Ethora

Audit Report

Wed Jan 10 2024

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

Ethora Audit Report

1 Executive Summary

1.1 Project Information

Description	Ethora is a decentralized binary options trading protocol.
Type	Options
Auditors	ScaleBit
Timeline	Fri Dec 08 2023 - Wed Jan 10 2024
Languages	Solidity
Platform	Base
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/LibraTechSolution/BO_SMC
Commits	6aed4c697485167207ed640012eff96e9faaf6f6 5da9e46e919f71335ec167502ba03ef28060e872

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
ABD	contracts/library/ABDKMath64x64.sol	77d7148dc649bad541f2378117df111e8d215d19
VAL	contracts/library/Validator.sol	20b067901608ea618d553db89ee12e3ae03eff4c
OMA	contracts/library/OptionMath.sol	3a72284be37173595e8bf52698940bb31f516880
ARE	contracts/AccountRegistrar.sol	1c2efc39bbfac7e3b299f04a4c5760987253f3c7
RDI	contracts/earn/RewardDistributor.sol	ef5f6f8547ac492ef8543751c24bddd6f47dfdf4
BTO	contracts/earn/BaseToken.sol	08c5940f64b0bc6290845175de3853906462fea0
MBT	contracts/earn/MintableBaseToken.sol	8aafb682e68452b6a879ce47091290503190096b
EETR	contracts/earn/EsETR.sol	0badcf0a8fe3e4edb1f58994c6bfb70c45786fb2
RTR	contracts/earn/RewardTracker.sol	acfb36d0e8c868dda43bd71ad0f11907a0a0d158
BDI	contracts/earn/BonusDistributor.sol	2433ee35e0ce7e4c8315c299bb796805d0e67d91
GOV	contracts/earn/Governable.sol	3297820b00dc42400aab430f5df62ebe37a4469f

RRO	contracts/earn/RewardRouter.sol	4846e33d87fc8fe8d581a5ea0129bb63594a37d6
VES	contracts/earn/Vester.sol	1003fb87044ff05b7cfc154d8c93ec891f75ccba
RST	contracts/ReferralStorage.sol	83e2a2c1acd4f0c7b2bf88a3bf34d0d8471b357e
SFD	contracts/SettlementFeeDistributor.sol	94c739971277df7cc53119c430730a0e4676d661
EBP	contracts/EthoraBinaryPool.sol	7de8b1d2e4d7b9e077b82fb127d887bb83af94be
CWI	contracts/CreationWindow.sol	e76f27d7b0e4f3ec24212b5cee03d4bdf8fc2a9c
EBO	contracts/EthoraBinaryOptions.sol	4237ed96090f0c9f86e93c66b70b0ceb63c3a102
FAU	contracts/Faucet.sol	ed0cfaa63b29d0da665d9a7630ab863f6f8866a9
OST	contracts/OptionStorage.sol	17ba77462a62aa1bfeba0894d194a08c3ab54bb6
IYT	contracts/interfaces/IYieldTracker.sol	fa89235d6d8e604f6c4e419ace045023bb165ed2
IRT	contracts/interfaces/IRewardTracker.sol	9caca4394303813c944db9b53de459faa6357ca4
IMI	contracts/interfaces/IMintable.sol	c9679f79b06dcdd29bac30f13b93884dd4150387
IEM	contracts/interfaces/IElpManager.sol	77885b336fa92e2b365f8dc78968dab0d9f56f16

IVE	contracts/interfaces/IVester.sol	a471971a9f7bfbc1a196487788148a94f800ce13
IRD	contracts/interfaces/IRewardDistributor.sol	0b51ac724136a38e0b57ff77dee92056aea9c3b9
IBT	contracts/interfaces/IBaseToken.sol	1d45b7cd98f691732c307e17843bc3394172c3b2
ITS	contracts/interfaces/ITokenSale.sol	ed1451f85d786a7f1e1d33f98856d0f27f08047b
INT	contracts/interfaces/interfaces.sol	a55501b7b191ef28140be123bbe73d53a3c8336c
ERO	contracts/EthoraRouter.sol	14193f85ee3c2d4465d6703739749599845d5c79
POIS	contracts/PoolOIStorage.sol	ffc86a0e221aa041bffeff7bffcd4f2284a72dc0
USDC	contracts/USDC.sol	c6071e6dcebf18e93f8c269e2c1925d494a273da
MOIC	contracts/MarketOIConfig.sol	ca8ecd19dc4e7f33fd6c1938a03afc78575f3980
POIC	contracts/PoolOIConfig.sol	33cf52cc8dd1d32b48534e5ce73678434d32974d
TSA	contracts/tokenSale/TokenSale.sol	0d4bd3a13f48210c7f8a5af68a8d43a3084c5095
WLI	contracts/tokenSale/WhiteList.sol	6b6f332f4873835b71b9feab50c62b75ecf637cc
BOO	contracts/Booster.sol	7a59082641794d1f81761a5dec704119b574cdfc

TOK	contracts/Token.sol	fe405b1393a1a0f72d7c71cd15c4794153f878ba
OCO	contracts/OptionsConfig.sol	2844d8ff02442bd0b2e4838fe5b4955fceff885c

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	28	25	3
Informational	3	3	0
Minor	17	16	1
Medium	3	2	1
Major	5	4	1
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Ethora to identify any potential issues and vulnerabilities in the source code of the Ethora smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 28 issues of varying severity, listed below.

ID	Title	Severity	Status
ABD-1	Splitting `require()` Statements that Use `&&` Saves Gas	Minor	Fixed
ARE-1	Centralization Risk for Trusted Owners	Major	Acknowledged
ARE-2	For Operations That will not Overflow, You could Use Unchecked	Minor	Fixed
ARE-3	Using Private rather than Public for Constants, Saves Gas	Minor	Fixed
ARE-4	Use Calldata Instead of Memory for Function Arguments That Do not Get Mutated	Informational	Fixed
BDI-1	State Variables Should Be Cached in Stack Variables Rather Than Re-reading Them from Storage	Minor	Fixed
BTO-1	`approve()` and `transferFrom()` Functions are Subject to Front-run attack	Medium	Acknowledged

BTO-2	When the Current Allowance is Set to `type(uint256).max`, There is No Need to Update the Allowance	Medium	Fixed
EBO-1	Use Custom Errors	Minor	Fixed
EBO-2	`++i` Costs Less Gas Than `i++`, Especially When It's Used in For-loops (`--i/i--` Too)	Minor	Acknowledged
EBO-3	Use Shift Right/Left instead of Division/Multiplication if Possible	Minor	Fixed
EBO-4	Use `!= 0` instead of `> 0` for Unsigned Integer Comparison	Minor	Fixed
EBO-5	Functions Not Used Internally Could be Marked External	Minor	Fixed
EBP-1	Use `_grantRole()` instead of `_setupRole()`	Major	Fixed
EBP-2	User Boost will not be Updated	Major	Fixed
EBP-3	Unused Variables	Minor	Fixed
EBP-4	Unused Import of `SafeMathUpgradeable`	Minor	Fixed
EBP-5	Using Bools for Storage Incurs Overhead	Minor	Fixed
EBP-6	Functions Guaranteed To Revert When Called By Normal Users Can be Marked Payable	Minor	Fixed
EBP-7	`require()` / `revert()` Statements Should Have Descriptive Reason Strings	Minor	Fixed

EBP-8	Long Revert Strings	Informational	Fixed
ERO-1	Blacklisted User can Block the `openTrades()`	Major	Fixed
ERO-2	Cache Array Length Outside of Loop	Minor	Fixed
ERO-3	Don't Initialize Variables with Default Value	Informational	Fixed
FAU-1	Single-step Ownership Transfer Can be Dangerous	Major	Fixed
OST-1	The Purpose of the save() Function	Medium	Fixed
POI-1	Event is Missing Indexed Fields	Minor	Fixed
TSA-1	`abi.encodePacked()` Should Not be Used with Dynamic Types When Passing the Result to a Hash Function Such as `keccak256()`	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Ethora Smart Contract:
Admin

- Admin can set contract registry through `setContractRegistry()`.
- Admin can set publisher addresses through `setPublisher()`.
- Admin can update admin address through `setAdmin()`.
- Admin can manage keepers through `setKeeper()`.
- Admin manages handlers through `setHandler()`.
- Admin sets the tokenX address through `setTokenX()`.
- Admin adjusts the lockup period through `setLockupPeriod()`.
- Admin modifies the maximum liquidity through `setMaxLiquidity()`.
- Admin registers an account through `registerAccount()`.
- Admin deregisters an account through `deregisterAccount()`.
- Admin configures the contract with parameters such as tokens, pools, and categories through `ownerConfig()`.
- Admin sets the IV configuration parameters for ITM and OTM options through `setIvConfig()`.
- Admin or authorized users can pause/unpause option creation through `setIsPaused()`.
- Admin approves specific addresses through `approveAddress()`.
- Admin sets token pairs through `setToken()`.

User

- User can approve transactions via signature through `approveViaSignature()`.
- User can revoke previously granted approvals through `revokeApprovals()`.
- Users contribute liquidity through `provide()`.
- Users withdraw liquidity through `withdraw()`.

    - User can stake their own ETR tokens through `stakeEtr()`.
- User can stake their ES ETR tokens through `stakeEsEtr()`.
- User can unstake their ETR tokens through `unstakeEtr()`.
- User can unstake their ES ETR tokens through `unstakeEsEtr()`.
- User can mint and stake ELP tokens through `mintAndStakeElp()`.
- User can unstake and redeem ELP tokens through `unstakeAndRedeemElp()`.
- User can claim accumulated rewards through `claim()`.
- User can claim ES ETR rewards through `claimEsEtr()`.
- User can claim fee rewards through `claimFees()`.
- User can compound rewards through `compound()`.
- User can handle various reward-related actions through `handleRewards()`.
- User can signal the transfer of stakes from one address to another through `signalTransfer()`.
- User can accept the signaled transfer of stakes between addresses through `acceptTransfer()`.

Keeper

- Keeper can open trades through `openTrades()`.
- Keeper can attempt early closure of pending trades through `closeAnytime()`.
- Keeper can execute specific options through `executeOptions()`.

Option issuers

- Option issuers lock funds in an option through `lock()`.
- Option issuers unlock funds from an option through `unlock()`.
- Option issuers distribute funds to liquidity providers after options expiration through `send()`.

Owner

- Owner can set the pool contract address through `setPool()`.

- Owner can set the address for the creation window contract through `setCreationWindowContract()`.
- Owner adjusts the minimum fee required for options creation through `setMinFee()`.
- Owner modifies the implied volatility (IV) through `setIV()`.
- Owner updates the platform fee through `setPlatformFee()`.
- Owner sets the address for the settlement fee disbursal contract through `setSettlementFeeDisbursalContract()`.
- Owner can configure the maximum period allowed for options through `setMaxPeriod()`.
- Owner can configure the minimum period allowed for options through `setMinPeriod()`.
- Owner sets the contract address for pool open interest storage through `setPoolOIStorageContract()`.
- Owner sets the contract address for pool open interest configuration through `setPoolOIConfigContract()`.
- Owner sets the contract address for market open interest configuration through `setMarketOIConfigContract()`.
- Owner adjusts the early close threshold through `setEarlyCloseThreshold()`.
- Owner toggles the allowance for early close functionality through `toggleEarlyClose()`.
- Owner adds a single address to the whitelist through `addAddressToWhitelist()`.
- Owner adds multiple addresses to the whitelist through `addAddressesToWhitelist()`.
- Owner removes a single address from the whitelist through `removeAddressFromWhitelist()`.
- Owner removes multiple addresses from the whitelist through `removeAddressesFromWhitelist()`.

Gov

- Gov can withdraw tokens mistakenly sent to the contract through `withdrawToken()`.
- Gov can batch stake ETR tokens for multiple accounts through `batchStakeEtrForAccount()`.

- Gov can stake ETR tokens for a specific account through `stakeEtrForAccount()`.
- Gov can batch compound rewards for multiple accounts through `batchCompoundForAccounts()`.
- Gov can compound rewards for a specific account through `compoundForAccount()`.

4 Findings

ABD-1 Splitting `require()` Statements that Use `&&` Saves Gas

Severity: Minor

Status: Fixed

Code Location:

contracts/library/ABDKMath64x64.sol#33,89

Descriptions:

Splitting `require()` statements using `&&` individually consume more gas in Solidity. Solidity combines all conditions into a single `require()` statement, and gas calculation occurs during the evaluation of the merged condition. Splitting conditions into separate `require()` statements leads to individual gas calculations for each statement rather than a consolidated evaluation, resulting in additional gas consumption. Consolidating these conditions within a single `require()` statement is more efficient and reduces gas usage.

Suggestion:

It is recommended to consolidate multiple conditions within a single `require()` statement using `&&` to optimize gas efficiency in Solidity.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ARE-1 Centralization Risk for Trusted Owners

Severity: Major

Status: Acknowledged

Code Location:

contracts/AccountRegistrar.sol#12,24,40

Descriptions:

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

```
contract AccountRegistrar is IAccountRegistrar, AccessControl

 external override onlyRole(ADMIN_ROLE) {

 external onlyRole(ADMIN_ROLE) {
```

Suggestion:

It is recommended to use multi-signature or additional permission controls can mitigate this risk.

ARE-2 For Operations That will not Overflow, You could Use Unchecked

Severity: Minor

Status: Fixed

Code Location:

contracts/AccountRegistrar.sol#50,53

Descriptions:

For Operations that will not overflow, you could use unchecked.

```
nonce: nonce + 1,

emit DeregisterAccount(user, nonce + 1);
```

Suggestion:

It is recommended to use Solidity's `unchecked` block to save the overflow checks.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ARE-3 Using Private rather than Public for Constants, Saves Gas

Severity: Minor

Status: Fixed

Code Location:

contracts/AccountRegistrar.sol#14

Descriptions:

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves 3406-3606 gas in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table.

```
bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
```

Suggestion:

It is recommended to use private rather than public for constants.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ARE-4 Use Calldata Instead of Memory for Function Arguments That Do not Get Mutated

Severity: Informational

Status: Fixed

Code Location:

contracts/AccountRegistrar.sol#23,39

Descriptions:

Mark data types as `calldata` instead of memory where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies memory storage.

Suggestion:

It is recommended to use `calldata` instead of `memory`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BDI-1 State Variables Should Be Cached in Stack Variables Rather Than Re-reading Them from Storage

Severity: Minor

Status: Fixed

Code Location:

contracts/earn/BonusDistributor.sol#121

Descriptions:

In Solidity, state variables ought to be cached in stack variables rather than repeatedly reading them from storage.

```
IERC20(rewardToken).safeTransfer(msg.sender, amount);
```

Suggestion:

It is recommended to cache state variables in local variables within functions to reduce repetitive reads from storage in Solidity.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BTO-1 `approve()` and `transferFrom()` Functions are Subject to Front-run attack

Severity: Medium

Status: Acknowledged

Code Location:

contracts/earn/BaseToken.sol#168;

contracts/earn/BaseToken.sol#176

Descriptions:

The `BaseToken.approve()` method overwrites the current allowance regardless of whether the spender already used it or not. It allows the spender to front-run and spend the amount before the new allowance is set.

Scenario:

- Alice allows Bob to transfer N of Alice's tokens (N>0) by calling the pool.approve method, passing the Bob's address and N as the method arguments
- After some time, Alice decides to change from N to M (M>0) the number of Alice's tokens Bob is allowed to transfer, so she calls the pool.approve method again, this time passing the Bob's address and M as the method arguments
- Bob notices the Alice's second transaction before it was mined and quickly sends another transaction that calls the pool.transferFrom method to transfer N Alice's tokens somewhere
- If the Bob's transaction will be executed before the Alice's transaction, then Bob will successfully transfer N Alice's tokens and will gain an ability to transfer another M tokens Before Alice noticed that something went wrong, Bob calls the pool.transferFrom method again, this time to transfer M Alice's tokens.
- So, an Alice's attempt to change the Bob's allowance from N to M (N>0 and M>0) made it possible for Bob to transfer N+M of Alice's tokens, while Alice never wanted to allow so many of her tokens to be transferred by Bob.

Suggestion:

It is recommended to use increaseAllowance and decreaseAllowance instead of approve as OpenZeppelin ERC20 implementation. Please see the details here:

https://forum.openzeppelin.com/t/explain-the-practical-use-of-increaseallowance-and-decreaseallowance-functions-on-erc20/15103/4

BTO-2 When the Current Allowance is Set to `type(uint256).max`, There is No Need to Update the Allowance

Severity: Medium

Status: Fixed

Code Location:

contracts/earn/BaseToken.sol#176-192

Descriptions:

In the `BaseToken.transferFrom()` function, it checks if the `_amount` is within the allowed limit based on the current allowance.If the check passes, it subtracts the transferred amount from the allowance.It then updates the allowance with the new value.Finally, it transfers the specified amount from the sender to the recipient.

```
uint256 nextAllowance = allowances[_sender][msg.sender].sub(
      _amount,
      "BaseToken: transfer amount exceeds allowance"
    );
    _approve(_sender, msg.sender, nextAllowance);
    _transfer(_sender, _recipient, _amount);
    return true;
```

However, adhering to OpenZeppelin's best practices, if the current allowance is set to the maximum value (`type(uint256).max`), there is no necessity to update the allowance. An allowance set to the maximum value effectively indicates an unlimited allowance, rendering any further updates redundant.

Suggestion:

It is recommended to refer to OpenZeppelin's best practices.

Resolution:

This issue has been fixed. The client followed OpenZeppelin's best practices.

EBO-1 Use Custom Errors

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryOptions.sol#187,189

Descriptions:

Instead of using error strings, to reduce deployment and runtime costs, you should use Custom Errors. This would save both deployment and runtime costs.Source

```
require(optionID < nextTokenId, "O10");

require(option.state == State.Active, "O5");
```

Suggestion:

It is recommended to use Custom Errors.

```
require(optionID < nextTokenId, "Invalid OptionID");
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EBO-2 `++i` Costs Less Gas Than `i++`, Especially When It's Used in For-loops (`--i/i--` Too)

Severity: Minor

Status: Acknowledged

Code Location:

contracts/EthoraBinaryOptions.sol#324

Descriptions:

The gas cost of `++i` is lower than `i++`, particularly when utilized in for-loops (`--i/i--` as well).

Suggestion:

It is recommended to use `++i` instead of `i++`.

EBO-3 Use Shift Right/Left instead of Division/Multiplication if Possible

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryOptions.sol#135

Descriptions:

A division/multiplication by any number x being a power of 2 can be calculated by shifting log2(x) to the right/left.

While the DIV opcode uses 5 gas, the SHR opcode only uses 3 gas. Furthermore, Solidity's division operation also includes a division-by-0 prevention which is bypassed using shifting.

```
optionParams.amount / 2,
```

Suggestion:

It is recommended to use shift Right/Left instead of division/multiplication.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EBO-4 Use `!= 0` instead of `> 0` for Unsigned Integer Comparison

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryOptions.sol#299,423

Descriptions:

When dealing with unsigned integer types, comparisons with `!= 0` are cheaper than with `> 0`.

```
require(maxTradeSize > 0, "O36");

if (referrerFee > 0)
```

Suggestion:

It is recommended to use `!= 0` instead of `> 0` for unsigned integer comparison.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EBO-5 Functions Not Used Internally Could be Marked External

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryOptions.sol#100,107,224,269,485,491

Descriptions:

In Solidity, functions that are not internally used within the contract can be more suitably marked as external. This ensures clarity in code and signifies that these functions are intended to be accessed from outside the contract by other contracts or external entities.

```
function approvePoolToTransferTokenX() public {

function setIsPaused() public {

function fees(

function getMaxOI() public view returns (uint256) {

function approveAddress(

function setToken(
```

Suggestion:

It is recommended to consider marking functions that are not internally used within the contract as external to enhance code readability and explicitly indicate that these functions are meant to be accessed externally. This practice provides clarity and aligns with the intended usage of the functions, making the contract interface more understandable for external interactions.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EBP-1 Use `_grantRole()` instead of `_setupRole()`

Severity: Major

Status: Fixed

Code Location:

contracts/EthoraBinaryPool.sol#47

Descriptions:

Access Control's `_setupRole()` has been deprecated in favor of `_grantRole()`, which performs the same operations.

https://docs.openzeppelin.com/contracts/4.x/api/access#AccessControl-_setupRole-bytes32-address-

However, it is still utilized in the `EthoraBinaryPool.initialize()` function.

```
function initialize(
    address _tokenX,
    uint32 _lockupPeriod
) external initializer {
    __ERC20_init("Ethora LP Token", "ELP");
    ACCURACY = 1e3;
    INITIAL_RATE = 1;
    OPTION_ISSUER_ROLE = keccak256("OPTION_ISSUER_ROLE");
    tokenX = ERC20Upgradeable(_tokenX);
    owner = msg.sender;
    maxLiquidity = 5000000 * 10 ** tokenX.decimals();
    lockupPeriod = _lockupPeriod;
    _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
}
```

Suggestion:

It is recommend to use `_grantRole()` instead of `_setupRole()`.

Resolution:

This issue has been fixed. The client used `_grantRole()` instead of `_setupRole()`.

EBP-2 User Boost will not be Updated

Severity: Major

Status: Fixed

Code Location:

contracts/EthoraBinaryPool.sol#165-169

Descriptions:

The function `getBoostPercentage()` is used to retrieve the boost percentage for a specific user and token. In the function,it checks whether the total number of boost trades (`totalBoostTrades`) for the user is greater than the total boost trades already used (`totalBoostTradesUsed`). If this condition is met, the function returns the boost percentage (`boostPercentage`). Otherwise, it returns 0.

```
function getBoostPercentage(
    address user,
    address token
) external view override returns (uint256) {
    UserBoostTrades memory userBoostTrade = userBoostTrades[token][user];
    if (
        userBoostTrade.totalBoostTrades >
        userBoostTrade.totalBoostTradesUsed
    ) {
        return boostPercentage;
    } else return 0;
}
```

However, in the protocol, `userBoostTrade.totalBoostTrades` is initialized to 0 by default, and there are no other places where it is updated. Therefore, this function will always return 0. Consequently, the `booster.updateUserBoost()` function will not be called in the `createFromRouter()` function for updates.

```
IBooster booster = IBooster(config.boosterContract());
    if (
        booster.getBoostPercentage(optionParams.user, address(tokenX)) > 0
    ) {
```

```
        booster.updateUserBoost(optionParams.user, address(tokenX));
    }
```

Suggestion:

It is recommended to implement updates for `userBoostTrade.totalBoostTrades` in the corresponding functions based on business logic.

Resolution:

This issue has been fixed. The client deleted the relevant code.

EBP-3 Unused Variables

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryPool.sol#22;

contracts/ReferralStorage.sol#16;

contracts/ReferralStorage.sol#22

Descriptions:

The variable in question serves no purpose as it is not utilized anywhere in the contract, leading to code redundancy and additional gas expenditure.

```
uint16 public ACCURACY;
```

```
mapping(uint8 => Tier) public tiers;
mapping(address => ReferralData) public UserReferralData;
```

Suggestion:

It is recommended to remove the variables or utilize them in the code.

Resolution:

This issue has been fixed. The client deleted the unnecessary variables.

EBP-4 Unused Import of `SafeMathUpgradeable`

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryPool.sol#8

Descriptions:

In this smart contract, the `SafeMathUpgradeable` library is imported, but none of its functions are actually used, potentially leading to code redundancy and unnecessary gas cost.

```
import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
```

Suggestion:

It is recommended to either remove unnecessary imports or utilize the functionalities of the imported libraries.

Resolution:

This issue has been fixed. The client deleted the unnecessary imports.

EBP-5 Using Bools for Storage Incurs Overhead

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryPool.sol#29

Descriptions:

Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from `false` to `true`, after having been `true` in the past. See source.

```
mapping(address => bool) public isHandler;
```

Suggestion:

It is recommended to use `uint256(1)` and `uint256(2)` for true/false.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EBP-6 Functions Guaranteed To Revert When Called By Normal Users Can be Marked Payable

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryPool.sol#60,192

Descriptions:

If a function modifier such as onlyOwner is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are CALLVALUE(2),DUP1(3),ISZERO(3),PUSH2(3),JUMPI(10),PUSH1(3),DUP1(3),REVERT(0),JUMPDEST(1),PO which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost.

Suggestion:

It is recommended that functions guaranteed to revert when called by normal users can be marked payable.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EBP-7 `require()` / `revert()` Statements Should Have Descriptive Reason Strings

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraBinaryPool.sol#211,429

Descriptions:

In Solidity, it's essential to include descriptive reason strings within `require()` or `revert()` statements.

```
require(to != address(0));
require(b > 0);
```

Suggestion:

It is recommended to add reason strings to `require()` or `revert()` .

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EBP-8 Long Revert Strings

Severity: Informational

Status: Fixed

Code Location:

contracts/EthoraBinaryPool.sol#210,343

Descriptions:

The error message string provided in the `require` function is too long and exceeds the EVM limit.

```
require(ll.locked, "Pool: lockedAmount is already unlocked");
```

Suggestion:

It is recommended to shorten the error message.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ERO-1 Blacklisted User can Block the `openTrades()`

Severity: Major

Status: Fixed

Code Location:

contracts/EthoraRouter.sol#554-555

Descriptions:

In the open trade process, the protocol transfers token from users. However, certain tokens, like USDC, have blacklists that prevent users from sending or receiving tokens. If a user is blacklisted, the protocol's attempt to call `transferFrom()` and withdraw funds from the user will fail, disrupting the protocol's operation and compromising its atomicity

```
ERC20Upgradeable tokenX = ERC20Upgradeable(optionsContract.tokenX());

tokenX.safeTransferFrom(user, admin, config.platformFee());
tokenX.safeTransferFrom(user, params.targetContract, revisedFee);
```

Suggestion:

It is recommended to add a try-catch block during token transfers.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ERO-2 Cache Array Length Outside of Loop

Severity: Minor

Status: Fixed

Code Location:

contracts/EthoraRouter.sol#122,161,232

Descriptions:

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

```
for (uint256 index = 0; index < revokeParams.length; index++) {

for (uint32 index = 0; index < params.length; index++) {

for (uint32 index = 0; index < closeParams.length; index++) {
```

Suggestion:

It is recommended to cache the array length before entering the loop.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ERO-3 Don't Initialize Variables with Default Value

Severity: Informational

Status: Fixed

Code Location:

contracts/EthoraRouter.sol#122,161

Descriptions:

Uninitialized variables are assigned with the types' default value. Explicitly initializing a variable with its default value costs unnecessary gas.

```
for (uint256 index = 0; index < revokeParams.length; index++) {

for (uint32 index = 0; index < params.length; index++) {
```

Suggestion:

It is recommended not to use default values to initialize variables.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

FAU-1 Single-step Ownership Transfer Can be Dangerous

Severity: Major

Status: Fixed

Code Location:

contracts/Faucet.sol#8;

contracts/EthoraRouter.sol#17;

contracts/earn/Governable.sol#5

Descriptions:

`Faucet` inherits from the `Ownable` contract. This contract does not implement a 2-Step-Process for transferring ownership. So ownership of the contract can easily be lost when making a mistake when transferring ownership.

```
contract Faucet is Ownable {
    USDC public token;
    uint256 public amount;
    uint256 public startTimestamp;
    uint256 public fee = 1e15; // 0.001 ETH
    address public fee_collector;
    mapping(address => uint256) public lastSavedTimestamp;
    mapping(bytes32 => bool) public previousHashedMessages;
```

Additionally, the modifications of the `admin` in `EthoraRouter.sol` and the `gov` in `Governable.sol` encounter the same issue.

Suggestion:

It is recommended to use the Ownable2Step contract from OZ (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol) instead.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

OST-1 The Purpose of the save() Function

Severity: Medium

Status: Fixed

Code Location:

contracts/OptionStorage.sol#13-19

Descriptions:

The `OptionStorage.save()` function only emits an event, please confirm if there are any other purposes for its usage.

```
function save(
    uint256 optionId,
    address optionsContractAddress,
    address user
  ) external {
    emit Save(optionId, optionsContractAddress, user);
  }
```

Suggestion:

It is recommended to remove the `OptionStorage` contract and emit the event directly.

Resolution:

This issue has been fixed. The client followed our advice.

POI-1 Event is Missing Indexed Fields

Severity: Minor

Status: Fixed

Code Location:

contracts/PoolOIStorage.sol#10

Descriptions:

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

Suggestion:

It is recommended to evaluate the necessity of indexing fields based on the specific needs of off-chain tools and gas usage. For events with three or more fields where gas consumption isn't critical, consider indexing up to three relevant fields. For events with fewer than three fields, index all available fields to optimize off-chain tool accessibility. Regularly assess and adjust indexing strategies based on the requirements of off-chain tools and gas efficiency.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TSA-1 `abi.encodePacked()` Should Not be Used with Dynamic Types When Passing the Result to a Hash Function Such as `keccak256()`

Severity: Minor

Status: Fixed

Code Location:

contracts/tokenSale/TokenSale.sol#821

Descriptions:

Use `abi.encode()` instead which will pad items to 32 bytes, which will prevent hash collisions (e.g. `abi.encodePacked(0x123,0x456)` => `0x123456` => `abi.encodePacked(0x1,0x23456)`, but `abi.encode(0x123,0x456)` => `0x0...1230...456`). "Unless there is a compelling reason, `abi.encode` should be preferred". If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` instead. If all arguments are strings and or bytes, `bytes.concat()` should be used instead.

Suggestion:

It is recommended to modify it according to the description.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.