CAIRO SECURITY CLAN

NIMBORA PROTOCOL

SECURITY ASSESSMENT REPORT

APRIL 2024

Prepared for

NIMBORA PROTOCOL

Contents

1 About Cairo Security Clan

Cairo Security Clan is a leading force in the realm of blockchain security, dedicated to fortifying the foundations of the digital age. As pioneers in the field, we specialize in conducting meticulous smart contract security audits, ensuring the integrity and reliability of decentralized applications built on blockchain technology.

At Cairo Security Clan, we boast a multidisciplinary team of seasoned professionals proficient in blockchain security, cryptography, and software engineering. With a firm commitment to excellence, our experts delve into every aspect of the Web3 ecosystem, from foundational layer protocols to application-layer development. Our comprehensive suite of services encompasses smart contract audits, formal verification, and real-time monitoring, offering unparalleled protection against potential vulnerabilities.

Our team comprises industry veterans and scholars with extensive academic backgrounds and practical experience. Armed with advanced methodologies and cutting-edge tools, we scrutinize and analyze complex smart contracts with precision and rigor. Our track record speaks volumes, with a plethora of published research papers and citations, demonstrating our unwavering dedication to advancing the field of blockchain security.

At Cairo Security Clan, we prioritize collaboration and transparency, fostering meaningful partnerships with our clients. We believe in a customer-oriented approach, engaging stakeholders at every stage of the auditing process. By maintaining open lines of communication and soliciting client feedback, we ensure that our solutions are tailored to meet the unique needs and objectives of each project.

Beyond our core services, Cairo Security Clan is committed to driving innovation and shaping the future of blockchain technology. As active contributors to the ecosystem, we participate in the development of emerging technologies such as Starknet, leveraging our expertise to build robust infrastructure and tools. Through strategic guidance and support, we empower our partners to navigate the complexities of the blockchain landscape with confidence and clarity.

In summary, Cairo Security Clan stands at the forefront of blockchain security, blending technical prowess with a client-centric ethos to deliver unparalleled protection and peace of mind in an ever-evolving digital landscape. Join us in safeguarding the future of decentralized finance and digital assets with confidence and conviction.

2 Disclaimer

Disclaimer Limitations of this Audit:

This report is based solely on the materials and documentation provided by you to Cairo Security Clan for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented here may not be exhaustive and may not identify all potential vulnerabilities. Cairo Security Clan provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, occurs entirely at your own risk.

Inherent Risks of Blockchain Technology:

Blockchain technology remains in its developmental stage and is inherently susceptible to unknown risks and vulnerabilities. This review is specifically focused on the smart contract code and does not extend to the compiler layer, programming language elements beyond the reviewed code, or other potential security risks outside the code itself.

Report Purpose and Reliance:

This report should not be construed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. No third party should rely on this report for any purpose, including making investment or purchasing decisions.

Liability Disclaimer:

To the fullest extent permitted by law, Cairo Security Clan disclaims all liability associated with this report, its contents, and any related services and products arising from your use. This includes, but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services:

Cairo Security Clan does not warrant, endorse, guarantee, or assume responsibility for any products or services advertised by third parties within this report, nor for any open-source or third-party software, code, libraries, materials, or information linked to, referenced by, or accessible through this report, its content, and related services and products. This includes any hyperlinked websites, websites or applications appearing on advertisements, and Cairo Security Clan will not be responsible for monitoring any transactions between you and third-party providers. It is recommended that you exercise due diligence and caution when considering any third-party products or services, just as you would with any purchase or service through any medium.

Disclaimer of Advice:

FOR THE AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, ACCESS, AND/OR USE, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Executive Summary

This document presents the security review performed by Cairo Security Clan on the Nimbora protocol.

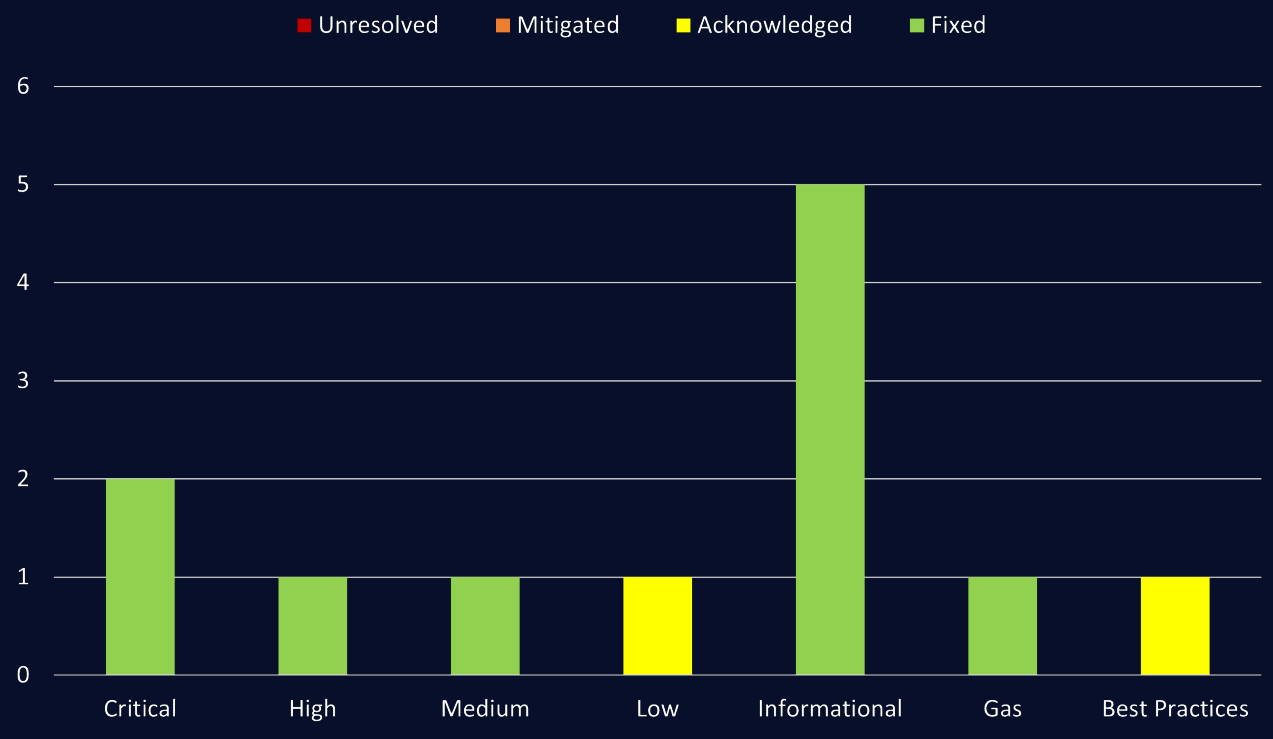
Nimbora is a Starknet-based cross-chain yield strategy platform that enables users to engage with Layer 1 protocols at a fraction of the cost. Powered by the innovative minds at the SpaceShard team, Nimbora leverages Starknet's Zero-Knowledge Rollups to provide cost-effective DeFi strategies without compromising security or trust. Learn more from Nimbora docs.

The audit was performed using

- manual analysis of the codebase,
- automated analysis tools,
- simulation of the smart contract,
- analysis of edge test cases

13 points of attention, where 3 are classified as `Critical`, 1 are classified as `High`, 1 is classified as `Medium`, 1 are classified as `Low`, 5 are classified as `Info` and 1 are classified as `Gas`, 1 are classified as `Best Practices`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 1 About Cairo Security Clan. Section 2 Disclaimer. Section 3 Executive Summary. Section 4 Summary of Audit. Section 5 Risk Classification. Section 6 Issues by Severity Levels. Section 7 Test Evaluation.



(a) distribution of issues according to the severity and status

Fig 1: (a) Distribution of issues: Critical (3), **High** (1), **Medium** (1), **Low** (1), **Informational** (5),**Gas** (1), **Best Practices** (1). **(b) Distribution of status: Fixed** (11), **Acknowledged** (2), **Mitigated** (0), **Unresolved** (0)

4 Summary of Audit

Audit Type	Security Review
Cairo Version	2.4.4
Solidity Version	0.8.20
Initial Report	12/04/2024
Response from Client	18/04/2024
Final Report	24/04/2024
Repository (L1 contracts)	0xSpaceShard/nimbora-yields-l1-public
Commit Hash (L1 contracts)	266abb8f2d6ac951cbd9abf0216d11182b9ccec6
Repository (L2 contracts)	0xSpaceShard/nimbora-yields-l2-public
Commit Hash (L2 contracts)	709cbd6f9faec067ee7ce8a610a61b3329b4d90f
Documentation	Website documentation
Test Suite Assessment	High

4.1 Scoped Files

	Layer 1 Contracts
1	/contracts/PoolingManager/PoolingManagerBase.sol
2	/contracts/PoolingManager/Implementation/StarknetPoolingManager.sol
3	/contracts/lib/StarknetMessaging.sol
4	/contracts/strategies/savingDai/savingDai.sol
5	/contracts/strategies/StrategyBase.sol

	Layer 2 Contracts
1	/src/token_manager/token_manager.cairo
2	/src/pooling_manager/pooling_manager.cairo
3	/src/token/token.cairo
4	/src/factory/factory.cairo
5	/src/token_bridge/token_bridge.cairo
6	/src/utils.cairo

4.2 Issues

	Findings	Severity	Update
1	Everyone can update critical storage variables.	Critical	Fixed
2	Everyone can upgrade the token bridge contract.	Critical	Fixed
3	Missing validation in `handle_deposit(...)`.	Critical	Fixed
4	L1 strategy calculated withdrawal amounts are inconsistent.	High	Fixed
5	L1 Relayer can lock deposited funds halt the L2 pooling manager.	Medium	Fixed
6	Lack of input validation of `l1_recipient`.	Low	Acknowledged
7	Token contract can not be upgraded.	Info	Fixed
8	Unwanted naming of access control system.	Info	Fixed
9	Consider checking `token_address` is registered to pooling manager.	Info	Fixed
10	Consider decreasing performance fee limit.	Info	Fixed
11	Token manager initialiser doesn't revert once already initialised.	Info	Fixed
12	Unnecessary address casting during report handling.	Gas	Fixed
13	Misleading variable naming.	Best Practices	Acknowledged

5 Risk Classification

The risk rating methodology used by Cairo Security Clan follows the principles established by the CVSS risk rating methodology. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Likelihood		
		High	**Medium**	**Low**
Impact	**High**	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Info/Best Practices

To address issues that do not fit a High/Medium/Low severity, Cairo Security Clan also uses three more finding severities: **Informational**, **Best Practices** and **Gas**

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

b) **Gas** findings are used when some piece of code uses more gas than it should be or have some functions that can be removed to save gas.

6 Issues by Severity Levels

6.1 CRITICAL

6.1.1 Everyone can update critical storage variables

File(s): /src/token_bridge/token_bridge.cairo

Description: Platform variables `l2_token` and `l1_bridge` addresses can be changed by everyone.

```
fn set_l2_token(ref self: ContractState, l2_token_address: ContractAddress) {
    assert(l2_token_address.is_non_zero(), ZERO_ADDRESS);
    self._l2_address.write(l2_token_address);
}

fn set_l1_bridge(ref self: ContractState, l1_bridge_address: felt252) {
    assert(l1_bridge_address.is_non_zero(), ZERO_ADDRESS);
    self._l1_bridge.write(l1_bridge_address);
}
```

Recommendation(s): Consider adding `accesscontrol` before updating storage.

Status: Fixed

Update from client: Fixed in this commit

6.1.2 Everyone can upgrade the token bridge contract.

File(s): /src/token_bridge/token_bridge.cairo

Description: `upgrade(...)` upgrades the token bridge contract to the targeted class hash. However, everyone can call this function and upgrade any class hashes.

```
fn upgrade(ref self: ContractState, new_class_hash: ClassHash) {
    self.upgradeable._upgrade(new_class_hash);
}
```

Recommendation(s): Consider adding `accesscontrol` before upgrading the contract.

Status: Fixed

Update from client: Fixed in this commit

6.1.3 Missing validation in `handle_deposit(...)`

File(s): /src/token_bridge/token_bridge.cairo

Description: `handle_deposit(...)` function assumes `from_address` variable as trusted. However, this parameter will written by the sequencer only if called from layer 1.

```
        fn handle_deposit(ref self: ContractState, from_address: felt252, account: felt252, amount: u256) {
            // Check account address is valid.
            assert(account.is_non_zero(), ZERO_ADDRESS);

            // Check token and bridge addresses are initialized and the handler invoked by the bridge.
            let l1_bridge = self._l1_bridge.read();
            assert(l1_bridge.is_non_zero(), UNINITIALIZED_L1_BRIDGE_ADDRESS);

            assert(from_address == l1_bridge, EXPECTED_FROM_BRIDGE_ONLY);
            //...
```

Recommendation(s): Consider validating `from_address` variable.

Status: Unresolved

Update from client: Function assumes `from_address` variable as trusted. However, this parameter will written by the sequencer only if called from layer 1.

6.2 HIGH

6.2.1 L1 strategy calculated withdrawal amounts are inconsistent.

File(s): /contracts/strategies/savingDai/savingDai.sol

Description: When a report with an withdrawal action is being processed by `_handleReport`, the L1 strategy's withdraw function is called, which interacts with the yield generating protocol, taking tokens and gained yield. The function will return the amount of underlying tokens that were successfully claimed, as shown below:

```
function _withdraw(uint256 _amount) internal override returns (uint256) {
    uint256 yieldAmountToWithdraw = ISavingDai(yieldToken).previewWithdraw(_amount);
    uint256 strategyYieldBalance = _yieldBalance();
    if (yieldAmountToWithdraw > strategyYieldBalance) {
        return ISavingDai(yieldToken).redeem(strategyYieldBalance, poolingManager, address(this));
    } else {
        return ISavingDai(yieldToken).withdraw(_amount, poolingManager, address(this));
    }
}
```

For the SavingsDai strategy, if the attempted withdraw amount is more than the contract is owed, the function redeem is used, otherwise withdraw is used instead. The return value from these functions are passed to the calling function, and is treated as the amount of underlying tokens that have been withdrawn.

However, the functions redeem and withdraw, based on the ERC4626 Tokenize Vault implementation return different values. redeem will return the amount of tokens recieved, while withdraw will return the number of yield tokens burned in order to claim the specified amount.

Since ERC4626 tokens are always worth more than the underlying, the return value from withdraw will be less than the actual amount of underlying tokens recieved. This less-than-underlying amount will be returned to the _handleReport logic which will then track that the withdrawal returned as less tokens than expected, even though the contract did recieve the correct amount of tokens. This difference in tokens which the report handler calculates as missing is then passed to the L2 pooling manager.

Over time, with the contract withdrawing more tokens that it has expected, the protocol may experience solvency issues where eventually users will not be able to withdraw their tokens, since the underlying tokens will have been slowly drained over time.

Recommendation(s): Consider correcting and re-checking the values returned by `redeem(...)` and `withdraw(...)` functions.

Status: Fixed

Update from client: Fixed in this commit

6.3 MEDIUM

6.3.1 L1 Relayer can lock deposited funds halt the L2 pooling manager.

File(s): /src/pooling_manager/pooling_manager.cairo

Description: In order to keep pooling manager data consistent between L1 and L2 strategy report, deposit, and withdrawal information is between each layer. In order to reduce gas costs this data is hashed before being sent. This data is then provided again in the "handle report" function, which is hashed, and the resulting hash must match what was sent from the other layer. This mechanism is designed to ensure that the correct data is sent between layers. A simplified version of the report/withdrawal/deposit hash function is shown below:

```
fn hash(...) -> Span<u256> {
    // Append epoch
    let mut ret_array = ArrayTrait::new();
    ret_array.append(new_epoch);

    // Append every deposit info
    loop {
        ret_array.append(deposit_info.l1_bridge);
        ret_array.append(deposit_info.amount);
        i += 1;
    };

    // Append every strategy report
    loop {
        ret_array.append(l1_strategy_u256);
        ret_array.append(strategy_report_l2_elem.action_id);
        ret_array.append(strategy_report_l2_elem.amount);
        ret_array.append(processed);
    };

    // Append every withdrawal info
    loop {
        ret_array.append(l1_bridge_u256);
        ret_array.append(bridge_withdrawal_info_elem.amount);
    };

    keccak_hash(ret_array.span());
}
```

This hashing approach involves three dynamic-sized arrays, but the lengths of these arrays are not part of the hash output. This makes it possible to shift data from one array to the other, while still having the same concatenated output. For example, a single strategy report of data 1,2,3,4 can be represented as two withdrawals containing 1,2 and 3,4. With the same concatenated output, the resulting hash will be the same, meaning it's possible to manipulate the report/deposit/withdrawal data.

When data is being sent from L2 to L1, the entries in `withdrawal_data` can be converted into additional `strategy_report` entries, as demonstrated below:

```
// Structure outlines
strategy_report_structure:
    - l1strategy (ethaddr)
    - data       (uint256)
    - amount     (uint256)
    - processed  (uint256)

deposit_data_structure:
    - bridge     (ethaddr)
    - amount     (uint256)

// Original data

strategy_report_array = []

deposit_data_structure = [
    {
        bridge: 0xBridgeAddressOne,
        amount: 0x2
    },
    {
        bridge: 0xBridgeAddressTwo,
```

```
        amount: 0x1234
    }
]

// Modified data

strategy_report_array = [
    {
        l1strategy: 0xBridgeAddressOne,
        data:       0x2,
        amount:     0xBridgeAddressTwo,
        processed:  0x1234
    }
]

deposit_data_structure = []
```

Since data has been removed from the `withdrawal_info` array, the withdrawals will never execute and funds will remain inaccessible on the bridge. When the additional corrupted strategy report is processed in the L1 `_handleReport` function, the action variable is derived from data, which is actually the withdrawal amount. This amount would have been arranged to be a small withdrawal of 2 wei, to align with the WITHDRAW enum value. This will cause an attempted call to withdraw on the l1strategy (that is actually the bridge address), which will fail since the bridge address does not support this function. The failed call will lead to the processed boolean being set to false. Finally this corrupted strategy report will be rewritten back into the array, and then hashed and sent to L2.

Once the hash message reaches the L2 pooling manager, it is impossible to call `handle_mass_report` without passing the hash check, requiring the corrupted strategy to be added. With the corrupted strategy added, it attempts to load the corresponding L2 token manager which will not exist, and then calls the function underlying on a zero address contract, as shown below:

```
// The `elem.l1_strategy` is corrupt data, the mapping will load the zero address
let strategy = self.l1_strategy_to_token_manager.read(elem.l1_strategy);
let strategy_disp = ITokenManagerDispatcher { contract_address: strategy };
// Attempted call to the zero address will fail, L2 pooling manager is DOS'd
let underlying = strategy_disp.underlying();
```

This not only leads to inaccessible funds on the L1 Starkgate bridge, but also leaves the L2 contract in a broken state which reverts on every call to `handle_mass_report`, requiring redeployment or a complex recovery attempt including a new implementation and storage migration.

Recommendation(s): Add the length of each array to the hash function to ensure that elements cannot be shifted between reports/deposits/withdrawals.

Status: Fixed

Update from client: Fixed in this commit

6.4 LOW

6.4.1 Lack of input validation of `l1_recipient`

File(s): /src/token_bridge/token_bridge.cairo

Description: `l1_recipient` parameter must be valid ethereum address. Otherwise, the l1 call might be failed.

```
fn initiate_withdraw(ref self: ContractState, l1_recipient: felt252, amount: u256) {
// ...
    let mut message_payload: Array<felt252> = ArrayTrait::new();
    message_payload.append(WITHDRAW_MESSAGE);
    message_payload.append(l1_recipient);
    message_payload.append(amount.low.into());
    message_payload.append(amount.high.into());
    send_message_to_l1_syscall(to_address: l1_bridge, payload: message_payload.span());
    self
        .emit(
            Event::WithdrawInitiated(
                WithdrawInitiated { l1_recipient: l1_recipient, amount: amount, caller_address: caller_address }
            )
        );
}
```

Recommendation(s): Consider checking `l1_recipient` is a valid Ethereum address with byte length.

Status: Acknowledged

Update from client: We are using it with tokenManager and the input can only be an Eth address.

6.5 INFORMATIONALS

6.5.1 Token contract can not be upgraded.

File(s): /src/token/token.cairo

Description: Token contracts `fn upgrade(...)` method can only called by token manager. However, the token manager has no functions that call the token contracts upgrade function.

Recommendation(s): Consider developing an upgrade pattern or removing the upgrade function from the token contract.

Status: Fixed

Update from client: Fixed in this commit.

6.5.2 Unwanted naming of access control system.

File(s): /src/factory/factory.cairo , /src/pooling_manager/pooling_manager.cairo , /src/token_manager/token_manager.cairo

Description: As implemented role-based access control system. Setting role `0` can be dangerous. Roles can be set as `felt252` values.

```
let has_role = access_disp.has_role(0, caller);
```

Recommendation(s): Use different role value than `0`.

Status: Fixed

Update from client: Fixed in this commit.

6.5.3 Consider checking token_address is registered to pooling manager

File(s): /src/pooling_manager/pooling_manager.cairo

Description: The Owner can set allowance for any ERC20 token from pooling manager.

```
fn set_allowance(
    ref self: ContractState, spender: ContractAddress, token_address: ContractAddress, amount: u256
) {
    self.accesscontrol.assert_only_role(0);
    // @audit-issue Check that token is registered.
    // It is not gonna needed to approve unregistered (non-bridged) token approval.
    assert(spender.is_non_zero() && token_address.is_non_zero(), Errors::ZERO_ADDRESS);
    let underlying_disp = ERC20ABIDispatcher { contract_address: token_address };
    underlying_disp.approve(spender, amount);
}
```

Recommendation(s): Consider checking that `token_address` is registered token inside `pooling_manager`.

Status: Fixed

Update from client: Function has been removed. Fixed in this commit.

6.5.4 Consider decreasing performance fee limit.

File(s): /src/pooling_manager/pooling_manager.cairo

Description: In current implementation performance fee can be set to any value lower than 10^18. This may cause accounting problems because fee can be up to almost 100

```
fn _set_performance_fees(ref self: ContractState, new_performance_fees: u256) {
    assert(new_performance_fees < CONSTANTS::WAD, Errors::INVALID_FEES);
    self.performance_fees.write(new_performance_fees);
}
```

Recommendation(s): Consider limiting performance fee lower than WAD value.

Status: Fixed

Update from client: Added limit of maximum 20% for performance fees in this commit.

6.5.5 Token manager initialiser doesn't revert once already initialised.

File(s): /src/token_bridge/token_bridge.cairo

Description: `Initialiser` implementations in most cases do not allow multiple calls, typically they include a check to revert if they have already been initialised. This can prevent important storage variables from being changed that could have a significant impact on how the contract behaves. The `initialiser` function in the token manager contract does not check if it has already been initialised, as shown below:

```
fn initialiser(ref self: ContractState, token: ContractAddress) {
    self._assert_only_pooling_manager();
    self.token.write(token);
}
```

Recommendation(s): With the current contract implementations, this doesn't pose a significant risk as the `initialiser` function can only be called by the pooling manager, which can only call `initialiser` when registering a strategy. However, the pooling manager is upgradeable, and this assumption may not hold for future implementations. To prevent future issues, and to adhere more closely to expected `initialiser` behaviors, we recommend reverting if the token manager has already been initialised.

Status: Fixed

Update from client: Fixed in this commit.

6.6 GAS OPTIMIZATIONS

6.6.1 Unnecessary address casting during report handling.

File(s): /contracts/PoolingManager/PoolingManagerBase.sol

Description: When a strategy report has been handled in the function _handleReport, the _report array is updated with a new StrategyReport object. When initializing this object, the l1Strategy field is assigned to be the address cast of the L1 strategy contract interface. An address cast can be avoided by using the already existing strategy variable, reducing the L1 gas costs for handling each report.

Recommendation(s): Consider removing address casting.

Status: Fixed

Update from client: Fixed in this commit.

6.7 BEST PRACTICES

6.7.1 Misleading variable naming.

File(s): /src/token_bridge/token_bridge.cairo

Description: In the pooling manager function handle_mass_report, the token manager is found by loading from the mapping l1_strategy_to_token_manager. The return value is stored in the variable strategy, although a name such as token_manager would be more appropriate. This follows through to the dispatcher, which is named strategy_disp where a name like token_manager_disp would be more accurate.

Recommendation(s): Consider renaming the variables mentioned above.

Status: Acknowledged

Update from client: Issue is acknowledged. Do not need for renaming.

7 Test Evaluation

7.1 Compilation Output

7.1.1 Layer 1

```
$ yarn compile
yarn run v1.22.19
warning ..\..\..\..\..\..\package.json: No license field
$ npx hardhat compile
Downloading compiler 0.8.20
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing
SPDX-License-Identifier: <SPDX-License> to each source file. Use SPDX-License-Identifier: UNLICENSED for non-open-source code. Please see https://spdx.org for more information.
--> contracts/mocks/erc20/weth.sol


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/mocks/starknet/bridge/MockStarknetErc20Bridge.sol:27:46:
   |
27 |     function depositReclaim(uint256 _amount, uint256 _l2Receiver, uint256 _nonce) external {
   |                                              ^^^^^^^^^^^^^^^^^^^


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/mocks/starknet/bridge/MockStarknetErc20Bridge.sol:27:67:
   |
27 |     function depositReclaim(uint256 _amount, uint256 _l2Receiver, uint256 _nonce) external {
   |                                                                   ^^^^^^^^^^^^^


Warning: Function state mutability can be restricted to pure
  --> contracts/mocks/chainlink/MockV3Aggregator.sol:71:5:
   |
71 |     function description() external view returns (string memory) {
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to pure
  --> contracts/mocks/starknet/starknetMock.sol:69:5:
   |
69 |     function getL1ToL2MsgHash(
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to pure
  --> contracts/strategies/Template.sol:39:5:
   |
39 |     function _underlyingToYield(uint256 _amount) internal view override returns (uint256) {
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to pure
  --> contracts/strategies/Template.sol:45:5:
   |
45 |     function _yieldToUnderlying(uint256 _amount) internal view override returns (uint256) {
   |     ^ (Relevant source part starts here and spans across multiple lines).


Generating typings for: 52 artifacts in dir: typechain-types for target: ethers-v6
Successfully generated 216 typings!
Compiled 49 Solidity files successfully (evm target: paris).
Done in 7.35s.
```

7.1.2 Layer 2

```
scarb build
   Compiling nimbora_yields v0.1.0 (/.../Scarb.toml)
    Finished release target(s) in 5 seconds
```

7.2 Tests Output

7.2.1 Layer 1

```
 npx hardhat test
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment
    containing SPDX-License-Identifier: <SPDX-License> to each source file. Use SPDX-License-Identifier:
    UNLICENSED for non-open-source code. Please see https://spdx.org for more information.
--> contracts/mocks/erc20/weth.sol

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/mocks/starknet/bridge/MockStarknetErc20Bridge.sol:27:46:
   |
27 |     function depositReclaim(uint256 _amount, uint256 _l2Receiver, uint256 _nonce) external {
   |                                              ^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/mocks/starknet/bridge/MockStarknetErc20Bridge.sol:27:67:
   |
27 |     function depositReclaim(uint256 _amount, uint256 _l2Receiver, uint256 _nonce) external {
   |                                                                   ^^^^^^^^^^^^^^

Warning: Function state mutability can be restricted to pure
  --> contracts/mocks/chainlink/MockV3Aggregator.sol:71:5:
   |
71 |     function description() external view returns (string memory) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning: Function state mutability can be restricted to pure
  --> contracts/mocks/starknet/starknetMock.sol:69:5:
   |
69 |     function getL1ToL2MsgHash(
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning: Function state mutability can be restricted to pure
  --> contracts/strategies/Template.sol:39:5:
   |
39 |     function _underlyingToYield(uint256 _amount) internal view override returns (uint256) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning: Function state mutability can be restricted to pure
  --> contracts/strategies/Template.sol:45:5:
   |
45 |     function _yieldToUnderlying(uint256 _amount) internal view override returns (uint256) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Compiled 49 Solidity files successfully (evm target: paris).

  Starknet Pooling Manager Test
    [PASS] Should register strategy
    [PASS] Should fail to register strategy: invalid strategy contract
    [PASS] Should fail to register strategy: not admin
HandleReport first deposit - 1/5
HandleReport withdraw all tokens - 2/5
HandleReport first deposit - 3/5
HandleReport withdraw half tokens - 4/5
HandleReport withdraw all the tokens - 5/5
    [PASS] Should success handle report, deposit/withdraw into a single strategy (sdai), simulate loose tokens
    (438ms)
HandleReport first deposit - 1/5
HandleReport withdraw all tokens - 2/5
HandleReport first deposit - 3/5
HandleReport withdraw half tokens - 4/5
HandleReport withdraw all the tokens - 5/5
```

```
    [PASS] Should success handle report, deposit/withdraw into a single strategy (sdai) (399ms)
HandleReport first deposit - 1/5
HandleReport withdraw all tokens - 2/5
HandleReport first deposit - 3/5
HandleReport withdraw half tokens - 4/5
HandleReport withdraw all the tokens - 5/5
    [PASS] Should success handle report, deposit/withdraw into a single strategy (uniswap) (382ms)
    [PASS] Should success handle report, deposit/withdraw into one strategy (sdai) when strategy reverted when withdraw (126ms)
    [PASS] Should success handle report, deposit/withdraw into one strategy (uniswapV3) when strategy reverted when withdraw (132ms)
    [PASS] Should success handle report, deposit/withdraw into one strategy (sdai) when strategy reverted when deposit (66ms)
    [PASS] Should success handle report, deposit/withdraw into one strategy (uniswap) when strategy reverted when deposit (63ms)
HandleReport deposit for multiple strategies - 1/2
HandleReport withdraw all from multiple strategies - 2/2
    [PASS] Should success handle report, deposit/withdraw into multiple strategies (sdai, uniswapV3) (249ms)
    [PASS] Should initilize the pooling manager
    [PASS] Should be able to call cancelDepositRequestBridge
    [PASS] Should be able to call claimBridgeCancelDepositRequest erc20
    [PASS] Should be able to call claimBridgeCancelDepositRequest eth
    [PASS] should return correct report hash (61ms)
    [PASS] Should register new strategy
    [PASS] Should fail to register new strategy, invalid contract implementation
    [PASS] Should fail to register new strategy, InvalidPoolingManager
    [PASS] Should return a valid l1->l2 hash
    [PASS] Should return a valid l2->l1 hash

  Strategy SavingDai Test
    [PASS] Should init the strategy
    [PASS] Should deposit into the strategy  (47ms)
    [PASS] Should withdraw from the strategy  (60ms)
    [PASS] Should return the underlyingToYield  (38ms)
    [PASS] Should return the yieldToUnderlying
    [PASS] Should return the depositCalldata
    [PASS] Should deposit into the strategy (66ms)
    [PASS] test acion withdraw enough yield balance (111ms)
    [PASS] Should withdraw when not enough yield balance (58ms)
    [PASS] Should withdraw all NAV (74ms)
    [PASS] Should fail if the caller is not admin
    [PASS] Should fail if the caller is pooling manager

  Strategy UniswapV3 Test
    [PASS] Should initilaze the strategy
    [PASS] Should be able to return chainlinkLatestAnswer
    [PASS] Should be able to return addressToApprove
    [PASS] Should be able to setMinReceivedAmountFactor, admin caller
    [PASS] Should not be able to setMinReceivedAmountFactor, not admin caller
    [PASS] Should fail setMinReceivedAmountFactor
    [PASS] Should return the correct underlyingToYield
    [PASS] Should return the correct yieldToUnderlying
amountOutMinimum 9500000000000000000n
expectedAmount 9500000000000000000n
    [PASS] Should return correct output applySlippageDepositExactInputSingle
    [PASS] Should return correct output applySlippageWithdrawExactOutputSingle
    [PASS] Should return the depositCalldata
    [PASS] Should deposit into the strategy (57ms)
    [PASS] Should deposit into the strategy many (267ms)
    [PASS] Should return different NAV (75ms)
    [PASS] Should withdraw from strategy (98ms)
    [PASS] Should withdraw from strategy, many times (177ms)
    [PASS] test action withdraw revert slippage


  50 passing (14s)
```

7.2.2 Layer 2

```
scarb test
     Running test nimbora_yields (snforge test)
   Compiling nimbora_yields v0.1.0 (/Users/erimvaris/Desktop/CSC/001-NIMBORA/fix/Scarb.toml)
    Finished release target(s) in 5 seconds


Collected 129 test(s) from nimbora_yields package
Running 129 test(s) from src/
[PASS] nimbora_yields::tests::test_factory::testFactory::set_token_manager_class_hash_zero_hash (gas: ~11022)
[PASS] nimbora_yields::tests::test_factory::testFactory::upgrade_factory (gas: ~11032)
[PASS] nimbora_yields::tests::test_factory::testFactory::set_token_class_hash (gas: ~58793)
[PASS] nimbora_yields::tests::test_factory::testFactory::set_token_class_hash_zero_hash (gas: ~11022)
[PASS] nimbora_yields::tests::test_factory::testFactory::deploy_strategy_wrong_caller (gas: ~11023)
[PASS] nimbora_yields::tests::test_factory::testFactory::set_token_class_hash_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_factory::testFactory::set_token_manager_class_hash_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    emit_performance_fees_updated_event_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_factory::testFactory::set_token_manager_class_hash (gas: ~58793)
[PASS] nimbora_yields::tests::test_factory::testFactory::deploy_strategy_test (gas: ~88239)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_request_withdrawal_and_claim_not_ready (gas:
    ~95649)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_l1_handler_wrong_epoch (gas: ~100692)
[PASS] nimbora_yields::tests::test_factory::testFactory::upgrade_factory_wrong_caller (gas: ~11023)
[PASS] nimbora_yields::tests::test_factory::testFactory::compute_salt_for_strategy_test (gas: ~11026)
[PASS] nimbora_yields::tests::test_factory::testFactory::compute_salt_for_strategy_large_data (gas: ~11026)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_fees_recipient_wrong_caller (gas:
    ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::register_underlying_wrong_caller (gas:
    ~58779)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_deposit_event_wrong_caller (gas:
    ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_performance_fees_updated_event (gas:
     ~88237)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::convert_to_assets (gas: ~90677)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_handle_report_wrong_data (gas: ~103134)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_fees_recipient_zero_address (gas:
    ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_deposit_event (gas: ~88241)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    emit_request_withdrawal_event_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_fees_recipient (gas: ~12251)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_l1_pooling_manager_wrong_caller (gas:
     ~11022)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_handle_report_multiple_strategy (gas: ~128804)
[PASS] nimbora_yields::tests::test_factory::testFactory::deploy (gas: ~11026)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_handle_report_2_deposit (gas: ~100587)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_handle_report (gas: ~103155)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_claim_withdrawal_event_wrong_caller
    (gas: ~11022)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::initialize_token_manager (gas: ~70416)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_handle_report_no_l1 (gas: ~100662)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_dust_limit_updated_event (gas:
    ~88236)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    emit_withdrawal_epoch_delay_updated_event_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_request_withdrawal_and_claim (gas: ~134896)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_mass_executions_no_l1_report (gas:
     ~100689)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_claim_withdrawal_event (gas: ~88239)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    emit_dust_limit_updated_event_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_mass_executions_invalid_hash (gas:
     ~103152)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_withdrawal_epoch_delay_updated_event
     (gas: ~88237)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_mass_executions (gas: ~106260)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_response_pending_hash (gas:
    ~11039)
```

```
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_l1_handler_pending_hash (gas: ~101918)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::is_initialised (gas: ~14705)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_response_invalid_caller (gas: ~100700)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_mass_executions_buffer_is_null (gas: ~87015)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_l1_pooling_manager_zero_address (gas: ~11023)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_response (gas: ~100699)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_dust_limit_wrong_caller (gas: ~88239)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::upgrade_token_manager_zero_class_hash (gas: ~88239)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_response_invalid_epoch (gas: ~100700)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::upgrade_token_manager_wrong_caller (gas: ~88239)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_l1_pooling_manager (gas: ~12251)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::upgrade_token_manager (gas: ~88221)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_factory_zero_address (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_factory_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::set_factory (gas: ~12251)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_mass_report_not_initialised (gas: ~11022)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::
    token_manager_set_performance_fees_wrong_caller (gas: ~88239)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_mass_executions_not_initialized (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    emit_token_manager_class_hash_updated_event (gas: ~12257)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    emit_token_manager_class_hash_updated_event_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    emit_token_class_hash_updated_event_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::handle_mass_executions_invalid_data (gas: ~88240)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l1_data_empty_strategy (gas: ~100692)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::register_underlying_zero_address (gas: ~58780)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    register_underlying_zero_address_l1_bridge (gas: ~58780)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::register_underlying_zero_address_bridge (gas: ~58780)
[PASS] nimbora_yields::tests::test_factory::testFactory::upgrade_factory_zero_class_hash (gas: ~11023)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::register_underlying (gas: ~61232)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::upgrade_pooling_manager (gas: ~11027)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l2_data_collision (gas: ~100744)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_handle_report_with_bigger_asset_value (gas: ~107521)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::upgrade_pooling_manager_zero_class_hash (gas: ~11023)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l1_data_collision (gas: ~100707)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::upgrade_pooling_manager_wrong_caller (gas: ~11023)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_l1_handler_wrong_caller (gas: ~100691)
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_simple_l1_handler (gas: ~106271)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::register_strategy_wrong_caller (gas: ~88239)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::register_strategy_zero_bridge (gas: ~88239)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_tvl_limit_updated_event_wrong_caller (gas: ~11022)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::register_strategy_duplicated (gas: ~106600)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::
    delete_all_pending_strategies_wrong_caller (gas: ~88239)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::delete_all_pending_strategies (gas: ~85781)
```

```
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_deposit_limit_updated_event (gas: ~88237)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_request_withdrawal_event (gas: ~88242)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_deposit (runs: 22, gas: {max: ~93173, min: ~93173, mean: ~93173.00, std deviation: ~0.00})
[PASS] nimbora_yields::tests::test_end_to_end::testEndToEnd::test_handle_report_5_deposit (gas: ~104406)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_handle_result_withdrawal (runs: 22, gas: {max: ~3687, min: ~15, mean: ~3353.00, std deviation: ~1055.63})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_tvl_limit (runs: 22, gas: {max: ~89452, min: ~88228, mean: ~89340.00, std deviation: ~351.88})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_deposit_tvl_limit_reached (runs: 22, gas: {max: ~88241, min: ~88241, mean: ~88241.00, std deviation: ~0.00})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_dust_limit_zero (gas: ~88239)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_handle_result_report (runs: 22, gas: {max: ~4921, min: ~2473, mean: ~4698.00, std deviation: ~599.22})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_dust_limit (runs: 22, gas: {max: ~89446, min: ~88222, mean: ~89334.00, std deviation: ~351.88})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_deposit_limit_zero (gas: ~88239)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_deposit_limit_wrong_caller (gas: ~88239)
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::emit_token_class_hash_updated_event (gas: ~12257)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_handle_result_deposit (runs: 22, gas: {max: ~7983, min: ~7983, mean: ~7983.00, std deviation: ~0.00})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_claim_withdrawal_invalid_id (runs: 22, gas: {max: ~89468, min: ~88244, mean: ~89412.00, std deviation: ~254.96})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_handle_report_invalid_caller (gas: ~88239)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_check_profit_and_mint (runs: 22, gas: {max: ~23904, min: ~20232, mean: ~23625.00, std deviation: ~900.81})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::deploy_token_manager_test_initial_values (gas: ~69197)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::initialize_token_manager_wrong_caller (gas: ~69189)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::
    token_manager_set_withdrawal_epoch_delay_zero_epoch (gas: ~88239)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::
    token_manager_set_withdrawal_epoch_delay_wrong_caller (gas: ~88239)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_total_underlying_due (runs: 22, gas: {max: ~304970, min: ~731, mean: ~108638.00, std deviation: ~89638.04})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_claim_withdrawal_already_claimed (runs: 22, gas: {max: ~91921, min: ~91921, mean: ~91921.00, std deviation: ~0.00})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_convert_to_shares (gas: ~90677)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_request_withdrawal (runs: 22, gas: {max: ~98098, min: ~95650, mean: ~97986.00, std deviation: ~509.92})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_claim_withdrawal_not_ready (runs: 22, gas: {max: ~93149, min: ~91925, mean: ~93093.00, std deviation: ~254.96})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_total_assets (runs: 22, gas: {max: ~236079, min: ~9822, mean: ~90704.00, std deviation: ~80229.72})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::
    test_internal_handle_withdrawals_epoch_lt_delay (runs: 22, gas: {max: ~11, min: ~10, mean: ~10.00, std deviation: ~0.21})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_convert_to_shares (runs: 22, gas: {max: ~258581, min: ~30043, mean: ~110198.00, std deviation: ~78637.52})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_withdrawal_epoch_delay (runs: 22, gas: {max: ~89447, min: ~88223, mean: ~89335.00, std deviation: ~351.88})
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l2_data_old_epoch (gas: ~100695)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_claim_withdrawal (runs: 22, gas: {max: ~98080, min: ~96856, mean: ~97968.00, std deviation: ~351.88})
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l1_data (gas: ~100696)
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_performance_fees_too_high (runs: 22, gas: {max: ~88241, min: ~88241, mean: ~88241.00, std deviation: ~0.00})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_convert_to_assets (runs: 22, gas: {max: ~258583, min: ~30046, mean: ~110201.00, std deviation: ~78637.28})
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l2_data_empty_all (runs: 20, gas: {max: ~100696, min: ~100696, mean: ~100696.00, std deviation: ~0.00})
```

```
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::
    test_internal_calculate_profit_and_handle_loss_profit_case (runs: 22, gas: {max: ~3689, min: ~2465, mean:
    ~3577.00, std deviation: ~351.88})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_withdrawal_exchange_rate (runs:
     22, gas: {max: ~6141, min: ~3693, mean: ~5918.00, std deviation: ~703.75})
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l2_data_empty_withdrawal (runs: 20,
    gas: {max: ~100718, min: ~100718, mean: ~100718.00, std deviation: ~0.00})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::token_manager_set_performance_fees (runs: 22,
     gas: {max: ~88224, min: ~87000, mean: ~88168.00, std deviation: ~254.96})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::
    test_internal_calculate_profit_and_handle_loss_loss_case (runs: 22, gas: {max: ~22277, min: ~3812, mean:
    ~12314.00, std deviation: ~4556.90})
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l2_data_empty_deposit (runs: 20, gas
    : {max: ~100718, min: ~100718, mean: ~100718.00, std deviation: ~0.00})
[PASS] nimbora_yields::tests::test_pooling_manager::testPoolingManager::hash_l2_data_empty_strategy_report (runs:
     20, gas: {max: ~100710, min: ~100710, mean: ~100710.00, std deviation: ~0.00})
[PASS] nimbora_yields::tests::test_token_manager::testTokenManager::test_internal_handle_withdrawals (runs: 200,
    gas: {max: ~23402, min: ~2470, mean: ~8079.00, std deviation: ~4278.64})
Tests: 129 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out
Fuzzer seed: 13350441203973598141
```