SYNACKTIV
DIGITAL SECURITY

Timing attack user enumeration in GLPI <= 9.4.1.1 CVE-2019-10233

Security advisory

2019-04-23

Julien SZLAMOWICZ
Damien PICARD

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 allows users to specify the hashing algorithm used in the *rememberme* feature and doesn't follow the same codepath whether the user exists or not, leading to a timing attack. This vulnerability can be used to determine precisely if a user identifier exists whithin the database.

Affected versions

The following versions are known to be affected:

- Branch 9.4: < 9.4.1.1
- Branch 9.3: < 9.3.4

Timeline

Date	Action
2019-02-25	Advisory sent to *GLPI Project* (glpi-security@ow2.org)
2019-03-15	Vendor releases the version 9.4.1.1 resolving the issue for the branch 9.4.X
2019-04-11	Vendor releases the version 9.3.4 resolving the issue for the branch 9.3.X

1 https://glpi-project.org/

Technical description and proof-of-concept

Authentication is required to access the features of the application using a set of credentials (username and password). However, bypassing the authentication is possible. An arbitrary identity can therefore be obtained.

The *rememberme* feature in GLPI allows users to provide an arbitrary hash type and uses the PHP *password_verify* function only if the user exists.

Indeed, the function *getAlternateAuthSystemsUserLogin*, at *inc/auth.class.php*, retrieves the *rememberme* cookie if provided by the user. This cookie needs to meet the following structure:

```
<session_cookie_name>_rememberme=[<user_id>,<personal_token_hash>]
```

The different values are:

- *session_cookie_name*: the actual session cookie name which follows the basic structure:

```
glpi_<session_identifier>
```

- *user_id*: the user identifier to impersonate
- *personal_token_hash*: the hash of the secret used for rememberme

For recently connected users, a value is stored in the *personal_token* column in the database. A hash of this value is expected here.

Then the following code snippet is called:

```
if ($CFG_GLPI["login_remember_time"]) {
   $data = json_decode($_COOKIE[$cookie_name], true);
   if (count($data) === 2) {
      list ($cookie_id, $cookie_token) = $data;
```

After ensuring that the *login_remember_time* is set in the configuration (which is the case by default) the application uses *json_decode* on the provided cookie.

In the next lines, the application verifies that the obtained array has 2 elements and stores these elements in 2 different variables:

- *cookie_id*
- *cookie_token*

Let's consider the next code snippet:

```
$user = new User();
$user->getFromDB($cookie_id);
$token = $user->getAuthToken();
if ($token !== false && Auth::checkPassword($token, $cookie_token)) {
   $this->user->fields['name'] = $user->fields['name'];
   return true;
} else {
   $this->addToError(__("Invalid cookie data"));
}
```

The call to the function *getAuthToken* returns *false* only if the user does not exist. In this case, the *if* clause condition is *false* and an error is immediately thrown. Otherwise, the *Auth::checkPassword* function is called (*inc/auth.class.php*):

```
static function checkPassword($pass, $hash) {
  $tmp = password_get_info($hash);
  if (isset($tmp['algo']) && $tmp['algo']) {
     $ok = password_verify($pass, $hash);
  } else if (strlen($hash)==32) {
     $ok = md5($pass) == $hash;
  } else if (strlen($hash)==40) {
     $ok = sha1($pass) == $hash;
  } else {
     $salt = substr($hash, 0, 8);
     $ok = ($salt.sha1($salt.$pass) == $hash);
  }
  return $ok;
}
```

In order to make sure the timing varies enough to notice a difference, the most convenient option is to enter the first *if* clause. It is then possible to pass a difficult *hash* with a high number of iterations such as *bcrypt* for instance:

```
glpi_<session_identifier>_rememberme=["<user_id>","$2y$16$DAA86CD8biMGj0aRIjxBy.EdjNEP2EQSgTgDmI4rgzVJN0JVCFa5a"]
```

Doing so, the PHP *password__get_info* function recognizes the provided hash type and thus the server will have to compute a *bcrypt* hash with a difficulty set to 16 which is long enough to observe a difference with the precedent case. As a consequence, if the server answers immediately, it means that the provided user identifier does not exist.

For instance, if the user doesn't exist (id=10), the servers answers immediately:

```
$ time curl -s -k -b 'glpi_0212c7703564e40d8dded2a951a0791f=ohgq697ua0dn7rpva69v5afdd5; glpi_0212c7703564e40d8dded2a951a0791f_rememberme=[10,"$2y$16$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy"]' 'http://glpi.lab.synacktiv.com/front/login.php'
real 0m0,097s
```

However, when the user exists (id=2), the server computes the hash and takes time before answering:

```
$ time curl -s -k -b 'glpi_0212c7703564e40d8dded2a951a0791f=ohgq697ua0dn7rpva69v5afdd5; glpi_0212c7703564e40d8dded2a951a0791f_rememberme=[2,"$2y$16$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy"]' 'http://glpi.lab.synacktiv.com/front/login.php'
real 0m3,135s
```