MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

CSIRT.SK

When Escaping Lies: From SQL Modes to RCE in OTRS

CVE-2026-48188

A configuration-induced SQL injection leading to authentication bypass and remote code execution

Created by: CSIRT.SK
Ministerstvo investícií, regionálneho rozvoja a informatizácie SR
Pribinova 25
811 09 Bratislava

Date of creation: June 2026

TLP: Clear

Introduction

Modern applications often rely on database escaping mechanisms they assume are universal. But what happens when the database silently changes the rules?

In this article, we explore a subtle but impactful vulnerability in OTRS deployments using MySQL or MariaDB. When a specific SQL mode (`NO_BACKSLASH_ESCAPES`) is enabled, assumptions about string escaping break down — turning otherwise safe queries into SQL injection primitives.

This misalignment between application logic and database behavior can ultimately lead to **authentication bypass and remote code execution**.

The Core Issue

At the heart of this vulnerability is a mismatch between how the application escapes input and how the database interprets it.

OTRS relies on backslash-based escaping to safely include user input in SQL queries. Under normal MySQL behavior, this works as expected.

However, when the SQL mode `NO_BACKSLASH_ESCAPES` is enabled, **backslashes lose their special meaning**.

Example

```sql
-- Application assumes this is safe
SELECT * FROM users WHERE login = 'admin\' OR 1=1 --';
```

Under normal conditions, the `\'` escapes the quote.

But with `NO_BACKSLASH_ESCAPES` enabled:

- The backslash is treated as a literal character
- The quote is no longer escaped
- The query structure breaks
- Injection becomes possible

Key Insight

- The application believes it is escaping input. The database disagrees.

Root Cause

The root cause is not simply improper escaping — but reliance on **assumed escaping semantics** that are not guaranteed across database configurations.

Root Cause in Code

The issue originates from how OTRS performs manual SQL escaping in its database abstraction layer.

In the MySQL driver implementation, user-controlled input is sanitized by replacing certain characters, including single quotes:

```perl
sub Quote {
    my ( $Self, $Text, $Type ) = @_;

    if ( defined ${$Text} ) {
        if ( $Self->{'DB::QuoteBack'} ) {
            ${$Text} =~ s/\\/$Self->{'DB::QuoteBack'}\\/g;
        }
        if ( $Self->{'DB::QuoteSingle'} ) {
            ${$Text} =~ s/'/$Self->{'DB::QuoteSingle'}'/g;
        }
        if ( $Self->{'DB::QuoteSemicolon'} ) {
            ${$Text} =~ s/;/$Self->{'DB::QuoteSemicolon'};/g;
        }
        if ( $Type && $Type eq 'Like' ) {
            if ( $Self->{'DB::QuoteUnderscoreStart'} || $Self->{'DB::QuoteUnderscoreEnd'} ) {
                ${$Text} =~ s/_/$Self->{'DB::QuoteUnderscoreStart'}_$Self->{'DB::QuoteUnderscoreEnd'}/g;
            }
        }
    }
    return $Text;
}
```

With configuration:

```perl
$Self->{'DB::QuoteSingle'} = '\\';
```

This results in single quotes being escaped as `' → \\'`.

Why This Breaks

This approach assumes that the database will interpret `` `\'` `` as an escaped quote.

However, when MySQL/MariaDB is running with `` `NO_BACKSLASH_ESCAPES` ``:

- Backslashes are treated as ordinary characters
- `` `\'` `` is no longer a valid escape sequence
- The quote terminates the string as-is

As a result, the escaping logic becomes ineffective, allowing injection.

➢ The application enforces escaping. The database silently disables it.

Note

While the code example above references the OTRS Community Edition, the same escaping logic and assumptions are present in the official OTRS codebase, as confirmed during coordinated disclosure.

Why This Is Subtle

This is not a typical SQL injection caused by missing sanitization.

Instead, it is:

- A **configuration-induced vulnerability**
- Triggered by a **database SQL mode**
- Caused by **invalid assumptions about escaping behavior**

This makes it particularly dangerous because:

- The application code may appear correct
- Security reviews may miss it
- It only manifests under specific environments

➢ Security assumptions don't fail loudly — they fail silently when environments change.

Attack Chain

While SQL injection alone is critical, the real impact comes from how it can be chained with existing functionality in OTRS.

SQL Injection

↓

Authentication Bypass

↓

Admin Access

↓

Package Installation

↓

Remote Code Execution

Relation to Prior Research

In previous research, I explored how OTRS administrative functionality can be abused to achieve remote code execution:

- When Admin Features Become RCE: A Case Study in OTRS Package Design\ https://habuon.github.io/2026/02/23/When-Admin-Features-Become-RCE-A-Case-Study-in-OTRS-Package-Design.html

That research assumed an attacker already had administrative access.

This new finding fundamentally changes the threat model:

- ➢ It provides a path to reach administrative functionality **without authentication**.

Proof of Concept (High-Level)

Goal

The goal of the proof of concept is to demonstrate that changing the SQL mode invalidates escaping assumptions and allows authentication bypass.

Environment

- OTRS with MySQL or MariaDB
- SQL mode: `NO_BACKSLASH_ESCAPES`

Core Idea

The injection relies on breaking out of a quoted string due to disabled backslash escaping.

Example input:

```
admin\' OR 1=1 --
```

Demonstration (Conceptual)

By submitting a crafted login value, the resulting SQL query evaluates to true, bypassing authentication checks.

No application errors or warnings are required — the behavior is entirely dependent on database interpretation.

Reproducing the Issue

! The following steps are intended for testing in a controlled environment only.

1. Environment Setup

The issue can be reproduced using a standard OTRS Community Edition Docker deployment with a MySQL backend.

2. Modify SQL Mode

Once the environment is running, modify the MySQL SQL mode to include `NO_BACKSLASH_ESCAPES`:

```
``` bash
docker exec -it <mysql_container_id> mysql -uroot -p -e "SET GLOBAL sql_mode =
CONCAT(@@sql_mode, ',NO_BACKSLASH_ESCAPES');"
```
```

In the default Docker setup, the root password is typically:

```
changeme
```

3. Triggering the Vulnerability

After modifying the SQL mode, the application's escaping logic becomes ineffective due to the change in how backslashes are interpreted.

At this point, authentication logic may be bypassed using crafted input that breaks out of the expected SQL query structure.

4. Proof of Concept Script

An accompanying Python script is provided to demonstrate the authentication bypass in a reproducible way.

The full PoC, including setup and automation, is available in the repository:

https://github.com/Habuon/CVE-2026-48188

Why This PoC Matters

Many developers would test this code and conclude it is safe.

The PoC demonstrates that correctness depends on database configuration — something often outside the direct control of the application.

This makes the issue particularly dangerous in real-world deployments.

Affected Configurations

This issue appears under the following conditions:

- OTRS with MySQL or MariaDB
- SQL mode including `` `NO_BACKSLASH_ESCAPES` ``

Impact

In affected environments, an attacker can:

- Perform SQL injection
- Bypass authentication
- Gain administrative access
- Execute arbitrary code on the system

In short:

- A remote unauthenticated attacker may achieve potential full system compromise.

Detection Considerations

Detecting this issue can be difficult:

- Authentication logs may only show successful logins
- SQL errors may not be generated
- Malicious input may appear as normal login attempts

Monitoring for anomalous authentication behavior is recommended.

Defensive Lessons

This vulnerability highlights several important lessons for secure development:

1. Never Rely on Escaping Alone

Escaping mechanisms are fragile and environment-dependent.

Use parameterized queries (prepared statements) instead.

2. Treat Configuration as Part of the Attack Surface

Security is not just about code.

Database settings, runtime flags, and environment differences can introduce vulnerabilities.

3. Test Across Different SQL Modes

Applications should be tested under varying database configurations, including:

- NO_BACKSLASH_ESCAPES
- Strict modes
- Compatibility modes

4. Validate Security Assumptions

If your application assumes a specific behavior:

- Document it
- Enforce it
- Or eliminate the dependency entirely

Disclosure

This issue was responsibly disclosed to the OTRS Product Security Team through a coordinated disclosure process.

Timeline

- **2026-03-06** Vulnerability identified during security research into OTRS authentication and database handling behavior.
- **2026-03-18** The issue was privately reported to the OTRS Security Team together with technical details and reproduction information.
- **2026-04-10** The vendor confirmed the vulnerability and verified the impact under affected MySQL/MariaDB configurations.
- **2026-05-21** The issue was assigned **CVE-2026-48188**.
- **2026-06-01** Coordinated public disclosure and advisory release.

Vendor Assessment

CVSS v4.0

Critical — **9.1**

CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N/AU:Y/R:U/V:D/RE:L/U:Amber

CVSS v3.1

Critical — **9.1**

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

Conclusion

This vulnerability is a reminder that security is not just about writing correct code — it's about ensuring that your assumptions hold in every environment.

This wasn't just a bug in code. It was a bug in assumptions.

When those assumptions fail, even well-written applications can become exploitable in unexpected ways.