Audit Report

High Severity Issues

H-01: Asset validation is not properly implemented in Module 5

Overview

In `main.sw`, **Module 5**, at **line 243**, a logic check ensures that the first output is strictly `FUEL_BASE_ASSET`:

```
assert(output_coin_asset_id(0).unwrap() == FUEL_BASE_ASSET);
```

However, at **line 241**, another assertion enforces that the second output must also be `FUEL_BASE_ASSET`:

```
assert(output_coin_asset_id(1).unwrap() == FUEL_BASE_ASSET);
```

This restriction mandates that both outputs must be the Fuel base asset, potentially blocking legitimate transactions that intend to transfer native assets instead.

Impact

- **Transaction Rejection:** If a user intends to transfer a native asset other than `FUEL_BASE_ASSET`, all the transaction fails

Recommendation

- Modify the assertion logic to correctly differentiate between **base asset** and **Assetid** transfers.

Fixed: Patch available here

H-02: Wallet Initialization Race Condition

Overview

The wallet is built on **Fuel**, which supports **parallel transaction execution**. However, a critical issue exists in the `initialize_wallet` function: **storage is updated after minting the nonce asset**. This allows two transactions (**Tx1** and **Tx2**) to be processed in parallel, both creating **identical nonce assets** with the same wallet address.

Vulnerable Code Snippet

```
fn initialize_wallet(
    master_addr: Address,
    owner_evm_addr: EvmAddress,
    initdata: InitData,
) -> EvmAddress {
    require(!_is_paused(), "Contract is paused");

    match initdata {
        InitData::InitModules(base_mods) => {
            require(storage.can_initialize.read(), "Wallet initialization is not

            let key = get_key1(owner_evm_addr, master_addr);

            require(
                !check_initialized(key),
                "Wallet already has Nonce, if error mint assets individually"
            );

            let (nonce_tfr_amt, nonce_assetid) = mint_nonce_asset(owner_evm_addr
            storage.v1_map.insert(key, nonce_assetid);
        }
    }
}
```

Exploit Scenario (Step-by-Step)

1. **Parallel Transactions Execution:**
    - An attacker sends **Tx1** and **Tx2** simultaneously, calling `initialize_wallet()`.
2. **Bypassing the Initialization Check:**

    - `check_initialized()` executes **before** storage updates, returning **false** for both transactions.
3. **Double Minting Occurs:**
    - **Tx1** mints a nonce asset.
    - **Tx2** mints another nonce asset before storage updates.

Impact

- **Duplicate Wallet Initializations:** Two separate nonce assets exist for the **same wallet address**.
- **Signature Replay Attacks:** This flaw impacts **Module 1 and Module 2**, enabling attackers to abuse wallet

Recommendation

- **Immediate Fix:** Update storage **before** minting the nonce asset.

Low Severity Issues

L-01: Zero Address Check Missing in Ownership Transfer

Overview

The `transfer_ownership()` function does not validate whether the new owner's address is a **zero address** before transferring ownership.

Vulnerable Code Snippet

```
#[storage(read, write)]
fn transfer_ownership(new_owner: Identity) {
    require_owner();
    storage.owner.write(State::Initialized(new_owner));
}
```

Impact

- If the ownership is transferred to a **zero address**, recovery is impossible.
- The contract becomes permanently inaccessible, leading to a loss of administrative control.

Recommendation

Fix required:

- Introduce a validation check preventing a zero address assignment.
- Implement event logging for ownership changes to enhance security monitoring.

L-02: Unused Variable `upperb_ovf`

Overview

The variable `upperb_ovf` is declared but never used, resulting in unnecessary computation and memory allocation.

Vulnerable Code Snippet

```
let upperb_ovf = is_overflow_u64(upper_bound_bn); //@audit unused variable

if actual_amount_bn < lower_bound_bn || actual_amount_bn > upper_bound_bn {
    return false;
}

return true;
```

Fix required:

- Remove the unused variable or integrate it properly within the function logic.

Informational Issues

I-01: Contract Pause Check Missing on Version Update

Overview

A function responsible for contract updates does not verify whether the contract is paused before executing an update.

Fix required:

- Implement a **pause check** before allowing a contract version update to proceed.