MINISTERSTVO
INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA
A INFORMATIZÁCIE
SLOVENSKEJ REPUBLIKY

CSIRT.SK

When Admin Features Become RCE: A Case Study in OTRS Package Design

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

Date of creation: March 2026

TLP: Clear

Introduction

Most modern web applications operate under the assumption that no one is fully trusted, even administrators. However, some architectural decisions still grant admins capabilities that effectively blur the line between *administration* and *arbitrary code execution*.

In this post, I will analyze a design decision in OTRS: the ability for administrators to install extension packages (`.opm`) that can execute arbitrary Perl code during installation.

This is not a vulnerability in the classic sense. There is no authentication bypass, no injection flaw, and no sandbox escape.

However, it is a **dangerous design pattern** that turns *admin access* into *server command execution*, often without operators realizing the security implications.

You can view a proof-of-concept demonstrating this behavior on my GitHub repo: [OTRS PoC Repository] (https://github.com/Habuon/exploits/tree/main/otrs)

Background: OTRS Package Management

OTRS supports a package format (`.opm`) intended to extend the system with new functionality. These packages can define installation logic using `<CodeInstall>` blocks:

```xml
<CodeInstall><![CDATA[
   use strict;
   use warnings;
   # arbitrary Perl code
]]></CodeInstall>
```

This code is executed automatically when the package is installed by an administrator via the web interface.

From a functionality perspective, this makes sense:

- Packages need to modify configuration
- They may need to install files
- They may need to interact with the system

From a security perspective, however, this means:

➢ Installing a package = executing arbitrary server-side code.

Proof of Concept

A malicious package can embed OS command execution in its install routine:

```perl
my $content = `id`;
```

This command is executed as the user running the OTRS service.

With minimal code, it can:

- Execute arbitrary shell commands
- Write output to the webroot
- Install persistence
- Exfiltrate secrets
- Deploy backdoors

A Python PoC automates this process by:

1. Logging in as admin
2. Uploading a crafted `.opm` file
3. Triggering installation
4. Retrieving command output

You can view the full PoC on my GitHub repo here: [OTRS PoC Repository] (https://github.com/Habuon/exploits/tree/main/otrs)

No exploit chains are required. No memory corruption. No deserialization bug. Just a feature.

Why This Is Not a CVE

This behavior is:

- Authenticated
- Intended
- Part of the system design

Therefore, it does not qualify as a vulnerability by most definitions.

However, “not a vulnerability” does not mean “safe design.”

Security Impact

The design creates a collapse of privilege layers:

Layer	Intended
Web admin	Application control
System user	OS-level execution

By allowing arbitrary code execution inside a package install routine, these two layers become equivalent.

In practice, this means:

- Compromising OTRS admin = compromising the server.

This breaks a key security principle:

Administrative authority ≠ arbitrary code execution.

Comparison to Other Platforms

Modern plugin systems (WordPress, browsers, IDEs) increasingly:

- Sandbox extensions
- Restrict filesystem access
- Disallow raw code execution hooks
- Use declarative install steps

OTRS's model is closer to:

- "Run this Perl script as part of installation."

Which is powerful, but extremely dangerous.

Lessons Learned

1. **Signed or trusted extensions are not safe by default** — Trust should be minimized, not maximized.
2. **Admin panels should not be shell interfaces** — A web admin account should not implicitly grant OS command execution.
3. **Extension systems should be declarative, not imperative** — Installation should describe *what* to do, not *how to execute code*.
4. **Post-auth impact matters** — Many compromises begin with stolen credentials.

Defensive Recommendations

If you operate an OTRS instance:

- Restrict admin accounts strictly
- Treat admin compromise as full server compromise
- Do not expose OTRS admin interface to the public internet
- Use OS-level isolation (containers, chroot, SELinux, AppArmor)
- Monitor package installation activity
- Consider read-only filesystem for web process where possible

Conclusion

This case study is not about a vulnerability. It is about a design choice that creates a dangerous security boundary collapse.

When extension systems allow arbitrary code execution during installation, administrators are no longer just administrators — they become shell users.

This may be acceptable in controlled environments, but it should be an explicit and understood risk.

Sometimes the most dangerous bugs are not bugs at all — they are features.