TrustGO
Audit Report

Tue Apr 09 2024

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

ScaleBit

TrustGO Audit Report

1 Executive Summary

1.1 Project Information

Description	groundbreaking market contract
Type	DeFi
Auditors	ScaleBit
Timeline	Fri Mar 29 2024 - Thu Apr 04 2024
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review

1.2 Files in Scope

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

ID	File	SHA-1 Hash
PMA	PreMarket.sol	ec92582c9afcf7e457d6f4ee4ac249661647e37a
TMA	Tmarket.sol	abcf939523efbe9fcb2f3241b525e225731db641

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	4	3	1
Informational	1	0	1
Minor	2	2	0
Medium	0	0	0
Major	1	1	0
Critical	0	0	0

1.4 MoveBit 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 TrustGO to identify any potential issues and vulnerabilities in the source code of the TrustGO 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 4 issues of varying severity, listed below.

ID	Title	Severity	Status
PMA-1	Excess Funds Not Returned	Major	Fixed
PMA-2	Calculated Risk	Minor	Fixed
PMA-3	Missing Decimals Check	Minor	Fixed
PMA-4	Centralization Risk	Informational	Acknowledged

3 Participant Process

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

Deployer

- Deployer can initialize `adminAddress` and `feeWalletAddress` through the `constructor` function, and set `DEFAULT_ADMIN_ROLE` , `OPERATOR_ROLE` and initialize config.

Admin

- Admin can set the Role for the specified address through the `grantRole` function.

Operator

- Operators can create new ACTIVE tokens through the `createToken` function.
- Operators can set the token and `SettleRate` through the `tokenToSettlePhase` function.
- Operator can switch the status of Token to `active/inactive` through `tokenToSettlePhase` function
- Operator can force the token in the `SETTLE` state to `inactive` through the `tokenForceCancelSettlePhase` function.
- Operator can update `settleDuration` through `updateSettleDuration` function.
- Operator can forcefully close the order and return the funds through the `forceCancelOrder` function.
- Operator can forcefully close orders in batches through the `forceCancelOrders` function.
- Operator can use the `settle2Steps` function to handle orders that cannot be directly settled by tokens.
- Operator can call the `settle2Steps` function in batches through the `settle2StepsBatch` function.
- Operator can update `feeSettle` , `feeWallet` , `feeRefund` , `pledgeRate` through the `updateConfig` function.
- Operator can set the platform's whitelist token address through the `setAcceptedTokens` function.

- Operator can use the `withdrawStuckToken` function to withdraw the ERC20 Token balance in the contract that is not `acceptedTokens`.

User

- Users can create offers to buy or sell tokens through the `newOffer` function, and hand over the funds to contract management.
- Users can specifically create Offers traded in ETH through the `newOfferETH` function.
- Users can partially or fully fulfill existing offers through the `fillOffer` function, specify tokens to be managed by the contract, and update quotes and status.
- Users can specifically create Orders for ETH transactions through the `fillOfferETH` function.
- Users can create Offers in batches through the `fillOffers/fillOffersETH` function.
- Users can close the offer through the `cancelOffer` function and return the funds after deducting the handling fee.
- Users can use the `settleFilled` function to transfer trading tokens and release locked funds or collateral after the transaction is executed, and transfer the designated tokens from the seller to the buyer to complete the settlement of the completed order.
- Users can call the `settleFilled` function in batches through the `settleFilleds` function.
- Users (buyer) can cancel settlement through the `settleCancelled` function and return the collateral to the buyer.
- Users can call the `settleCancelled` function in batches through the `settleCancelleds` function.

4 Findings

PMA-1 Excess Funds Not Returned

Severity: Major

Status: Fixed

Code Location:

PreMarket.sol#870

Descriptions:

The excess funds invested by the user into the contract were not returned, Located at `newOfferETH` , `fillOfferETH` , `fillOffersETH` .

```
require(_ethAmount <= msg.value, "Insufficient Funds");
...
require(msg.value >= totalEthAmount, "Insufficient Funds");
```

Suggestion:

It is recommended to return excess funds.

Resolution:

have added refund logic in the `newOfferETH` , `fill0fferETH` , and `fillOffersETH` functions.

```
    // (PMA-1)Refund excess ETH
    if (msg.value > _ethAmount) {
        uint256 excessAmount = msg.value - _ethAmount;
        (bool refundSuccess, ) = msg.sender.call{value: excessAmount}("");
        require(refundSuccess, "Refund of excess ETH failed");
    }
```

PMA-2 Calculated Risk

Severity: Minor

Status: Fixed

Code Location:

PreMarket.sol#433,516

Descriptions:

1. The variable `pledgeRate` is set through the function `updateConfig`. OPERATOR can set `pledgeRate` arbitrarily. When creating an offer, risks such as over-collateralization may occur when calculating collateral. At the same time, when transferring, since the calculation of collateral depends on pledgeRate, the amount of _transferAmount is unexpected.

```
uint256 collateral = (value * config.pledgeRate) / WEI6;
...
_transferAmount = (offer.collateral * amount) / offer.amount;
```

2. When calculating the matching of different decimals tokens in the contract during the settlement phase, `tokenAmount` depends on the setting of OPERATOR in the `tokenToSettlePhase` function, and there may be calculation risks caused by OPERATOR.

```
uint256 tokenAmount = (order.amount * token.settleRate) / WEI6;
```

Suggestion:

It is recommended that given a setting range, the expectation is that it is divisible.

Resolution:

The official confirmed the mortgage rate range and made restrictions. In addition, the impact of settleRate on exchange will be determined by OPERATOR.

```
require(pledgeRate_ >= WEI6 / 100 && pledgeRate_ <= WEI6, "Pledge Rate out of range");
```

PMA-3 Missing Decimals Check

Severity: Minor

Status: Fixed

Code Location:

PreMarket.sol#457

Descriptions:

Lack of decimals check. When creating an offer, there may be inconsistencies in the decimals of the corresponding currency. Since `newOfferETH` is recorded with the value `amount` instead of msg.value, it may cause problems, for example, ETH and USDC, because the decimals of USDC are 6 , the decimals of ETH is 18. When calculating `_ethAmount` , assume pledgeRate/WEI6=1, then `msg.value` only needs to be greater than x*1e6, but the `_newOffer` record value is the parameter amount.

```
require(_ethAmount <= msg.value, "Insufficient Funds");
```

Suggestion:

It is recommended to take measures to mitigate this issue.

Resolution:

Through `convertDecimalsCeil` , `convertDecimals` function ensures accuracy when handling tokens of different precisions.

```
uint8 exTokenDecimals;
if (offer.exToken == address(0)) {
exTokenDecimals = 18;
} else {
exTokenDecimals = IERC20Metadata(offer.exToken).decimals();
}
uint256 collateral = convertDecimals(
(order.amount * offer.collateral) / offer.amount,
18,
exTokenDecimals
);
uint256 value = convertDecimals(
```

```
(order.amount * offer.value) / offer.amount,
18,
exTokenDecimals
);
```

PMA-4 Centralization Risk

Severity: Informational

Status: Acknowledged

Code Location:

PreMarket.sol#1149

Descriptions:

Centralization risk was identified in the smart contract.

OPERATOR can withdraw funds from the contract through the `withdrawStuckToken` function, and OPERATOR can modify `acceptedTokens`.

```
    function withdrawStuckToken(address _token,address _to) external
onlyRole(OPERATOR_ROLE) {
        require(_token != address(0) && !acceptedTokens[_token],"Invalid Token Address");
        uint256 _contractBalance = IERC20(_token).balanceOf(address(this));
        IERC20(_token).safeTransfer(_to, _contractBalance);
    }
```

Suggestion:

It is recommended to take measures to mitigate this issue.

Resolution:

The official description is reserved for emergency situations and will be used in the early stages of the contract.

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.