HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Standard Protocol
Date: May 18th, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Standard Protocol
Approved By	Noah Jelich | Lead SC Auditor at Hacken OU
Type	DEX
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.standardweb3.com/
Changelog	18.05.2023 - Initial Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Standard Protocol (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/standardweb3/new-order-contracts
Commit	70bb67f
Whitepaper	https://github.com/standardweb3/Whitepaper/blob/main/whitepaper_en.md
Functional Requirements	
Technical Requirements	
Contracts	File: contracts/MatchingEngine.sol SHA3: 6d553d6d3381ec4ec507f6915311bd4543c387291ab6b4da3fd1469c6403e8ac File: contracts/interfaces/IEngine.sol SHA3: a8cb31559b41fa5f1f229335de6a96e7321f2b9528105c200006af059b31fafa File: contracts/interfaces/IERC20Minimal.sol SHA3: 9d97e9f2da45594d073100a62acd74b5bc26d0aec1e586a48b3057ff14944c11 File: contracts/interfaces/IOrderbook.sol SHA3: 067bc0d439b70a51509d47d9cc5cbc648f1b74f057e401ba0080974baf0fcb64 File: contracts/interfaces/IOrderbookFactory.sol SHA3: 23a9ec240a324be8743dbfc8ac6286b33c96fd0ef26b685ce4e845c4c57ce783 File: contracts/interfaces/IOrderFactory.sol SHA3: ddd88819e95e5fc2b3ee28018a621470ef06eb0e22ecb7b71fbefdb7b16549f9 File: contracts/libraries/CloneFactory.sol SHA3: fe5d2e3a0795d1b190b09a77959b51e1d3801da76bd8e0c9c31e0c5c39bc9c2b File: contracts/libraries/NewOrderLinkedList.sol SHA3: 52f2498ca4ab6443b298d6ac8322d5082dd30eadfd40ff6e820b22de32aad662 File: contracts/libraries/NewOrderOrderbook.sol SHA3: 7a0efed63c23fa726597069d16bff4c3085771956a6a38cae6f0817fb76b6922 File: contracts/libraries/TransferHelper.sol SHA3: 5685485c2b38be811e6a33d26f17cad8ae7d223fdd6e977776baee130d523686 File: contracts/membership/BlockAccountant.sol

	SHA3: 150bf16f74850b55bd57e671cf4a5ae570a55028f52031681e0215249588fd18 File: contracts/membership/Membership.sol SHA3: 90e9b9a88c4cff5634bb8eeaf8dd5aed8f62563826910ba1aaf9375ad031be3d File: contracts/membership/Metadata.sol SHA3: c93110e31594e96e9e3c95643fa97b4ad2298d8db944e9482ae6d23225ef66dd File: contracts/membership/SABT.sol SHA3: e43a434b6fa14d7e548f6d7cb91e4f851bc22ee72410c42e182dac5231c7904a File: contracts/membership/TimeAccountant.sol SHA3: 39179e4b0a162462a247671d5d02d1e26e1c769a63871f65e57c5db6efb08d4b File: contracts/membership/Treasury.sol SHA3: 412e13aec9db061f758c73197dc9243f9136a7e5a5b352c9a6ea3b1a24defbf9 File: contracts/membership/interfaces/IAccountant.sol SHA3: 86ea4f2e900b51b93bc95ac5890db835e413b717ec38cc1f32f4095befb2fc72 File: contracts/membership/interfaces/IMetadata.sol SHA3: 82b3ee216488ad6af7f7d8c95ed60875f59b31703e13a56cc1b4caf6d6e41c83 File: contracts/membership/interfaces/ISABT.sol SHA3: e899a8d096411a0302a9bad74c430ea9d9168ce53676a1f2844cd28f70c134d2 File: contracts/membership/libraries/BlockAccountantLib.sol SHA3: d6243847acb78848abfc6aa26701f69c7fcacd4a5ed4852c972775ce9675876a File: contracts/membership/libraries/MembershipLib.sol SHA3: 0c19fe21063f86771daa69c574c7e9ebf01538931c45d6e14bd1dc1152c32649 File: contracts/membership/libraries/TimeAccountantLib.sol SHA3: 0c5947df8d4b8b39cdae543381209ca258d1ed80eb45bc15f745eddd506a27f9 File: contracts/membership/libraries/TransferHelper.sol SHA3: f78c895185830be20bc4faec39b67b12e7360641a1d3270b0f6ed35efd05f213 File: contracts/membership/libraries/TreasuryLib.sol SHA3: 4f81255c73f00ef20c2dca2b80b520d15d5be2e619b075cdd9c3210568e59afc File: contracts/orderbooks/Orderbook.sol SHA3: c123c00c3737ddda63e9beeee3f857d288c0838bcf3b1481ed58b5453c68c438 File: contracts/orderbooks/OrderbookFactory.sol SHA3: e838298c1ca1ff70346951810b90c918a4da25e54718ce356dab5d96370cda00 File: contracts/security/Initializable.sol SHA3: 84db31f84b6c762bf8b924d87690e0cf457a06cc90ca8738ba7ead0d6289eae1

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **1** out of **10**.

- Functional requirements are not provided.
  - Missing Natspec in most of the contracts/functions.
  - Documentation provided does not go down contracts and functions' expected behavior and functionality.
  - Orderbook logic needs deeper explanation.
  - Linked lists seem illogic and unoptimized, needing deeper explanation on how they are build and managed.
  - User interactions and whole flows need deeper explanation (see Questionnaire).
  - It is not clear which contracts are upgradeable or the use of proxies.
- Technical description is not provided.
  - Project technical specification is not provided.
  - Descriptions of the development environment are not provided.
  - The NatSpec is partially provided.

Code quality

The total Code Quality score is **0** out of **10**.

- The code has redundant variable updates, infinite loops.
- Functions are too big and use duplicated code.
- Lack of use of modifiers (e.g. role checks should be modifiers unless impossible).
- Code is duplicated for the use of ask/bid instead of using a compatible structure for both.
- The code violates Solidity style guides.
- The development environment is not configured.
- Function names and variable names are not self-explanatory (e.g. _absdiff()).
- Error messages do not self-explain (e.g. "IA")
- The `BlockAccountant` contract occasionally utilizes the `require` statement to manually verify the admin role, while at other times, it leverages the `onlyRole` modifier for the same purpose.

Test coverage

Code coverage of the project is **29.51%** (branch coverage).

- None of the contracts are tested thoroughly.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Security score

As a result of the audit, the code contains **6** critical, **17** high, **17** medium and **14** low severity issues. The security score is **0** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **0.1**. The system users should acknowledge all the risks summed up in the risks section of the report.

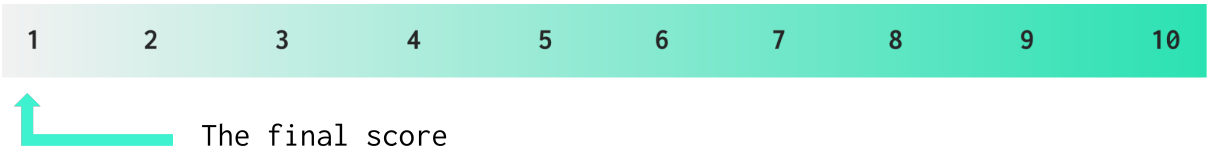

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
18 May 2023	14	17	17	6

Risks

- The factory contract creates an Orderbook pair with any type of ERC20, ERC777 token without any validation. This may result in creation of malicious token pairs.
- The use of ERC777 pairs may allow re-entrancy attacks.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

New Order is the decentralized order book exchange to use a smart contract as a matching engine. It provides functions for creating the order books, submitting and canceling orders as well as querying the order book state.

The main contract MatchingEngine.sol serves as the entry point for interacting with the system, while the Orderbook, NewOrderLinkedList, and NewOrderOrderbook libraries provide the underlying functionality for managing orders and prices.

The files in the scope:

- MatchingEngine.sol - main contract responsible for managing the order books. It uses OrderbookFactory to create new order books for each trading pair. It also provides functions for submitting and canceling orders, as well as querying the order book state.
- OrderbookFactory.sol - acts as a factory for creating new instances of the Orderbook contract. It ensures that there is only one order book per trading pair.
- Orderbook.sol - represents a single order book for a specific trading pair. It stores and manages buy and sell orders using the NewOrderLinkedList and NewOrderOrderbook libraries.
- NewOrderOrderbook.sol - provides a custom implementation to store and manage orders within the Orderbook contract.
- NewOrderLinkedList.sol - provides a custom implementation of a linked list to store and manage the bid and ask prices in the order book.
- TransferHelper.sol - provides methods for interacting with ERC20 tokens and sending ETH safely.
- Initializable.sol - provides support of initializer functions.
- CloneFactory.sol - responsible for efficient deployment of multiple order books.
- IOrderbookFactory.sol - interface for OrderbookFactory.sol.
- IOrderbook.sol - interface for Orderbook.sol.
- IERC20Minimal.sol - interface for ERC20 decimals.
- TreasuryLib.sol - The TreasuryLib library provides functions for exchanging membership points for rewards, claiming rewards, and settling remaining funds within the Treasury system.
- Treasury.sol - The Treasury contract enables subscribers to exchange membership points for rewards, and investors to claim rewards.
- MembershipLib.sol - The Membership library manages membership subscriptions, registration, and subscription status.
- Membership.sol - The Membership contract. It is based on the MembershipLib library; it allows users to register as members, subscribe to memberships until a certain block height, unsubscribe

from memberships, check membership balances, and retrieve membership metadata.

- BlockAccountant.sol - the BlockAccountant contract is responsible for managing membership points and their accounting.
- BlockAccountantLib.sol - The BlockAccountantLib library provides functions for reporting membership points, migrating points between members, checking subscription status, subtracting membership points, and retrieving total points for a given era.
- TimeAccountantLib.sol - TimeAccountantLib is a library that enables the reporting of membership points based on time, allowing the calculation of total points and rewards for different eras in a time-based accounting system.
- TimeAccount.sol - The TimeAccountant contract is an implementation of the membership accountant that tracks and reports membership points based on time, allowing the retrieval of total points and tokens for different eras.
- SABT.sol - The SABT contract is an ERC1155 token contract that represents membership tokens and allows for the minting and transfer of customized membership tokens.

Privileged roles

- OrderbookFactory
  - DEFAULT_ADMIN_ROLE: can change implementation of the order book and engine contract.
- MatchingEngine
  - DEFAULT_ADMIN_ROLE: can change the order book factory contract, fee token and fee amounts as well as fee receiver.
- SABT
  - DEFAULT_ADMIN_ROLE: can initialize the contract.
- TimeAccountant
  - DEFAULT_ADMIN_ROLE: can set membership, engine, reference currency contract addresses.
- Treasury
  - DEFAULT_ADMIN_ROLE: can set the claim amount for each uid and settlement id.
  - REPORTER_ROLE: capability for token withdrawals from the contract.
- Membership
  - DEFAULT_ADMIN_ROLE: can initialize the contract by setting the fees for registration and subscription, as well as the token address.
- BlockAccountant

- DEFAULT_ADMIN_ROLE: Admin has the ability to set BFS values, membership, engine, reference currency, and treasury contract addresses. Furthermore, they can migrate points from one era to another.
- REPORTER_ROLE: can report the membership point of the member.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Failed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Failed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Failed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Failed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Failed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Failed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Failed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Failed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Failed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Failed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Failed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Failed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Failed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Failed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

C01. Invalid Calculations; Assets Integrity

The *_convert()* function produces unexpected results for certain input values, potentially due to a problem in the calculation logic. Specifically, when the function is executed with the following parameters:

- `dQuote = 32`
- `quoteD = 6`
- `baseBquote = true`
- `price = 1000000000000000`
- `amount = 1000000000000000`
- `isAsk = false`

It returns `*1590897978359414784000000000*`, and when executed with `*isAsk = true*`, it returns `*628*`. These values are not consistent with expected outcomes.

This could lead to incorrect calculation of the required tokens when placing or executing orders, potentially causing incorrect execution prices, imbalances, or failed transactions.

Path: ./contracts/orderbooks/Orderbook.sol : _convert()

Recommendation: Refactor the function to work with all decimal places.

Found in: 70bb67f

Status: New

C02. Access Control Violation; Point Manipulation

The *report* function in the *TimeAccountant* contract is accessible to any user, which could potentially expose the system to malevolent activities. This unrestricted access enables malicious actors to manipulate any user points.

This can lead to a loss of funds.

Path: ./contracts/membership/TimeAccountant.sol : report()

Recommendation: restrict access to the *report* function. Allow only authorized parties.

Found in: 70bb67f

Status: New

C03. Invalid Calculations; Loss of Funds

The *_unsubscribe* function contains an erroneous fee computation, attributed to an incorrect addition operation within the *_subscribe* function. The latter function erroneously sums the *untilBh_* input variable with the *block.number* value, subsequently mapping it to *subscribedUntil[uid_].bh*. According to the function's design, *untilBh_* should be greater than *block.number*, implying that an extra block should be added to the current block number. However, the actual implementation erroneously doubles the current block number. This discrepancy enables a malicious user to repeatedly subscribe and unsubscribe, thereby exploiting the system to receive higher fee refunds than initially paid.

This can lead to a loss of funds.

Path: ./contracts/membership/libraries/MembershipLib.sol : _subscribe(), _unsubscribe()

Recommendation: Within the *_subscribe* function, the assignment *self.subscribedUntil[uid_].bh = uint64(block.number) + untilBh_* could be modified to *self.subscribedUntil[uid_].bh = untilBh_*.

Found in: 70bb67f

Status: New

C04. Invalid Calculations; Unlimited Reward

The system does not account for the claimed rewards. The users can claim the tokens as much as they want.

This can lead to a loss of funds.

Path: ./contracts/membership/libraries/TreasuryLib.sol : _getClaim(), _getReward(), _claim(), _exchange()

Recommendation: Re-implement the reward mechanism.

Found in: 70bb67f

Status: New

C05. Undocumented Behavior; Proxy Data Consistency; Access Control

It is not documented which contracts are upgradeable or use proxy patterns.

While some contracts use initialization functions instead of constructors, others use both.

It is important to understand which contracts are upgradeable in order to implement the different patterns needed.

One of them is the fact that state variables are not initialized and constructors do not work correctly. Thus, the contract variables must be updated using initialization functions.

Path: .

Recommendation: 1) Provide clear documentation about which contracts are upgradable and/or use proxy patterns, 2) use initialization functions for upgradeable contracts instead of constructors and do not initialize state variables as explained in OpenZeppelin docs, 3) use disableInitializers() in constructors.

Found in: 70bb67f

Status: New

C06. Calls to Unstrusted Contracts

Calls to untrusted contracts can introduce several unexpected risks or errors. External calls may execute malicious code in that contract or any other contract that it depends upon.

It is recommended to not call any untrusted external contracts. When it is not possible, or undesirable to remove external calls, use the following recommendations to minimize the danger.

The users can manually introduce the pairs they want to work with, introducing tokens that are out of the control of the system.

Path: contracts/libraries/TransferHelper.sol: safeApprove(), safeTransfer(), safeTransferFrom(), safeTransferETH(), decimals().

Recommendation:

- Mark untrusted contracts (in their variable name).
- Avoid state changes after external calls (use a CEI pattern).
- Whitelist contracts.
- Use ERC functions instead of low-level calls.

Found in: 70bb67f

Status: New

High

H01. Undocumented Behavior

The *initialize* and *setImpl* functions in the *OrderbookFactory.sol* contract can be called multiple times, but this behavior is not documented.

The *initialize* functions in the *SABT.sol* contract can be called multiple times, but this behavior is not documented.

This can potentially lead to unintended consequences or confusion for users interacting with the contract.

Path: ./contracts/orderbooks/OrderbookFactory.sol : initialize(), setImpl()

Recommendation: Either clearly document the behavior of the *initialize* and *setImpl* functions in the contract, including reasons why it is allowed to call them multiple times or prevent them from being called more than once or implement initializer modifier.

Found in: 70bb67f

Status: New

H02. Denial Of Service Vulnerability; Infinite Loop

In case of *else if (required == 0)* the loop in the function will never end because the variable *i* is not updated.

Path: ./contracts/MatchingEngine.sol: _matchAt()

Recommendation: update the variable *i* before using the *continue* keyword.

Found in: 70bb67f

Status: New

H03. Highly Permissive Role Access

DEFAULT_ADMIN_ROLE of the *BlockAccountant.sol* contract can change the *treasury* address. The *treasury* address can decrease the users points with the *subtractMP* function. This would allow the owner to control the user points.

DEFAULT_ADMIN_ROLE in the *Treasury.sol* contract can alter the reward ratio. Such a change could have a significant impact on pending user rewards.

Owners should not have access to funds that belong to users.

Path: ./contracts/membership/libraries/BlockAccountantLib.sol

Recommendation: Either articulate this feature explicitly in the public-facing documentation or consider implementing constraints on owner privileges.

Found in: 70bb67f

Status: New

H04. Highly Permissive Role Access

DEFAULT_ADMIN_ROLE of the *Membership.sol* contract possesses the ability to alter the *feeToken* address subsequent to users subscribing and remitting the fee with the original *feeToken*. In the event a user decides to unsubscribe after having paid the fee with the previous feeToken address, they may receive a token of a value that diverges from their initial expectations, either more or less valuable.

Owners should not have access to funds that belong to users.

Path: ./contracts/membership/Membership.sol: subscribe()

Recommendation: Either articulate this feature explicitly in the public-facing documentation or consider implementing constraints on owner privileges.

Found in: 70bb67f

Status: New

H05. Highly Permissive Role Access; Undocumented Behavior

The function refundFee() allows the transfer of the balance in Treasury.sol to any address without notice, although the name of the function suggests it only allows the transfer of fees.

A role with such high impact should be properly documented and have robust access control mechanisms.

If a key leak were to occur, the potential consequences could be significant, potentially leading to security breaches and undermining the overall integrity of the system.

Path: ./contracts/membership/Treasury.sol: refundFee().

Recommendation: To ensure transparency and accountability, it is advised to provide a comprehensive explanation of highly-permissive access in the system's public documentation. This would help to ensure that users are fully informed of the implications of such access and can make informed decisions accordingly.

Found in: 70bb67f

Status: New

H06. Highly Permissive Role Access; Undocumented Behavior

The function migrate() allows the transfer of membership points between addresses without notice.

A role with such high impact should be properly documented and have robust access control mechanisms.

If a key leak were to occur, the potential consequences could be significant, potentially leading to security breaches and undermining the overall integrity of the system.

Path: ./contracts/membership/BlockAccountant.sol: migrate().

Recommendation: To ensure transparency and accountability, it is advised to provide a comprehensive explanation of highly-permissive access in the system's public documentation. This would help to ensure that users are fully informed of the implications of such access and can make informed decisions accordingly. Multiple signatures for such roles are also recommended.

Found in: 70bb67f

Status: New

H07. Requirement Violation; Missing check

The register function includes the condition *require(membership.metas[metaId_].metaId_ == metaId_, "IM")*, which stipulates that the input *metaId_* should be equivalent to *_membership.metas[metaId_].metaId*. if a malicious user submits *metaId_* as zero. Given that *membership.metas[metaId_]* is empty, this condition would be satisfied. Since the token address defaults to 0x0, this allows the malicious user to mint the zero *metaId_* without any payment.

This can lead to a loss of funds.

Path: ./contracts/membership/Membership.sol: register()

Recommendation: Implement necessary checks.

Found in: 70bb67f

Status: New

H08. Undocumented Behavior

The system provides three distinct types of meta id(*USER_META_ID*, *INVESTOR_META_ID*, *FOUNDATION_META_ID*), offering users the flexibility to select a specific meta id when minting tokens. Users can mint a particular meta id by transmitting the corresponding parameter to the register function. This feature is not documented.

The code should not contain undocumented functionality.

Path: ./contracts/membership/Membership.sol: register()

Recommendation: Articulate this feature explicitly in the public-facing documentation.

Found in: 70bb67f

Status: New

H09. Funds Lock; Insufficient Funds

Under circumstances of insufficient funds, the contract blocks fulfilling user rewards or executing exchanges. In this scenario, the contract still allows new user membership registration.

This will lead to blocking user rewards or exchange and refundFee functions.

Path: ./contracts/membership/Treasury.sol: exchange(), claim(), refundFee()

Recommendation: Implement a balance tracking mechanism to ensure sufficient funds are available to fulfill claim obligations. If the contract's balance is insufficient to cover forthcoming rewards, temporarily suspending new registrations would be better.

Found in: 70bb67f

Status: New

H10. Ambiguous Third Party Integration

The contract MatchingEngine is designed as an upgradeable contract inheriting from OpenZeppelin upgradeable contracts but does not implement storage gaps.

As a result, there is an incompatibility with the integration of OpenZeppelin that will result in data consistency issues.

Path: contracts/MatchingEngine.sol

Recommendation: use storage gaps.

Found in: 70bb67f

Status: New

H11. Data Consistency; Missing UID check

The update of data from uid is not checked against the existence of uid.

This can lead to data consistency issues, unexpected behavior, data overwriting.

Path:
./contracts/membership/BlockAccountant.sol: migrate(), report(), subtractMP().
./contracts/membership/Membership.sol: subscribe(), unsubscribe().
./contracts/membership/SABT.sol: setMetaId().
./contracts/membership/TimeAccountant.sol: report().
./contracts/membership/Treasury.sol: exchange(), claim(), settle(), setClaim(), setSettlement().
./contracts/MatchingEngine.sol: marketBuy(), marketSell(), limitBuy(), limitSell(), stopBuy(), stopSell(), cancelOrder(),

Recommendation: Check that uid exists every time data is updated regarding uid.

Found in: 70bb67f

Status: New

H12. Undocumented Functionality: Orders

It is not clear how the system manages the coexistence of several orders of the same price in terms of storage and matching: are they accumulated in the same position or in different, are they partially filled and eliminated only once filled completely, etc.

Path: .

Recommendation: clearly document how orders of the same price are stored and matched.

Found in: 70bb67f

Status: New

H13. Undocumented Functionality: Price feeds.

It is not clear how the system feeds prices: is the system integrating an external oracle? If so, which is that oracle and how is it integrated? Otherwise, the used method should be documented.

Path: .

Recommendation: clearly document how the system gets the prices of the traded assets.

Found in: 70bb67f

Status: New

H14. Highly Permissive Role Access

`DEFAULT_ADMIN_ROLE` in the `Treasury.sol` contract can alter the reward ratio. Such a change could have a significant impact on pending user rewards.

Owners should not have access to funds that belong to users.

Path: ./contracts/membership/Treasury.sol

Recommendation: Either articulate this feature explicitly in the public-facing documentation or consider implementing constraints on owner privileges.

Found in: 70bb67f

Status: New

H15. Denial Of Service - Function Inaccessibility

The function will no longer work in this scenario; If the owner of the contract executes the function with a *600000* input by mistake due to the *require(self.totalClaim <= 600000, "OVERFLOW")* require check, the function no longer can be used except if the given parameter is zero.

Paths: ./contracts/membership/libraries/TreasuryLib.sol: _setClaim()

Recommendation: Either explain the behavior in the documentation if it's intended or re-implement the function.

Found in: 70bb67f

Status: New

H16. Inconsistent data - Point Unbalances

The function subtracts points from the *pointOf* mapping but does not subtract from the *totalPointsOn* and *totalTokensOn* mappings.

This can lead to unbalances.

Paths: ./contracts/membership/libraries/BlockAccountantLib.sol: _subtractMP()

Recommendation: Either explain the behavior in the documentation if it's intended or also subtract points from the *totalPointsOn* and *totalTokensOn* mapping.

Found in: 70bb67f

Status: New

H17. Denial Of Service - Improper Verification

The control statement *require(feeNum < 1e8 && feeDenom < 1e8 && feeNum_ < feeDenom_, "NF")*; in the function verifies the state variables rather than the input(*feeNum_, feeDenom_*) variables. If inputs exceeding 1e8 are provided, the function may fail to operate in future instances. This discrepancy could potentially disrupt the expected functionality.

Path: ./contracts/MatchingEngine.sol: setFee

Recommendation: Use the input variables *feeNum_* and *feeDenom_* instead of the current state variables *feeNum* and *feeDenom*. Following this change, the *feeNum_ < 1e8* check becomes redundant, given the existing *feeNum_ < feeDenom_* check.

Found in: 70bb67f

Status: New

Medium

M01. Missing Events

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Paths:
./contracts/orderbooks/Orderbook.sol: placeAsk(), cancelOrder(), placeBid(), execute()
./contracts/orderbooks/OrderbookFactory.sol: setImpl()
./contracts/orderbooks/Orderbook.sol: setLmp(), placeBid(), placeAsk()
./contracts/MatchingEngine.sol : initialize(), setFeeTo(), setFee(), setListingFee(), setOrderbookFactory(), setMembership(), setAccountant()
./contracts/membership/BlockAccountant.sol: setEngine(), setMembership(), setTreasury(), setReferenceCurrency(), setBFS(), migrate(), report(), subtractMP()
./contracts/membership/Treasury.sol: exchange(), claim(), settle(), setClaim(), setSettlement(), refundFee()
./contracts/membership/TimeAccountant.sol: setEngine(), setMembership(), setReferenceCurrency()
./contracts/membership/SABT.sol: setMetaId()
./contracts/membership/Membership.sol : setFoundation(), setMembership(), register(), subscribe(), unsubscribe()

Recommendation: create and emit related events.

Found in: 70bb67f

Status: New

M02. Inefficient Gas Model - Uncontrolled Num of Loop Iterations

The functions contain loops with uncontrolled numbers of iterations that depend on the input data. This can potentially lead to out-of-Gas exceptions if the input variable `n` is large enough.

Paths:
./contracts/MatchingEngine.sol: _limitOrder()
./contracts/libraries/NewOrderOrderbook.sol: _getPrices()
./contracts/libraries/NewOrderOrderbook.sol: _getOrders
./contracts/orderbooks/OrderbookFactory.sol: getAllPairs()
./contracts/MatchingEngine.sol: cancelOrders()

Recommendation: implement a maximum limit on the number of iterations in the loop.

Found in: 70bb67f

Status: New

M03. Bad variable naming

Undocumented usage of variables named *uid*, *fb*, *bfs*, *lmp*, *era* overwhelm code and makes further development difficult.

Paths:
./contracts/membership/BlockAccountant.sol
./contracts/membership/libraries/BlockAccountantLib.sol
./contracts/MatchingEngine.sol
./contracts/membership/Membership.sol
./contracts/membership/libraries/MembershipLib.sol
./contracts/membership/SABT.sol
./contracts/membership/TimeAccountant.sol
./contracts/membership/libraries/TimeAccountantLib.sol
./contracts/membership/Treasury.sol
./contracts/membership/libraries/TreasuryLib.sol
./contracts/interfaces/IOrderbook.sol
./contracts/orderbooks/Orderbook.sol
./contracts/MatchingEngine.sol
./contracts/libraries/NewOrderLinkedList.sol

Recommendation: Provide variable names according to their purposes.

Found in: 70bb67f

Status: New

M04. Bad function naming

The function _absdiff() does not represent the functionality of the function. In addition, the input parameters a and b, increase the complexity of comprehension.

Other functions that have non-representative names are: fb(), getBfs(), subtractMP(), setBFS(), _subtractMP(), _getMP(), _detStop(), mktPrice(), _setLmp(), _mktPrice(), _fpop(), setLmp(), fpop(), _createImpl(), getMp().

Using function names and variables that do not represent their purpose decrease code readability.

Paths:
./contracts/orderbooks/Orderbook.sol: _absdiff(), setLmp(), fpop().
./contracts/membership/BlockAccountant.sol: fb(), getBfs(), subtractMP(), setBFS()
./contracts/membership/libraries/BlockAccountantLib.sol: _subtractMP(), _getMP().
./contracts/MatchingEngine.sol: _detStop(), mktPrice().
./contracts/libraries/NewOrderLinkedList.sol: _setLmp(), _mktPrice().
./contracts/libraries/NewOrderOrderbook.sol: _fpop().
./contracts/orderbooks/OrderbookFactory.sol: _createImpl().
./contracts/membership/TimeAccountant.sol: getMp().
./contracts/membership/libraries/TimeAccountantLib.sol: _getMP().

Recommendation: Provide variable names according to their purposes.

Found in: 70bb67f

Status: New

M05. Inconsistent data - Division before multiplication

The calculation *uint32(28 days / bfs_)*, could potentially result in a loss of precision. This imprecision may subsequently result in an incorrect computation of *_accountant.era*, potentially inducing unanticipated behaviors.

Paths: ./contracts/membership/BlockAccountant.sol: setBFS()

Recommendation: Either explain the behavior in the documentation if it's intended or perform multiplication before division.

Found in: 70bb67f

Status: New

M06. Inconsistent Data - Unsafe Casting

The *_migrate()* function parameter accepts the uint256 *amount_* then casts the variable to uint64. The overflow may happen when casting uint256 to uint64.

The *_report()* function has the calculation *uint32((block.number - self.fb) / self.era)* the function then casts calculation results to the uint32. The overflow may happen when casting uint256 to uint32.

The *_report()* function has the calculation *uint32((block.timestamp - self.ft) / 28 days);* the function then casts calculation results to the uint32. The overflow may happen when casting uint256 to uint32.

The *_report()* function has the calculation *uint64((converted * 1e5) / IAccountant(token).decimals());* the function then casts calculation results to the uint64. The overflow may happen when casting uint256 to uint64.

Path: ./contracts/membership/libraries/BlockAccountantLib.sol: _migrate(), _report()

./contracts/membership/libraries/TimeAccountantLib.sol: _report()

Recommendation: Ensure that the castings are safe and do not result in overflow or loss of data.

Found in: 70bb67f

Status: New

M07. Inconsistent data - Division before multiplication

The mathematical operation `converted *1e5 / 10 ** IAccountant(self.stablecoin).decimals();` could potentially return an incorrect result, in scenarios where the `converted` variable has a decimal value different from `IAccountant(self.stablecoin).decimals()`

Paths: ./contracts/membership/BlockAccountantLib.sol: _report()

Recommendation: Either explain the behavior in the documentation if it's intended or perform multiplication before division.

Found in: 70bb67f

Status: New

M08. Contradiction - Unfinalized code

The provided code should be implemented in the full logic of the project. Since any missing parts, TODOs, or drafts can change in time, the robustness of the audit cannot be guaranteed.

Incomplete code impacts project reliability and makes it harder to evaluate project security.

The Membership contract Lines 57-59.

The Orderbook contract Line 29.

The contracts contain commented functionality.

Paths:

./contracts/membership/Membership.sol: register()
./contracts/membership/BlockAccountant.sol: migrate().
./contracts/orderbooks/Orderbook.sol

Recommendation: Remove the commented code or finalize its implementation.

Found in: 70bb67f

Status: New

M09. Contradiction - Missing validation

According to implementation, the `untilBh_` value should be greater than the current `block.number`. However, in the function the validation is missing.

Paths: ./contracts/membership/Membership.sol: _subscribe()

Recommendation: Implement the necessary validation.

Found in: 70bb67f

Status: New

M10. Requirement Violation

The smart contract assumes that one block is created every three 15, which may not be accurate due to the varying block creation times on the blockchain network. This can result in the calculation of inaccurate *`subscribedUntil`* variables.

Block values are not precise, and the use of them can lead to wrong calculations.

Path: ./contracts/StakeSpacePi.sol : pending()

Recommendation: Instead of assuming a fixed block generation time, use the *`block.timestamp`* value for calculations in the smart contract.

Found in: 70bb67f

Status: New

M11. Best practice violation - CEI pattern violation

The Checks-Effects-Interactions pattern is violated. During the functions, some state variables are updated after the external calls.

Path: ./contracts/membership/libraries/MembershipLib.sol : _subscribe(), _unsubscribe()

Recommendation: implement the functions according to the Checks-Effects-Interactions pattern.

Found in: 70bb67f

Status: New

M12. Inconsistent data - Broken Logic

The function behavior is intended to show the prices shown in the orderbook but actually the function logic is not working as expected.

Path: ./contracts/libraries/NewOrderOrderbook.sol : _getPrices()

Recommendation: Re-implement the function logic according to the intended behavior.

Found in: 70bb67f

Status: New

M13. Best Practice Violation - Uninitialized Implementation

The UUPSUpgradeable contract is inherited but not initialized.

Path: ./contracts/MatchingEngine.sol: initialize()

Recommendation: According to Openzeppelin all projects using the UUPS proxy pattern should initialize their implementation contracts.

Found in: 70bb67f

Status: New

M14. Non-representative error message

The following error messages do not provide valuable information about the cause of failure: “IA”, “IM”, “TMM”, “NF”, “DECIMALS”, “OrderbookFactory: IA”, “OrderbookFactory: OB”, “NM”, "IA:notRprtr", "NO_UID", “AF”, “TF”, “TFF”, “ETF”, “DF”.

This leads to confusion, lack of precision and decreased readability and troubleshooting.

Paths:
./contracts/orderbooks/Orderbook.sol: initialize(), createBook(), execute(), cancelOrder(), placeAsk(), placeBid(), setLmp().
./contracts/membership/BlockAccountant.sol: setEngine(), setMembership(), setTreasury(), setReferenceCurrency(), setBFS(), report(), subtractMP().
./contracts/membership/libraries/BlockAccountantLib.sol: _migrate().
./contracts/orderbooks/OrderbookFactory.sol: initialize().
./contracts/membership/TimeAccountant.sol: setEngine(), setMembership(), setReferenceCurrency().
./contracts/membership/Treasury.sol: setClaim(), setSettlement(), refundFee().
./contracts/membership/libraries/TreasuryLib.sol: _checkMembership(), _getClaim(), _getSettlement().
./contracts/membership/Membership.sol: register().
./contracts/MatchingEngine.sol:_matchAt(), setFee().
./contracts/membership/SABT.sol: mint(), setMetaId().
./contracts/libraries/TransferHelper.sol: safeApprove(), safeTransfer(), safeTransferFrom(), safeTransferETH(), decimals().

Recommendation: Provide an error message that reflects the cause.

Found in: 70bb67f

Status: New

M15. Code Duplication

Role checks in the project are introduced by a piece of code that is repeated many times among contracts’ functions, instead of using the onlyRole modifier.

Extensive code duplications make smart contracts less readable and more expensive to deploy.

Path:

./contracts/membership/BlockAccountant.sol: setEngine(), setMembership(), setTreasury(), report().

./contracts/MatchingEngine.sol: _authorizeUpgrade().
./contracts/orderbooks/OrderbookFactory.sol: initialize().
./contracts/membership/TimeAccountant.sol: setEngine(), setMembership(), setReferenceCurrency().
./contracts/membership/Treasury.sol: setClaim(), setSettlement(), refundFee().

Recommendation: It is recommended to use modifiers for repetitive checks when possible.

Found in: 70bb67f

Status: New

M16. Copy of Well-known contracts

Well-known contracts from projects like OpenZeppelin should be imported directly from source as the projects are in development and may update the contracts in future.

Path: ./contracts/security/Initializable.sol

Recommendation: Import the contract directly from source instead of modifying it.

Found in: 70bb67f

Status: New

M17. Inefficient Gas Model: Storage Abuse

The storage struct self is accessed many times instead of creating a memory variable in order to save Gas.

Due to the complexity of the functions, the impact on Gas expense is huge.

Path:
./contracts/membership/libraries/BlockAccountantLib.sol: _report(), _migrate().
./contracts/membership/libraries/MembershipLib.sol: _subscribe(), _unsubscribe().
./contracts/libraries/NewOrderLinkedList.sol: _mktPrice(), _insert().
./contracts/libraries/NewOrderOrderbook.sol: _insertId().
contracts/membership/libraries/TimeAccountantLib.sol: _report().

Recommendation: Use memory variables as a method to save Gas instead of working directly with storage variables.

Found in: 70bb67f

Status: New

Low

L01. Floating Pragma

The project uses floating pragmas *0.8.10, 0.8.0*

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Paths:
./contracts/libraries/TransferHelper.sol
./contracts/libraries/NewOrderOrderbook.sol
./contracts/libraries/NewOrderLinkedList.sol
./contracts/libraries/CloneFactory.sol
./contracts/orderbooks/Orderbook.sol
./contracts/orderbooks/OrderbookFactory.sol
./contracts/interfaces/IOrderbookFactory.sol
./contracts/interfaces/IOrderFactory.sol
./contracts/interfaces/IERC20Minimal.sol
./contracts/MatchingEngine.sol
./contracts/security/Initializable.sol
./contracts/membership/libraries/BlockAccountantLib.sol
./contracts/membership/libraries/MembershipLib.sol
./contracts/membership/libraries/TimeAccountantLib.sol
./contracts/membership/libraries/TransferHelper.sol
./contracts/membership/libraries/TreasuryLib.sol
./contracts/membership/BlockAccountant.sol
./contracts/membership/Membership.sol
./contracts/membership/Metadata.sol
./contracts/membership/TimeAccountant.sol
./contracts/membership/Treasury.sol
./contracts/membership/interfaces/ISABT.sol

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 70bb67f

Status: New

L02. Functions That Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

Paths: ./contracts/MatchingEngine.sol : initialize()

./contracts/orderbooks/Orderbook.sol: initialize()

./contracts/orderbooks/Orderbook.sol: getRequired()

./contracts/membership/TimeAccountant.sol: setEngine(), setMembership(), setReferenceCurrency(), report(), getTotalPoints(), getTotalTokens()

./contracts/membership/SABT.sol: mint(), setMetaId(), transfer(), initialize()

Recommendation: use the external attribute for functions never called from the contract.

Found in: abc1135

Status: New

L03. State Variable Default Visibility

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Path: ./contracts/orderbooks/Orderbook.sol: *uint256* index

./contracts/membership/libraries/TreasuryLib.sol: USER_META_ID, INVESTORS_META_ID, FOUNDATION_META_ID, denom

Recommendation: variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

Found in: abc113528d4f9bc41ea0ebdeec630288339a18e3

Status: **Fixed (Revised commit: 70bb67f)**

L04. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Paths:
./contracts/orderbooks/OrderbookFactory.sol : setImpl(), initialize(), createBook().
./contracts/MatchingEngine.sol : initialize(), setFeeTo(), setListingFee(), setOrderbookFactory(), setMembership(), setAccountant(), setFeeTo(), marketBuy().
./contracts/membership/BlockAccountant.sol : constructor(), setEngine(), setMembership(), setTreasury(), setReferenceCurrency()
./contracts/membership/Treasury.sol: constructor()
./contracts/membership/libraries/TreasuryLib.sol: _exchange(), _claim(), _settle(), _setClaim()
./contracts/membership/TimeAccountant.sol: constructor(), setEngine(), setMembership(), setReferenceCurrency()
./contracts/membership/SABT.sol: mint(), initialize().
./contracts/membership/Membership.sol: initialize(), setMembership(), setFoundation()

./contracts/orderbooks/Orderbook.sol: initialize(), placeBid(), placeAsk().

Recommendation: implement zero address checks.

Found in: abc1135

Status: New

L05. Style Guide Violation

The provided projects should follow the official guidelines.

Inside each contract, library or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Modifiers
5. Functions

Functions should be grouped according to their visibility and ordered:

1. constructor
2. receive function (if exists)
3. fallback function (if exists)
4. external
5. public
6. internal
7. private

Within a grouping, place the view and pure functions last.

It is best practice to cover all functions with NatSpec annotation and to follow the Solidity naming convention. This will increase overall code quality and readability.

Path: ./contracts/contract.sol : function()

Recommendation: follow the official Solidity guidelines.

Found in: abc1135

Status: New

L06. Use of hard-coded values

The constants '28 days' and '1e5' are directly embedded into the computations. These magic numbers can lead to potential misunderstandings or errors in future changes or optimizations.

In the getReward function, a calculation (point * totalTokens * 4) / 10) / totalMP is performed, where the magic numbers '4' and '10' appear. The semantic meaning of these constants is unclear.

Similarly, in the _getSettlement function, there's a computation ((totalTokens * (600000 - self.totalClaim)) / denom). Here, the magic number '600000' is used, and its purpose is unclear without additional context or documentation.

The use of these hard-coded values can lead to less maintainable code, reduce understandability and transparency of the code, and potentially introduce errors if these constants need to change in the future.

Paths: ./contracts/membership/BlockAccountant.sol: constructor(), setBFS(), _report()

./contracts/membership/libraries/TimeAccountantLib.sol: _report()

./contracts/membership/libraries/TreasuryLib.sol: _getReward(), _getSettlement(), _setClaim

Recommendation: Convert these variables into constants.

Found in: 70bb67f

Status: New

L07. Duplicate Control Statement

The function checks the same *isAdd* variable two times and updates state variables accordingly. This will increase the code size, gas consumption and reduce the readability of the code.

Paths: ./contracts/membership/libraries/BlockAccountantLib.sol: _report()

Recommendation: Instead of multiple if-else shortcuts currently in place, it would be more efficient to implement a single if-else control statement, with variables updated accordingly.

Found in: 70bb67f

Status: New

L08. Redundant Import

The usage of *AccessControl* is unnecessary for the contract.

Paths: ./contracts/membership/Metadata.sol

Recommendation: Remove the redundant import.

Found in: 70bb67f

Status: New

L09. Zero Valued Transactions

The function _subscribe can execute a zero-valued transaction if `self.subscribedUntil[uid_].bh` and `self.subscribedAt[uid_].bh` equal to zero.

This can lead to a transaction with zero value to be sent.

Paths:./contracts/membership/libraries/MembershipLib.sol: _subscribe()

Recommendation: Implement conditional checks for the zero-valued transaction.

Found in: 70bb67f

Status: New

L10. Redundant Condition

The condition defined as `self.count = self.count == 0 || self.count == type(uint256).max ? 1 : self.count + 1;` contains redundancy. The `self.count == 0` condition is superfluous because, in this scenario, the else statement will yield an equivalent result.

Paths: ./contracts/libraries/NewOrderOrderbook.sol: _createOrder()

Recommendation: Remove the `self.count == 0` condition.

Found in: 70bb67f

Status: New

L11. Undocumented Literal

Different numbers are used among the contracts, without documentation about them.

Paths:
./contracts/membership/libraries/TreasuryLib.sol: _getSettlement() → 600000, _setClaim() → 600000.
./contracts/MatchingEngine.sol: _matchAt() → 100.

Recommendation: Document all literals.

Found in: 70bb67f

Status: New

L12. Non-explicit variable size

Different numbers are used among the contracts, without documentation about them.

Paths:
./contracts/membership/libraries/TransferHelper.sol: safeApprove(), safeTransfer(), safeTransferFrom(), safeTransferETH().

./contracts/orderbooks/OrderbookFactory.sol: _createImpl().

Recommendation: Set variable size explicitly for uint and bytes.

Found in: 70bb67f

Status: New

L13. Variables That Can Be Set Immutable

Use immutable keywords on state variables to limit changes to their state and save Gas.

Paths:

./contracts/membership/SABT.sol: membership, metadata.

Recommendation: Set variable size explicitly for uint and bytes.

Found in: 70bb67f

Status: New

L14. Inefficient Gas Model: Cache Length

The length of storage arrays is used in loops instead of caching the length in memory variables, resulting in a big and unnecessary amount of Gas expense.

Path:
./contracts/MatchingEngine.sol: cancelOrders().
contracts/orderbooks/OrderbookFactory.sol: getAllPairs().

Recommendation: Cache the length of storage arrays into memory variables when performing loops.

Found in: 70bb67f

Status: New

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.