['password', 'password2', 'email', 'realName', 'admin', 'avatar'], true)) {
335      -                           $submittedValue = $d[$field] ?? null;
     346 +                       if ($isBeUsersField) {
     347 +                           $submittedValue = $beUsersSubmission[$field] ?? null;
336  348                             if (!isset($config['access']) || ($this->checkAccess($config) && ($backendUser->user[$field] !== $submittedValue))) {
337  349                                 if ($fieldType === 'check') {
338      -                               $fieldValue = (int)($d[$field] ?? 0);
     350 +                               $fieldValue = (int)($submittedValue ?? 0);
339  351                                 } else {
340  352                                     $fieldValue = $submittedValue;
341  353                                 }
342  354                                 $storeRec['be_users'][$beUserId][$field] = $fieldValue;
343  355                                 $backendUser->user[$field] = $fieldValue;
344  356                             }
345      -                       }
346      -                       if ($fieldType === 'check') {
347      -                           $backendUser->uc[$field] = (int)($d[$field] ?? 0);
348  357                         } else {
349      -                           $backendUser->uc[$field] = htmlspecialchars($d[$field] ?? '');
     358 +                       if ($fieldType === 'check') {
     359 +                           $backendUser->uc[$field] = (int)($userSettingsSubmission[$field] ?? 0);
     360 +                       } else {
     361 +                           $backendUser->uc[$field] = htmlspecialchars($userSettingsSubmission[$field] ?? '');
```

```
                362 +                        }
350             363                      }
351             364                  }
352             365                  // Personal data for the users be_user-record (email, name,
                    password...)
353             366                  // If email and name is changed, set it in the users record:
354             -                    $be_user_data = $d;
                367 +                $be_user_data = $beUsersSubmission;
                368 +                // Temporarily hold `password2` for the hook to be able to
                    adjust the password
                369 +                $be_user_data['password2'] =
                    $userSettingsSubmission['password2'] ?? '';
355             370                  // Possibility to modify the transmitted values. Useful to do
                    transformations, like RSA password decryption
356             371                  foreach ($GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']
                    ['ext/setup/mod/index.php']['modifyUserDataBeforeSave'] ?? [] as $function) {
357             372                      $params = ['be_user_data' => &$be_user_data];
358             373                      GeneralUtility::callUserFunction($function, $params,
                    $this);
359             374                  }
360             375                  $this->passwordIsSubmitted = (string)($be_user_data['password']
                    ?? '') !== '';
361             376                  $passwordIsConfirmed = $this->passwordIsSubmitted &&
                    $be_user_data['password'] === $be_user_data['password2'];
                377 +                unset($be_user_data['password2']);
362             378
363             379                  // Validate password against password policy
364             380                  $contextData = new ContextData(
                    @@ -408,6 +424,10 @@ protected function
                    storeIncomingData(ServerRequestInterface $request): void
408             424
409             425                  $doSaveData = true;
410             426              }
                427 +            // Explicitly unset disallowed field names
                428 +            foreach (self::DISALLOWED_FIELD_NAMES as $disallowedFieldName) {
                429 +                unset($backendUser->uc[$disallowedFieldName]);
                430 +            }
411             431              // Inserts the overriding values.
412             432              $backendUser->overrideUC();
413             433              $save_after = md5(serialize($backendUser->uc));
```

```
@@ -472,7 +492,7 @@ protected function getFieldsFromShowItem(): array
472 492         {
473 493             // just keep field names, filter out control sequences
474 494             $tcaFieldNames = array_keys($this->userSettingsSchema->getColumns());
475     -           $allowedFields = GeneralUtility::trimExplode(',', $this->userSettingsSchema->getTcaShowitem(), true);
    495 +           $allowedFields = GeneralUtility::trimExplode(',', $this->userSettingsSchema->getRawShowitem(), true);
476 496             $allowedFields = array_map($this->extractShowitemFieldName(...), $allowedFields);
477 497             $allowedFields = array_filter($allowedFields, static fn(string $field): bool => in_array($field, $tcaFieldNames, true));
478 498             $backendUser = $this->getBackendUser();
@@ -661,6 +681,25 @@ protected function addFlashMessages(ModuleTemplate $view): void
661 681             }
662 682         }
663 683
    684 +     /**
    685 +      * @param array<string, scalar> $data
    686 +      * @return array<string, scalar>
    687 +      */
    688 +     protected function extractPartitionData(array $data, string $partition): array
    689 +     {
    690 +         $partitionData = [];
    691 +         $prefix = $partition . '__';
    692 +         $length = strlen($prefix);
    693 +         foreach ($data as $key => $value) {
    694 +             if (!str_starts_with($key, $prefix)) {
    695 +                 continue;
    696 +             }
    697 +             $normalizedKey = substr($key, $length);
    698 +             $partitionData[$normalizedKey] = $value;
    699 +         }
    700 +         return $partitionData;
    701 +     }
    702 +
664 703         protected function getBackendUser(): BackendUserAuthentication
```

```
665  704         {
666  705             return $GLOBALS['BE_USER'];
```

...ataProvider/UserSettingsDatabaseEditRow.php

```
@@ -34,6 +34,8 @@
34  34   */
35  35  readonly class UserSettingsDatabaseEditRow implements FormDataProviderInterface
36  36  {
    37  +     public function __construct(private UserSettingsSchema $userSettingsSchema) {}
    38  +
37  39      public function addData(array $result): array
38  40      {
39  41          if ($result['command'] !== 'edit' || $result['tableName'] !== 'be_users_settings') {
@@ -43,26 +45,31 @@ public function addData(array $result): array
43  45          $backendUser = $this->getBackendUser();
44  46          $userSettings = $backendUser->getUserSettings()->toArray();
45  47
46      -         // Set most rows from the current user
47      -         $userSettingsSchema = GeneralUtility::makeInstance(UserSettingsSchema::class);
    48  +         $userSettingsColumns = $this->userSettingsSchema->getColumns();
    49  +         $jsonFieldSettingKeys = $this->userSettingsSchema->getJsonFieldSettingKeys();
48  50          // Also provide direct access to be_users fields that are shown in the form
49  51          // These are needed for fields with inheritFromParent=true
50      -         foreach ($userSettingsSchema->getColumns() as $column => $config) {
    52  +         foreach ($userSettingsColumns as $column => $config) {
    53  +             $partitionedColumnName = $this->userSettingsSchema->getTcaFieldName($column);
51  54              if (isset($backendUser->user[$column])) {
52      -                 $result['databaseRow'][$column] = $backendUser->user[$column];
    55  +                 $result['databaseRow'][$partitionedColumnName] = $backendUser->user[$column];
53  56              } elseif (isset($userSettings[$column])) {
54      -                 $result['databaseRow'][$column] = $userSettings[$column];
```

```
                57 +                $result['databaseRow'][$partitionedColumnName] =
                   $userSettings[$column];
55              58             }
56              59         }
57              60         // Set the uid from the current user
58              61         $result['databaseRow']['uid'] = (int)$backendUser->user['uid'];
59              62         $result['databaseRow']['pid'] = 0;
60              -          $result['databaseRow']['password'] = $backendUser->user['password'] ??
                   '';
61              -          $result['databaseRow']['password2'] = $backendUser->user['password'] ??
                   '';
62              -          $result['databaseRow']['avatar'] = $this-
                   >getAvatarFileUid((int)$backendUser->user['uid']);
                63 +       // Fill in random to passwords to avoid FormEngine issuing the required
                   field error
                64 +       $randomPassword = bin2hex(random_bytes(20));
                65 +       $passwordFieldName = $this->userSettingsSchema-
                   >getTcaFieldName('password');
                66 +       $result['databaseRow'][$passwordFieldName] = $randomPassword;
                67 +       $passwordConfirmationFieldName = $this->userSettingsSchema-
                   >getTcaFieldName('password2');
                68 +       $result['databaseRow'][$passwordConfirmationFieldName] =
                   $randomPassword;
                69 +       // Forward the avatar FAL id
                70 +       $avatarFieldName = $this->userSettingsSchema->getTcaFieldName('avatar');
                71 +       $result['databaseRow'][$avatarFieldName] = $this-
                   >getAvatarFileUid((int)$backendUser->user['uid']);
63              72
64              -          // Load user settings from uc array
65              -          $result['databaseRow']['user_settings'] = $backendUser-
                   >getUserSettings()->toArray();
66              73         return $result;
67              74     }
68              75
```

...Upgrades/UserSettingsScrubbingMigration.php

```
                ...  @@ -0,0 +1,168 @@
                 1 + <?php
                 2 +
```

```
3  + declare(strict_types=1);
4  +
5  + /*
6  +  * This file is part of the TYPO3 CMS project.
7  +  *
8  +  * It is free software; you can redistribute it and/or modify it under
9  +  * the terms of the GNU General Public License, either version 2
10 +  * of the License, or any later version.
11 +  *
12 +  * For the full copyright and license information, please read the
13 +  * LICENSE.txt file that was distributed with this source code.
14 +  *
15 +  * The TYPO3 project - inspiring people to share!
16 +  */
17 +
18 + namespace TYPO3\CMS\Backend\Upgrades;
19 +
20 + use Doctrine\DBAL\Result;
21 + use Doctrine\DBAL\Types\Type;
22 + use Doctrine\DBAL\Types\Types;
23 + use TYPO3\CMS\Core\Attribute\UpgradeWizard;
24 + use TYPO3\CMS\Core\Database\Connection;
25 + use TYPO3\CMS\Core\Database\ConnectionPool;
26 + use TYPO3\CMS\Core\Upgrades\DatabaseUpdatedPrerequisite;
27 + use TYPO3\CMS\Core\Upgrades\UpgradeWizardInterface;
28 +
29 + /**
30 +  * Removes confidential data "password" and "password2" from user settings.
31 +  *
32 +  * @since 14.3
33 +  * @internal This class is only meant to be used within EXT:backend and is not
     part of the TYPO3 Core API.
34 +  */
35 + #[UpgradeWizard('setup_userSettingsScrubbingMigration')]
36 + final readonly class UserSettingsScrubbingMigration implements
     UpgradeWizardInterface
37 + {
38 +     public function __construct(
39 +         private ConnectionPool $connectionPool,
40 +     ) {}
```

```
41  +
42  +     public function getTitle(): string
43  +     {
44  +         return 'Scrub user settings to remove confidential credential data from
      serialized data (uc and user_settings)';
45  +     }
46  +
47  +     public function getDescription(): string
48  +     {
49  +         return 'Evaluates all uc and user_settings profile data in the
      "be_users" database table, removes "password" and "password2" field contents.';
50  +     }
51  +
52  +     public function getPrerequisites(): array
53  +     {
54  +         return [DatabaseUpdatedPrerequisite::class];
55  +     }
56  +
57  +     public function updateNecessary(): bool
58  +     {
59  +         return $this->hasRecordsToMigrate();
60  +     }
61  +
62  +     public function executeUpdate(): bool
63  +     {
64  +         $connection = $this->connectionPool->getConnectionForTable('be_users');
65  +         $hasFailures = false;
66  +
67  +         $result = $this->getRecordsToMigrate();
68  +         while ($record = $result->fetchAssociative()) {
69  +             try {
70  +                 // Redact "user_settings" (may fall back to "uc" for data)
71  +                 $userSettings = $record['user_settings'];
72  +                 if (!is_string($userSettings)) {
73  +                     continue;
74  +                 }
75  +                 $userSettings = json_decode(json: $userSettings, flags:
      JSON_THROW_ON_ERROR | JSON_OBJECT_AS_ARRAY);
76  +                 if (is_string($userSettings)) {
```

```
77  +                    // https://review.typo3.org/c/Packages/TYPO3.CMS/+/93395 introduced a double `json_encode()` issue,
78  +                    // which is resolved here as well. The fallback remains in place in case the generic cleanup upgrade
79  +                    // wizard from https://review.typo3.org/c/Packages/TYPO3.CMS/+/93726 has not been executed yet,
80  +                    // ensuring affected records are cleaned up during this step.
81  +                    $userSettings = json_decode(json: $userSettings, flags: JSON_THROW_ON_ERROR | JSON_OBJECT_AS_ARRAY);
82  +                }
83  +                if (!is_array($userSettings)) {
84  +                    $hasFailures = true;
85  +                    continue;
86  +                }
87  +                // Remove invalid entries.
88  +                unset($userSettings['password'], $userSettings['password2']);
89  +
90  +                $uc = @unserialize($record['uc'], ['allowed_classes' => false]);
91  +                // Update "uc", only if unserializable - redact two keys, only if set.
92  +                if (is_array($uc)) {
93  +                    // Remove invalid entries.
94  +                    unset($uc['password'], $uc['password2']);
95  +                    $connection->update(
96  +                        'be_users',
97  +                        [
98  +                            'uc' => serialize($uc),
99  +                            // The array must be passed directly to prevent double JSON encoding.
100 +                            // See: https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293
101 +                            'user_settings' => $userSettings,
102 +                        ],
103 +                        [
104 +                            'uid' => (int)$record['uid'],
105 +                        ],
106 +                        [
```

```
107 +                            'uc' => Connection::PARAM_LOB,
108 +                            // @todo This behavior cannot be modified yet; the
    array value must be passed directly
109 +                            //       until
    https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293 is merged,
110 +                            //       otherwise the value will be JSON-encoded
    twice.
111 +                            'user_settings' => Type::getType(Types::JSON),
112 +                        ],
113 +                    );
114 +                } else {
115 +                    // No "uc" serialization, only update user_settings.
116 +                    $connection->update(
117 +                        'be_users',
118 +                        [
119 +                            'user_settings' => $userSettings,
120 +                        ],
121 +                        ['uid' => (int)$record['uid']],
122 +                        [
123 +                            // @todo This behavior cannot be modified yet; the
    array value must be passed directly
124 +                            //       until
    https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293 is merged,
125 +                            //       otherwise the value will be JSON-encoded
    twice.
126 +                            'user_settings' => Type::getType(Types::JSON),
127 +                        ],
128 +                    );
129 +                }
130 +            } catch (\Throwable) {
131 +                $hasFailures = true;
132 +            }
133 +        }
134 +
135 +        return !$hasFailures;
136 +    }
137 +
138 +    private function hasRecordsToMigrate(): bool
139 +    {
```

```
140 +         $queryBuilder = $this->connectionPool->getQueryBuilderForTable('be_users');
141 +         return (bool)$queryBuilder
142 +             ->count('uid')
143 +             ->from('be_users')
144 +             ->where(
145 +                 $queryBuilder->expr()->or(
146 +                     $queryBuilder->expr()->like('uc', $queryBuilder->createNamedParameter('%' . $queryBuilder->escapeLikeWildcards(':"password') . '%')),
147 +                     $queryBuilder->expr()->like('user_settings', $queryBuilder->createNamedParameter('%' . $queryBuilder->escapeLikeWildcards('"password') . '%')),
148 +                 ),
149 +             )
150 +             ->executeQuery()
151 +             ->fetchOne();
152 +     }
153 +
154 +     private function getRecordsToMigrate(): Result
155 +     {
156 +         $queryBuilder = $this->connectionPool->getQueryBuilderForTable('be_users');
157 +         return $queryBuilder
158 +             ->select('uid', 'uc', 'user_settings')
159 +             ->from('be_users')
160 +             ->where(
161 +                 $queryBuilder->expr()->or(
162 +                     $queryBuilder->expr()->like('uc', $queryBuilder->createNamedParameter('%' . $queryBuilder->escapeLikeWildcards(':"password') . '%')),
163 +                     $queryBuilder->expr()->like('user_settings', $queryBuilder->createNamedParameter('%' . $queryBuilder->escapeLikeWildcards('"password') . '%')),
164 +                 ),
165 +             )
166 +             ->executeQuery();
167 +     }
168 + }
```

...ades/UserSettingsScrubbingMigrationTest.php

```
@@ -0,0 +1,251 @@
+ <?php
+
+ declare(strict_types=1);
+
+ /*
+  * This file is part of the TYPO3 CMS project.
+  *
+  * It is free software; you can redistribute it and/or modify it under
+  * the terms of the GNU General Public License, either version 2
+  * of the License, or any later version.
+  *
+  * For the full copyright and license information, please read the
+  * LICENSE.txt file that was distributed with this source code.
+  *
+  * The TYPO3 project - inspiring people to share!
+  */
+
+ namespace TYPO3\CMS\Backend\Tests\Functional\Upgrades;
+
+ use Doctrine\DBAL\Types\Type;
+ use Doctrine\DBAL\Types\Types;
+ use PHPUnit\Framework\Attributes\Test;
+ use TYPO3\CMS\Backend\Upgrades\UserSettingsScrubbingMigration;
+ use TYPO3\CMS\Core\Database\Connection;
+ use TYPO3\CMS\Core\Database\ConnectionPool;
+ use TYPO3\TestingFramework\Core\Functional\FunctionalTestCase;
+
+ final class UserSettingsScrubbingMigrationTest extends FunctionalTestCase
+ {
+     #[Test]
+     public function updateNecessaryReturnsTrueWhenDoubleJsonEncodedUserSettingsWithInvalidFieldsExists(): void
+     {
+         $connection = $this->get(ConnectionPool::class)->getConnectionForTable('be_users');
+
```

```
35 +         // Create a user with uc but without user_settings
36 +         $connection->insert(
37 +             'be_users',
38 +             [
39 +                 'username' => 'testuser',
40 +                 'uc' => serialize(['colorScheme' => 'dark', 'titleLen' => 50, 'password' => 'some-plain-password']),
41 +                 // The array would normally be passed directly to prevent double JSON encoding;
42 +                 // however, in this case the desired test context requires it to be intentionally
43 +                 // JSON-encoded here.
44 +                 // See: https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293
45 +                 'user_settings' => json_encode(['colorScheme' => 'dark', 'titleLen' => 50, 'password' => 'some-plain-password']),
46 +             ],
47 +             [
48 +                 'uc' => Connection::PARAM_LOB,
49 +                 // @todo This behavior cannot be modified yet; the array value must be passed directly
50 +                 //       until https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293 is merged,
51 +                 //       otherwise the value will be JSON-encoded twice.
52 +                 'user_settings' => Type::getType(Types::JSON),
53 +             ],
54 +         );
55 +
56 +         $subject = $this->get(UserSettingsScrubbingMigration::class);
57 +         self::assertTrue($subject->updateNecessary());
58 +     }
59 +
60 +     #[Test]
61 +     public function updateNecessaryReturnsTrueWhenSingleJsonEncodedUserSettingsWithInvalidFieldsExists(): void
62 +     {
63 +         $connection = $this->get(ConnectionPool::class)->getConnectionForTable('be_users');
64 +
65 +         // Create a user with uc but without user_settings
```

```
66  +         $connection->insert(
67  +             'be_users',
68  +             [
69  +                 'username' => 'testuser',
70  +                 'uc' => serialize(['colorScheme' => 'dark', 'titleLen' => 50,
    'password' => 'some-plain-password']),
71  +                 // The array must be passed directly to prevent double JSON
    encoding.
72  +                 // See: https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293
73  +                 'user_settings' => ['colorScheme' => 'dark', 'titleLen' => 50,
    'password' => 'some-plain-password'],
74  +             ],
75  +             [
76  +                 'uc' => Connection::PARAM_LOB,
77  +                 // @todo This behavior cannot be modified yet; the array value
    must be passed directly
78  +                 //       until
    https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293 is merged,
79  +                 //       otherwise the value will be JSON-encoded twice.
80  +                 'user_settings' => Type::getType(Types::JSON),
81  +             ],
82  +         );
83  +
84  +         $subject = $this->get(UserSettingsScrubbingMigration::class);
85  +         self::assertTrue($subject->updateNecessary());
86  +     }
87  +
88  +     #[Test]
89  +     public function
    updateNecessaryReturnsFalseWhenDoubleJsonEncodedUserSettingsWithoutInvalidField
    sExists(): void
90  +     {
91  +         $connection = $this->get(ConnectionPool::class)-
    >getConnectionForTable('be_users');
92  +
93  +         // Create a user with uc but without user_settings
94  +         $connection->insert(
95  +             'be_users',
96  +             [
97  +                 'username' => 'testuser',
```

```
98  +                'uc' => serialize(['colorScheme' => 'dark', 'titleLen' => 50]),
99  +                // The array would normally be passed directly to prevent double JSON encoding;
100 +                // however, in this case the desired test context requires it to be intentionally
101 +                // JSON-encoded here.
102 +                // See: https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293
103 +                'user_settings' => json_encode(['colorScheme' => 'dark', 'titleLen' => 50]),
104 +            ],
105 +            [
106 +                'uc' => Connection::PARAM_LOB,
107 +                // @todo This behavior cannot be modified yet; the array value must be passed directly
108 +                //       until https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293 is merged,
109 +                //       otherwise the value will be JSON-encoded twice.
110 +                'user_settings' => Type::getType(Types::JSON),
111 +            ],
112 +        );
113 +
114 +        $subject = $this->get(UserSettingsScrubbingMigration::class);
115 +        self::assertFalse($subject->updateNecessary());
116 +    }
117 +
118 +    #[Test]
119 +    public function updateNecessaryReturnsFalseWhenSingleJsonEncodedUserSettingsWithoutInvalidFieldsExists(): void
120 +    {
121 +        $connection = $this->get(ConnectionPool::class)->getConnectionForTable('be_users');
122 +
123 +        // Create a user with uc but without user_settings
124 +        $connection->insert(
125 +            'be_users',
126 +            [
127 +                'username' => 'testuser',
128 +                'uc' => serialize(['colorScheme' => 'dark', 'titleLen' => 50]),
```

```
129 +                // The array must be passed directly to prevent double JSON encoding.
130 +                // See: https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293
131 +                'user_settings' => ['colorScheme' => 'dark', 'titleLen' => 50],
132 +            ],
133 +            [
134 +                'uc' => Connection::PARAM_LOB,
135 +                // @todo This behavior cannot be modified yet; the array value must be passed directly
136 +                //       until https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293 is merged,
137 +                //       otherwise the value will be JSON-encoded twice.
138 +                'user_settings' => Type::getType(Types::JSON),
139 +            ],
140 +        );
141 +
142 +        $subject = $this->get(UserSettingsScrubbingMigration::class);
143 +        self::assertFalse($subject->updateNecessary());
144 +    }
145 +
146 +    #[Test]
147 +    public function executeUpdateMigratesDoubleJsonEncodedUserSettingsRemovingInvalidValues(): void
148 +    {
149 +        $connection = $this->get(ConnectionPool::class)->getConnectionForTable('be_users');
150 +
151 +        // Create a user with uc but without user_settings
152 +        $connection->insert(
153 +            'be_users',
154 +            [
155 +                'username' => 'testuser',
156 +                'uc' => serialize(['colorScheme' => 'dark', 'titleLen' => 50]),
157 +                // The array would normally be passed directly to prevent double JSON encoding;
158 +                // however, in this case the desired test context requires it to be intentionally
159 +                // JSON-encoded here.
160 +                // See: https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293
```

```
161 +                'user_settings' => json_encode(['colorScheme' => 'dark', 'titleLen' => 50, 'password' => 'some-plain-password']),
162 +            ],
163 +            [
164 +                'uc' => Connection::PARAM_LOB,
165 +                // @todo This behavior cannot be modified yet; the array value must be passed directly
166 +                //       until https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293 is merged,
167 +                //       otherwise the value will be JSON-encoded twice.
168 +                'user_settings' => Type::getType(Types::JSON),
169 +            ],
170 +        );
171 +        $userId = (int)$connection->lastInsertId();
172 +
173 +        // Check user settings is double json_encoded
174 +        $row = $connection->select(['user_settings'], 'be_users', ['uid' => $userId])->fetchAssociative();
175 +        self::assertNotEmpty($row['user_settings'], 'user_settings should not be empty');
176 +        $rawValue = $row['user_settings'];
177 +        self::assertTrue(str_starts_with($rawValue, '"{'));
178 +        self::assertTrue(str_ends_with($rawValue, '}"'));
179 +        $decoded = json_decode(json: $rawValue, flags: JSON_THROW_ON_ERROR | JSON_OBJECT_AS_ARRAY);
180 +        self::assertIsString($decoded);
181 +        $decoded = json_decode(json: $decoded, flags: JSON_THROW_ON_ERROR | JSON_OBJECT_AS_ARRAY);
182 +        self::assertIsArray($decoded);
183 +        self::assertArrayHasKey('password', $decoded);
184 +
185 +        $subject = $this->get(UserSettingsScrubbingMigration::class);
186 +        self::assertTrue($subject->updateNecessary());
187 +        self::assertTrue($subject->executeUpdate());
188 +
189 +        // Check user settings is now single json_encoded
190 +        $row = $connection->select(['user_settings'], 'be_users', ['uid' => $userId])->fetchAssociative();
191 +        self::assertNotEmpty($row['user_settings'], 'user_settings should not be empty');
```

```
192 +         $rawValue = $row['user_settings'];
193 +         self::assertFalse(str_starts_with($rawValue, '"{'));
194 +         self::assertFalse(str_ends_with($rawValue, '}"'));
195 +         $decoded = json_decode(json: $rawValue, flags: JSON_THROW_ON_ERROR |
    JSON_OBJECT_AS_ARRAY);
196 +         self::assertIsArray($decoded);
197 +         self::assertArrayNotHasKey('password', $decoded);
198 +
199 +         self::assertFalse($subject->updateNecessary());
200 +     }
201 +
202 +     #[Test]
203 +     public function
    executeUpdateMigratesSingleJsonEncodedUserSettingsRemovingInvalidValues(): void
204 +     {
205 +         $connection = $this->get(ConnectionPool::class)-
    >getConnectionForTable('be_users');
206 +
207 +         // Create a user with uc but without user_settings
208 +         $connection->insert(
209 +             'be_users',
210 +             [
211 +                 'username' => 'testuser',
212 +                 'uc' => serialize(['colorScheme' => 'dark', 'titleLen' => 50]),
213 +                 // The array must be passed directly to prevent double JSON
    encoding.
214 +                 // See: https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293
215 +                 'user_settings' => ['colorScheme' => 'dark', 'titleLen' => 50,
    'password' => 'some-plain-password'],
216 +             ],
217 +             [
218 +                 'uc' => Connection::PARAM_LOB,
219 +                 // @todo This behavior cannot be modified yet; the array value
    must be passed directly
220 +                 //       until
    https://review.typo3.org/c/Packages/TYPO3.CMS/+/89293 is merged,
221 +                 //       otherwise the value will be JSON-encoded twice.
222 +                 'user_settings' => Type::getType(Types::JSON),
223 +             ],
224 +         );
```

```
+         $userId = (int)$connection->lastInsertId();
+
+         // Check user settings is double json_encoded
+         $row = $connection->select(['user_settings'], 'be_users', ['uid' => $userId])->fetchAssociative();
+         self::assertNotEmpty($row['user_settings'], 'user_settings should not be empty');
+         $rawValue = $row['user_settings'];
+         self::assertTrue(str_starts_with($rawValue, '{'));
+         self::assertTrue(str_ends_with($rawValue, '}'));
+         $decoded = json_decode(json: $rawValue, flags: JSON_THROW_ON_ERROR | JSON_OBJECT_AS_ARRAY);
+         self::assertIsArray($decoded);
+         self::assertArrayHasKey('password', $decoded);
+
+         $subject = $this->get(UserSettingsScrubbingMigration::class);
+         self::assertTrue($subject->updateNecessary());
+         self::assertTrue($subject->executeUpdate());
+
+         // Check user settings is now single json_encoded
+         $row = $connection->select(['user_settings'], 'be_users', ['uid' => $userId])->fetchAssociative();
+         self::assertNotEmpty($row['user_settings'], 'user_settings should not be empty');
+         $rawValue = $row['user_settings'];
+         self::assertFalse(str_starts_with($rawValue, '"{'));
+         self::assertFalse(str_ends_with($rawValue, '}"'));
+         $decoded = json_decode(json: $rawValue, flags: JSON_THROW_ON_ERROR | JSON_OBJECT_AS_ARRAY);
+         self::assertIsArray($decoded);
+         self::assertArrayNotHasKey('password', $decoded);
+     }
+ }
```

...asses/Authentication/UserSettingsSchema.php

```
@@ -17,6 +17,8 @@

namespace TYPO3\CMS\Core\Authentication;

+ use TYPO3\CMS\Core\Utility\GeneralUtility;
```

```
   21 +
20 22    /**
21 23     * Provides unified access to backend user settings configuration.
22 24     *
   ↕      @@ -81,7 +83,8 @@ public function getTca(): array
81 83        {
82 84            $columns = $GLOBALS['TCA']['be_users']['columns']['user_settings']
   ['columns'] ?? [];
83 85            foreach ($columns as $fieldName => $columnConfig) {
84    -             $columns[$fieldName] = $this->resolveInheritFromParent($fieldName,
   $columnConfig);
   86 +             $partitionedFieldName = $this->getTcaFieldName($fieldName);
   87 +             $columns[$partitionedFieldName] = $this-
   >resolveInheritFromParent($fieldName, $columnConfig);
85 88            }
86 89
87 90            return [
   ↕      @@ -100,9 +103,53 @@ public function getTca(): array
100 103       }
101 104
102 105       /**
103   -      * Get the showitem string (merged from TCA and legacy global).
   106 +      * Returns a partitioned field name for use in TCA.
   107 +      * - e.g. `be_users__password`, reflecting `be_users` values
   108 +      * - e.g. `user_settings__titleLen`, reflecting JSON values
   109 +      */
   110 +     public function getTcaFieldName(string $fieldName): string
   111 +     {
   112 +         $configuration = $this->getColumn($fieldName);
   113 +         $partition = ($configuration['table'] ?? null) === 'be_users' ?
   'be_users' : 'user_settings';
   114 +         return $partition . '__' . $fieldName;
   115 +     }
   116 +
   117 +     private function resolveTcaFieldName(string $fieldName, bool $strict =
   true): string
   118 +     {
   119 +         $configuration = $this->getColumn($fieldName);
   120 +         if ($configuration !== null) {
```

```
+             $partition = ($configuration['table'] ?? null) === 'be_users' ? 'be_users' : 'user_settings';
+             return $partition . '__' . $fieldName;
+         }
+         if (!$strict) {
+             return $fieldName;
+         }
+         throw new \LogicException(
+             sprintf(
+                 'Column "%s" not found in UserSettingsSchema',
+                 $fieldName
+             ),
+             1776439141
+         );
+     }
+
+     /**
+      * Get the partitioned showitem string to be used as virtual TCA.
      */
     public function getTcaShowitem(): string
+     {
+         $items = GeneralUtility::trimExplode(',', $this->getRawShowitem(), true);
+         $items = array_map(
+             fn(string $fieldName): string => $this->resolveTcaFieldName($fieldName, false),
+             $items
+         );
+         return implode(',', $items);
+     }
+
+     /**
+      * Get the raw showitem string (merged from TCA and legacy global).
+      */
+     public function getRawShowitem(): string
     {
         $tcaShowitem = trim($GLOBALS['TCA']['be_users']['columns']['user_settings']['showitem'] ?? '');
         // @deprecated since TYPO3 v14, remove in TYPO3 v15
```

...zedCredentialDataInBeUsersDatabaseTable.rst

```
@@ -0,0 +1,34 @@
+ ..  include:: /Includes.rst.txt
+
+ ..  _important-109585-1776329549:
+
+ ====================================================================
+ Important: #109585 - Serialized Credential Data in be_users settings
+ ====================================================================
+
+ See :issue:`109585`
+
+ Description
+ ===========
+
+ The new mechanism of using serialized JSON data for storing
+ backend user settings since TYPO3 14.2 has introduced a vulnerability
+ that stored the "password" and "verify password" input data
+ when changing a user's password inside the serialized user
+ settings representation.
+
+ These passwords are no longer stored in the database columns
+ :sql:`be_users.uc` and :sql:`be_users.user_settings` anymore,
+ but may exist in database records during the period where
+ TYPO3 v14.2 was used.
+
+ An upgrade wizard has been added that will remove these credentials
+ from the serialized representation.
+
+ This upgrade wizard will detect possible records that contain
+ the string `"password` or `:"password` and then unserialize
+ the data, remove the two fields and re-serialize the data. It
+ is important to execute this wizard for safety. If the wizard
+ does not show up, no serialized credential data is found.
+
+ ..  index:: Backend, PHP-API, ext:backend, NotScanned
```

Comments 0

Please sign in to comment.