AKERVA

SECURITY ADVISORY

Dolibarr

Cross-Site Scripting (Authenticated)

BELABED Skander

26/05/2023

CVE-2023-38888

AKERVA

RENNES – PARIS

Siège social - 37 Boulevard Solférino - 35000 RENNES

Tel +33(0)1 53 76 86 35 | E-mail contact@akerva.com

Table of contents

SUMMARY **3**
Context 3
Description 3
Products and versions affected 3
Impact 3
Mitigations 3
Disclosure timeline 3
TECHNICAL DETAILS **4**
Vulnerability Details 4
Proof of Concept (PoC) 5
Risk Characterization 6
References 6
ABOUT AKERVA **7**
Who are we? 7
Join us 7
Contact 7

SUMMARY

Context

Product description:

> Dolibarr ERP CRM is an open source, free software package for companies of any size, foundations or freelancers. It includes different features for enterprise resource planning (ERP) and customer relationship management (CRM) but also other features for different activities.

Description

The user inputs submitted over the API are not controlled, giving the possibility to inject malicious data and perform “Cross-Site Scripting”.

Products and versions affected

Affected products:

- Dolibarr 17.0.1 and earlier

Impact

Any user having access to the API and the permission of writing or editing any object can inject HTML or JavaScript content that can be executed in the browser of the other users.

Mitigations

Upgrade to the latest version of the product.

Disclosure timeline

DATE	EVENT
26/05/2023	Initial discovery.
07/07/2023	Initial contact with security@dolibarr.org
21/07/2023	Vulnerability acknowledged by Dolibarr’s team
18/08/2023	Fix published by Dolibarr’s team
18/09/2023	Public disclosure

TECHNICAL DETAILS

Vulnerability Details

One of the defense mechanisms used by Dolibarr against "Cross-Site Scripting" (XSS) attacks is to pass each user input submitted over the interface to a control function present in *htdocs/main.inc.php* following this workflow:

1. Passing POST values to *analyseVarsForSqlAndScriptsInjection* function if the "scan POST data" option is enabled.

```php
// Sanity check on POST
if (!defined('NOSCANPOSTFORINJECTION')) {
    analyseVarsForSqlAndScriptsInjection($_POST, 0);
}
```

FIGURE 1: SANITY CHECK ON POST DATA (HTDOCS/MAIN.INC.PHP).

2. Extracting each variable of the array and pass it to *testSqlAndScriptInject* function.

```php
/**
 * Security: WAF layer for SQL Injection and XSS Injection (scripts) protection (Filters on GET, POST, PHP_SELF).
 * Warning: Such a protection can't be enough. It is not reliable as it will always be possible to bypass this. Good protection can
 * only be guaranted by escaping data during output.
 *
 * @param       string      $val        Brute value found into $_GET, $_POST or PHP_SELF
 * @param       string      $type       0=POST, 1=GET, 2=PHP_SELF, 3=GET without sql reserved keywords (the less tolerant test)
 * @return      int                     >0 if there is an injection, 0 if none
 */
function testSqlAndScriptInject($val, $type)
```

FIGURE 2: TESTSQLANDSCRIPTINJECT FUNCTION PRESENT IN HTDOCS/MAIN.INC.PHP.

This last function (*testSqlAndScriptInject*) performs multiple checks as: decoding strings in order to prevent some forms of bypass or comparing strings to a tag/event handlers blacklist.

```
// Decode string first because a lot of things are obfuscated by encoding or multiple encoding.
    // So <svg o&#110;load='console.log(&quot;123&quot;)' become <svg onload='console.log(&quot;123&quot;)'
    // So "&colon;&apos;" become ":'" (due to ENT_HTML5)
    // So "&Tab;&NewLine;" become ""
    // So "&lpar;&rpar;" become "()"

    // Loop to decode until no more things to decode.
    //print "before decoding $val\n";
    do {
        $oldval = $val;
        $val = html_entity_decode($val, ENT_QUOTES | ENT_HTML5);    // Decode '&colon;', '&apos;', '&Tab;', '&NewLine', ...
        // Sometimes we have entities without the ; at end so html_entity_decode does not work but entities is still interpreted by browser.
        $val = preg_replace_callback('/&#(x?[0-9][0-9a-f]+;?)/i', function ($m) {
            // Decode '&#110;', ...
            return realCharForNumericEntities($m); }, $val);

        // We clean html comments because some hacks try to obfuscate evil strings by inserting HTML comments. Example: on<!-- -->error=alert(1)
        $val = preg_replace('/<!--[^>]*-->/', '', $val);
        $val = preg_replace('/[\r\n\t]/', '', $val);
    } while ($oldval != $val);
    //print "type = ".$type." after decoding: ".$val."\n";

```

FIGURE 3: DECODING PHASE IN TESTSQLANDSCRIPTINJECT FUNCTION.

```
// For XSS Injection done by closing textarea to execute content into a textarea field
    $inj += preg_match('/<\/textarea/i', $val);
    // For XSS injection done by adding javascript with script
    // This is all cases a browser consider text is javascript:
    // When it found '<script', 'javascript:', '<style', 'onload\s=' on body tag, '="&' on a tag size with old browsers
    // All examples on page: http://ha.ckers.org/xss.html#XSScalc
    // More on https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
    $inj += preg_match('/<audio/i', $val);
    $inj += preg_match('/<embed/i', $val);
    $inj += preg_match('/<iframe/i', $val);
    $inj += preg_match('/<object/i', $val);
    $inj += preg_match('/<script/i', $val);
    $inj += preg_match('/Set\.constructor/i', $val); // ECMA script 6
    if (!defined('NOSTYLECHECK')) {
        $inj += preg_match('/<style/i', $val);
    }
    $inj += preg_match('/base\s+href/si', $val);
    $inj += preg_match('/=data:/si', $val);
    // List of dom events is on https://www.w3schools.com/jsref/dom_obj_event.asp and https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers
    $inj += preg_match('/on(mouse|drag|key|load|touch|pointer|select|transition)([a-zl*)\s*=/i', $val); // onmousexxx can be set on img or any html tag like <img title='...' onmouseover=alert(1)>
    $inj += preg_match('/on(abort|afterprint|animation|auxclick|beforecopy|beforecut|beforeprint|beforeunload|blur|cancel|canplay|canplaythrough|change|click|close|contextmenu|cuechange|copy|cut)\s*=/i', $val);
    $inj += preg_match('/on(dblclick|drop|durationchange|emptied|end|ended|error|focus|focusin|focusout|formdata|gotpointercapture|hashchange|input|invalid)\s*=/i', $val);
    $inj += preg_match('/on(lostpointercapture|offline|online|pagehide|pageshow)\s*=/i', $val);
    $inj += preg_match('/on(paste|pause|play|playing|progress|ratechange|reset|resize|scroll|search|seeked|seeking|show|stalled|start|submit|suspend)\s*=/i', $val);
    $inj += preg_match('/on(timeupdate|toggle|unload|volumechange|waiting|wheel)\s*=/i', $val);
    // More not into the previous list
    $inj += preg_match('/on(repeat|begin|finish|beforeinput)\s*=/i', $val);

    // We refuse html into html because some hacks try to obfuscate evil strings by inserting HTML into HTML. Example: <img on<a>error=alert(1) to bypass test on onerror
    $tmpval = preg_replace('/<[^<]+>/', '', $val);
    // List of dom events is on https://www.w3schools.com/jsref/dom_obj_event.asp and https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers
    $inj += preg_match('/on(mouse|drag|key|load|touch|pointer|select|transition)([a-zl*)\s*=/i', $tmpval); // onmousexxx can be set on img or any html tag like <img title='...' onmouseover=alert(1)>
    $inj += preg_match('/on(abort|afterprint|animation|auxclick|beforecopy|beforecut|beforeprint|beforeunload|blur|cancel|canplay|canplaythrough|change|click|close|contextmenu|cuechange|copy|cut)\s*=/i', $tmpval);
    $inj += preg_match('/on(dblclick|drop|durationchange|emptied|end|ended|error|focus|focusin|focusout|formdata|gotpointercapture|hashchange|input|invalid)\s*=/i', $tmpval);
    $inj += preg_match('/on(lostpointercapture|offline|online|pagehide|pageshow)\s*=/i', $tmpval);
    $inj += preg_match('/on(paste|pause|play|playing|progress|ratechange|reset|resize|scroll|search|seeked|seeking|show|stalled|start|submit|suspend)\s*=/i', $tmpval);
    $inj += preg_match('/on(timeupdate|toggle|unload|volumechange|waiting|wheel)\s*=/i', $tmpval);
    // More not into the previous list
    $inj += preg_match('/on(repeat|begin|finish|beforeinput)\s*=/i', $tmpval);

    //$inj += preg_match('/on[A-Z][a-z]+\*=/', $val);   // To lock event handlers onAbort(), ...
    $inj += preg_match('/&#58;|&#0000058|&#x3A/i', $val); // refused string ':' encoded (no reason to have it encoded) to lock 'javascript:...'
    $inj += preg_match('/j\s*a\s*v\s*a\s*s\s*c\s*r\s*i\s*p\s*t\s*:/i', $val);
    $inj += preg_match('/vbscript\s*:/i', $val);
    // For XSS Injection done by adding javascript closing html tags like with onmousemove, etc... (closing a src or href tag with not cleaned param)
    if ($type == 1 || $type == 3) {
        $val = str_replace('enclosure="', 'enclosure=X', $val); // We accept enclosure=" for the export/import module
        $inj += preg_match('/"/i', $val); // We refused " in GET parameters value.
    }
    if ($type == 2) {
        $inj += preg_match('/[:;"\'<>\?\(\){}\$%]/', $val); // PHP SELF is a file system (or url path without parameters). It can contains spaces.
    }
```

FIGURE 4: BLACKLIST COMPARISON PHASE IN TESTSQLANDSCRIPTINJECT.

But in case of an API call (POST/PUT) the data is not correctly parsed / controlled, and as a result, it is possible to inject malicious data.

Proof of Concept (PoC)

Let's assume that the REST API module is enabled in Dolibarr and we have an account which has the permission of creating/editing any object type (in our case our user has the permission to create third parties). We just need to get the API key linked to our account and send an authenticated API request containing our malicious code in one of the body fields:

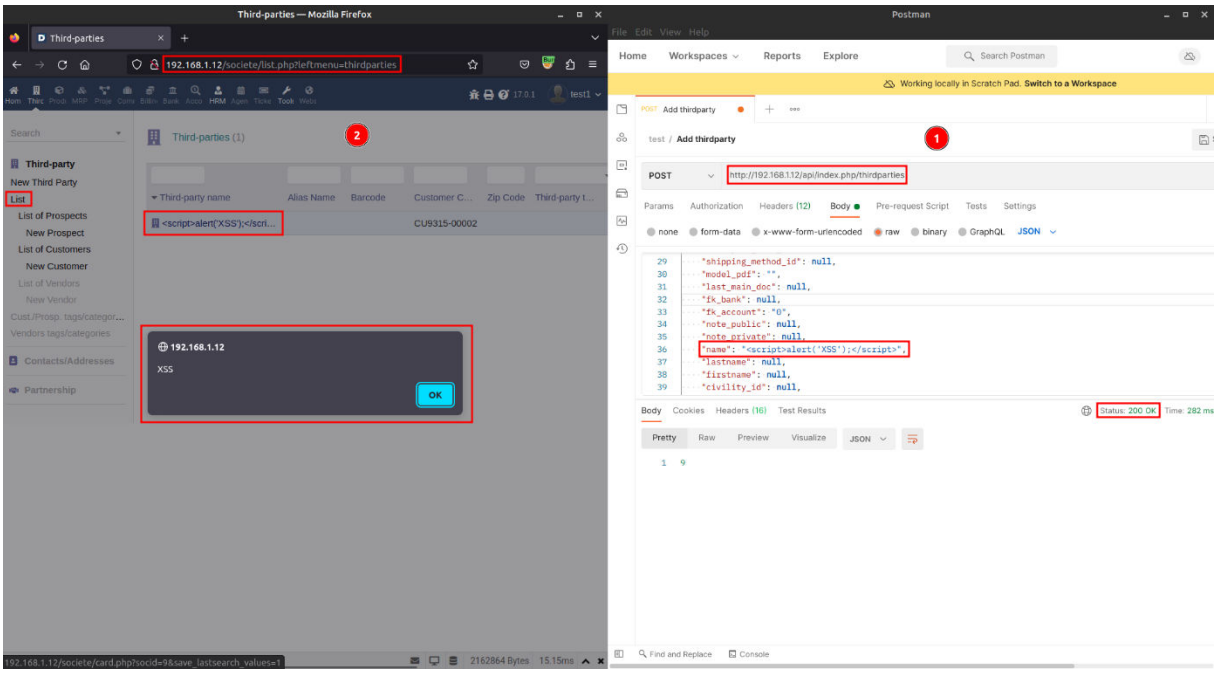

FIGURE 5: XSS EXPLOITATION STEPS.

Risk Characterization

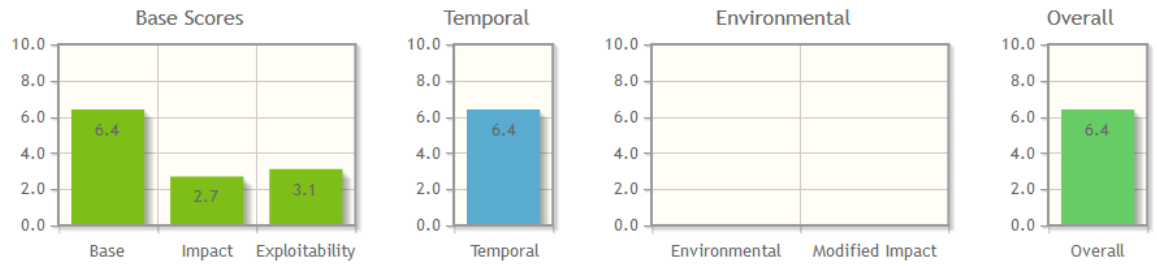

FIGURE 6: CVSS SCORING.

CVSS v3.1 – Base Score			
Attack Vector (AV)	Network (N)	**Scope (S)**	Changed (C)
Attack Complexity (AC)	Low (L)	**Confidentiality (C)**	Low (L)
Privileges Required (PR)	Low (L)	**Integrity (I)**	Low (L)
User Interaction (UI)	None (N)	**Availability (A)**	None (N)
CVSS v3.1 – Temporal Score			
Exploit Code Maturity (E)	High (H)		
Remediation Level (RL)	Not Defined (X)		
Report Confidence (RC)	Confirmed (C)		

References

- Dolibarr, Wikipedia https://www.citethisforme.com/cite/sources/websiteautociteeval

ABOUT AKERVA

Who are we?

Founded in 2013, *Akerva* is a consulting firm specialized in CyberSecurity and Risk Management. Our *Offensive Technology team (OffTech)* work for our customers to provide them with security assessments through offensive and technical audits in order to identify credible real world compromission scenarios and business risk. Missions such as application or network penetration testing, red team engagements or phishing and social engineering campaigns are complemented by R&D and vulnerability research in our dedicated lab to maintain the highest technical proficiency for our team.

Join us

Want to be part of the adventure? Join our team of experts by sending your application: https://akerva.com/jobs/

Contact

- **Website:** https://akerva.com
- **Blog:** https://akerva.com/blog/
- **Email:** contact@akerva.com
- **LinkedIn:** https://fr.linkedin.com/company/akerva
- **Twitter:** https://twitter.com/Akerva_FR