CERTIK

Security Assessment

Tranchess

Jun 2nd, 2021

Table of Contents

Summary

This report has been prepared for Tranchess smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Tranchess
Platform	Ethereum
Language	Solidity
Codebase	• https://github.com/tranchess/contract-exchange • https://github.com/tranchess/contract-core • https://github.com/tranchess/contract-oracle
Commits	• https://github.com/tranchess/contract-exchange/commit/a210bb72c5bccb7f837be83fb6f86c3caaf7068a • https://github.com/tranchess/contract-core/commit/240dbb2ce2ccdde046e3f18942b62a8e0b3a47c2 • https://github.com/tranchess/contract-oracle/commit/fbf3d9939c4ed0608e2f5213ebdb875634b400a9 • https://github.com/tranchess/contract-core/commit/1f2b867961d2d33a4ed004eec4eaa5791df3daab • https://github.com/tranchess/contract-core/commit/525366e32044ae22e006b34b717f1406287fd483 • https://github.com/tranchess/contract-core/commit/85a83d5d82183663e1a151fa5433cbaeaeb85827 • https://github.com/tranchess/contract-core/commit/dad1d78e76f1e0c941a0be1e61f8a3bcfc284b7a • https://github.com/tranchess/contract-core/commit/c27305daecfe936b4db5297a30bdf3439775abf5 • https://github.com/tranchess/contract-core/commit/eeae12713dbe6ad482929a62196f71d3679fb5c5 • https://github.com/tranchess/contract-core/commit/91aa9377950e0e6acd163a892a023ac0d18bac26 • https://github.com/tranchess/contract-exchange/commit/d84b79a082b00fbe3cdc2f13dd2e3bdda1269b7f • https://github.com/tranchess/contract-exchange/commit/17ac07f9357c18322f24a0c73731b7fb71b19ef6 • https://github.com/tranchess/contract-exchange/commit/616f4d03c103cf39631f1a6b20749348f935d3a7

Audit Summary

Delivery Date	Jun 02, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	exchange, core, oracle

Vulnerability Summary

Total Issues	26
● Critical	0
● Major	5
● Medium	5
● Minor	6
● Informational	10
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
AOC	contracts/core/AprOracle.sol	95658fffcc674f3c32c6d4f5dbbae7add128b9d71540dd3321923bdbeb5de930
FCK	contracts/core/Fund.sol	a141d6b29b3e8a3e760b6e10419adbf125b9ac58739ff756616ba1dda3c040f0
FRC	contracts/core/FundRoles.sol	56130a73d44aa5f4615ef4826936379a0ea01c099670d17c9c44b779bc03596e
CGC	contracts/core/Governance/Chess.sol	83a42f0d062cbb66a0ebbc54c15a00856f2d07c9566032b75a3b7a54ce13ac7c
CRG	contracts/core/Governance/ChessRoles.sol	be3b9e78271906ca006ccb565dba2b33d07623f27101ae6139a89413f9572284
IRB	contracts/core/Governance/InterestRateBallot.sol	834ef82dbcfeec7544102819463c0a73180c28ba274d4b10b5ea3d49d3568fcc
VEG	contracts/core/Governance/VotingEscrow.sol	281e52f5e5216b5e5f7b607b5be8523a650e5f7f048bd1d93d53cbd7c6e54e48
PMC	contracts/core/PrimaryMarket.sol	00866f36741e9b3953614c1dab1398649e6b9723847cc53a89158a4afb0b5292
SCK	contracts/core/Share.sol	32faa9be26c77966a7dab2a39be9a79db99d24703f2dc0cb92f56f87f2724840
IAO	contracts/core/interfaces/IAprOracle.sol	d370de10c021d6d5ba2c485b859d662f91c966848b92c8b3accbf019b23c2c96
IBC	contracts/core/interfaces/IBallot.sol	2f240b562dd45772b190a9f141d75186a9f50c6adf3449d7052f1d72eee22149
ICC	contracts/core/interfaces/IChess.sol	ec4e9bf78623c3bd64621e9660be1cb457a6da3e912b87c72acaa0187eb6bfc0
IFC	contracts/core/interfaces/IFund.sol	79bfd0daeda2712cecae008c04a4b325d1094bdb95904eaf462b7630222f6365
IPM	contracts/core/interfaces/IPrimaryMarket.sol	e3a1378eabd40b00b31c004eb5be324bd5ba629fc335acd008cf04f39e6dd1fd
ITC	contracts/core/interfaces/IToken.sol	06d77dcb43a4d881cd5148f9842de05abb81fb8e54e9bc745e4bca7243b97f17
ITI	contracts/core/interfaces/ITrancheIndex.sol	188de430a67d1f5089ac5b0f476bdafcd9bd0cc60e384b62cf5a316e3b336e39
ITO	contracts/core/interfaces/ITwapOracle.sol	b354f67cb26e7db0d81e56de84792cad5e9d434260c4f01af17fff25416c62f3
IVE	contracts/core/interfaces/IVotingEscrow.sol	cdeb100fd88107ce005489bbebffb0f6990b46ae619214e2dac0e407d2ddfdaa
MAO	contracts/core/test/MockAprOracle.sol	a3a665ff6ac8fed52e740959b72f77d1d4e1d30106cce7538b10a5942a8f7e43
MTC	contracts/core/test/MockToken.sol	609cc5f31fe6098593882fbcdfc0e82b7ff44b697a78a2e5be0eb5e706548a0c
MTO	contracts/core/test/MockTwapOracle.sol	7821c3a6f39e045d74680c826cb7f1b2332d72c70b4b0f00d9c145320bc92b14

ID	file	SHA256 Checksum
CMC	contracts/core/utils/CarefulMath.sol	674c707eddbe74daff8bed62dbfcc23e2cdcf129c3ef37697ca09f96e13394d5
ECK	contracts/core/utils/Exponential.sol	bff072433ae28ed2708fe4f417a66df73eb999f310a6eee13332a8c74c2aa3be
ENE	contracts/core/utils/ExponentialNoError.sol	5460d329de8405c30cf1cfbb29a6aa38e6aaefde4e2c81c87f214f2b1ba073f0
SDM	contracts/core/utils/SafeDecimalMath.sol	90002de648a1b160c40fd7c5faadd2a3476b59aec25ab227ac2bda4380d8fbc1
ECP	contracts/exchange/Exchange.sol	71010dde43247d5290cde633ae7a8794071220bb96fe6d4cb26def8364bc360d
EOB	contracts/exchange/ExchangeOrderBook.sol	a0a74ac020696e6cd42a7df63141d2dcd431ba05330e948a219f87facbf25b92
ERC	contracts/exchange/ExchangeRoles.sol	ff229f3a8a2ab13a698a16f1484106ec345722b2c2bfe5b7508c4359ec8be090
ETC	contracts/exchange/ExchangeTrade.sol	58b8a7aa6fbcbb5e05e1472f970edc95f6489ff066d638644f5d5ed43fa771da
SCP	contracts/exchange/Staking.sol	0456f881fd6b843c5777496c25092e747a59c6f2a41a37af27edb79f85e85dfc
ICK	contracts/exchange/interfaces/IChess.sol	ec4e9bf78623c3bd64621e9660be1cb457a6da3e912b87c72acaa0187eb6bfc0
IFK	contracts/exchange/interfaces/IFund.sol	79bfd0daeda2712cecae008c04a4b325d1094bdb95904eaf462b7630222f6365
ITK	contracts/exchange/interfaces/ITrancheIndex.sol	188de430a67d1f5089ac5b0f476bdafcd9bd0cc60e384b62cf5a316e3b336e39
ITP	contracts/exchange/interfaces/ITwapOracle.sol	b354f67cb26e7db0d81e56de84792cad5e9d434260c4f01af17fff25416c62f3
IVC	contracts/exchange/interfaces/IVotingEscrow.sol	cdeb100fd88107ce005489bbebffb0f6990b46ae619214e2dac0e407d2ddfdaa
CCC	contracts/exchange/rewards/ChessController.sol	bf55fe083f8fc64baf1ebb884ee2af8dc3601aa0604647c1b397fe3849d79832
MTK	contracts/exchange/test/MockToken.sol	609cc5f31fe6098593882fbcdfc0e82b7ff44b697a78a2e5be0eb5e706548a0c
STW	contracts/exchange/test/StakingTestWrapper.sol	dc8df2012377e058ff5ca5ce36be15181d036eaa9662d6aa4508df121360b773
SDC	contracts/exchange/utils/SafeDecimalMath.sol	90002de648a1b160c40fd7c5faadd2a3476b59aec25ab227ac2bda4380d8fbc1
TPC	contracts/exchange/utils/TranchessProxy.sol	7dda7175b7844831b23744977e03b5b7bf88de314de88a6a7fc4df9995cf7a81
TOC	contracts/oracle/TwapOracle.sol	c55ca5c29c8e7af6a9e8313671e4d2bf51e4b2dd444717ccd67315b21daff2c6

ID	file	SHA256 Checksum
IOC	contracts/oracle/interfaces/ITwapOracle.sol	b354f67cb26e7db0d81e56de84792cad5e9d434260c4f01af17fff25416c62f3

There are a few depending injection contracts or addresses in the current project:

`TOKEN`, `AAVE_LENDING_POOL`, `CTOKEN` and `_fund` for `AprOracle`;

`tokenP_`, `tokenA_`, `tokenB_` and `primaryMarket_` for `FundRoles` (we assume `tokenP_`, `tokenA_` and `tokenB_` are implemented by Share.sol and `primaryMarket_` is implemented by PrimaryMarket.sol);

`twapOracle_`, `tokenUnderlying_`, `tokenP_`, `tokenA_`, `tokenB_`, `aprOracle_`, `ballot_`, `primaryMarket_` and `governance_` for `Fund` (we assume `twapOracle_` is implemented by TwapOracle.sol; `tokenP_`, `tokenA_` and `tokenB_` are implemented by Share.sol; `aprOracle_` is implemented by AprOracle.sol; `ballot_` is implemented by InterestRateBallot.sol);

`fund_` for `PrimaryMarket` (we assume `fund_` is implemented by Fund.sol);

`fund_` for `Share` (we assume `fund_` is implemented by Fund.sol);

`_token` and `_checker` for `VotingEscrow`;

`votingEscrow_` and `fund_` for `InterestRateBallot` (we assume `votingEscrow_` is implemented by VotingEscrow.sol and `fund_` is implemented by Fund.sol);

`fund_`, `chess_`, `chessController_` and `quoteAssetAddress_` for `Staking` (we assume `fund_` is implemented by Fund.sol; `chess_` is implemented by Chess.sol; `chessController_` is implemented by `ChessController.sol`);

`votingEscrow_` for `ExchangeRoles` (we assume `votingEscrow_` is implemented by VotingEscrow.sol);

`fund_`, `chess_`, `chessController_`, `quoteAssetAddress_` and `votingEscrow_` for `Exchange` (we assume `fund_` is implemented by Fund.sol; `chess_` is implemented by Chess.sol; `chessController_` is implemented by `ChessController.sol`; `votingEscrow_` is implemented by VotingEscrow.sol);

`primarySource_` and `secondarySource_` for `TwapOracle`.

We assume these contracts are valid and non-vulnerable actors, and implementing proper logic to collaborate with the current project.

To set up project correctly, improve overall project quality and preserve the upgradability, the following roles, are adopted in the codebase:

Admin, is adopted to add other roles in contract `FundRoles` and `ChessRoles`;

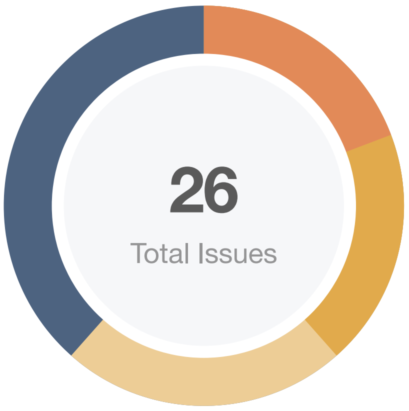
`PRIMARY_MARKET_ROLE`, is adopted to mint and burn tokens for accounts in contract `Fund`;

`MINTER_ROLE`, is adopted to mint CHESS for accounts in contract `Chess`;

Owner, is adopted to update prices if they are not previously set in contract `TwapOracle`.

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to invoke aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**5** (19.23%)
🟨	**Medium**	**5** (19.23%)
🟨	**Minor**	**6** (23.08%)
🟦	**Informational**	**10** (38.46%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
AOC-01	Variables Should Be Declared Constant	Language Specific	● Informational	✓ Resolved
CGC-01	Missing Check for Integer Overflow and Underflow	Mathematical Operations	● Minor	✓ Resolved
CRG-01	Centralization Risks	**Centralization / Privilege**	● **Major**	**Partially Resolved**
ECP-01	Required Validation on Quote Asset	Logical Issue	● Informational	ⓘ Acknowledged
ECP-02	Missing Check for Duplicated Identifier	Logical Issue	● Minor	✓ Resolved
ECP-03	Missing Check for Integer Overflow and Underflow	Mathematical Operations	● Medium	✓ Resolved
EOB-01	Missing Check for Integer Overflow and Underflow	Mathematical Operations	● Minor	✓ Resolved
ETC-01	Missing Check for Integer Overflow and Underflow	Mathematical Operations	● Medium	✓ Resolved
FCK-01	Missing Check for Reentrancy	Logical Issue	● Major	✓ Resolved
FCK-02	Missing Return Value Handling	Logical Issue	● Minor	✓ Resolved
FCK-03	Missing Check for Conversion Size	Logical Issue	● Informational	✓ Resolved
FCK-04	Required Validation on Underlying Asset	Logical Issue	● Informational	ⓘ Acknowledged
FCK-05	Reusable Code	Language Specific	● Informational	ⓘ Acknowledged

ID	Title	Category	Severity	Status
FRC-01	Centralization Risks	**Centralization / Privilege**	● **Major**	**Partially Resolved**
IRB-01	Reusable Code	Language Specific	● Informational	ⓘ Acknowledged
PMC-01	Missing Check for Reentrancy	Logical Issue	● Major	✓ Resolved
PMC-02	Missing Return Value Handling	Logical Issue	● Minor	✓ Resolved
PMC-03	Missing Check for History Creation Rate	Logical Issue	● Informational	✓ Resolved
SCK-01	Variable Should Be Declared Constant	Language Specific	● Informational	✓ Resolved
SCK-02	Redundant Conversions of Allowances	Gas Optimization	● Informational	✓ Resolved
SCP-01	Reusable Code	Language Specific	● Informational	ⓘ Acknowledged
SCP-02	Missing Check for Integer Overflow and Underflow	Mathematical Operations	● Medium	✓ Resolved
TOC-01	Centralization Risks	**Centralization / Privilege**	● **Major**	ⓘ **Acknowledged**
TOC-02	Missing Check for Integer Overflow and Underflow	Mathematical Operations	● Medium	✓ Resolved
VEG-01	Missing Return Value Handling	Logical Issue	● Minor	✓ Resolved
VEG-02	Missing Check for Integer Overflow and Underflow	Mathematical Operations	● Medium	✓ Resolved

AOC-01 | Variables Should Be Declared Constant

Category	Severity	Location	Status
Language Specific	● Informational	contracts/core/AprOracle.sol: 37, 40, 44	✓ Resolved

Description

Variables at the aforementioned line do not depend on any inputs nor change after assignments, so they should be declared `constant`.

Recommendation

We recommend declaring variables at the aforementioned lines `constant`.

Alleviation

The client heeded the advice and resolved this issue in commit `1f2b867961d2d33a4ed004eec4eaa5791df3daab` .

CGC-01 | Missing Check for Integer Overflow and Underflow

Category	Severity	Location	Status
Mathematical Operations	● Minor	contracts/core/Governance/Chess.sol: 86, 144, 167	✓ Resolved

Description

Integer overflow and underflow are not checked for integer operations at the aforementioned lines.

Recommendation

We recommend using `SafeMath` for integer operations.

Alleviation

[**Tranchess Team**]: It is unlikely to happen since the `Minter` role is guarded by Tranchess.

CRG-01 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	● Major	contracts/core/Governance/ChessRoles.sol: 42~44	Partially Resolved

Description

The role `MINTER_ROLE`, which is granted by the role admin, is allowed to mint token CHESS for `account` in the contract `Chess`. If there is an account other than contract `Staking` granted the role minter, it will be able to mint CHESS for any account and thus drain rewards which should be claimed by users with stakings.

Recommendation

We advise the client to only allow the contract `Staking` to be set as minter or carefully manage the project's private key and avoid any potential risks of being hacked. We also advise the client to adopt `Timelock` with reason and delay to add a new minter, Multisig with community-selected 3-party independent co-signers, and/or DAO with transparent governance with the project's community in the project to manage sensitive role accesses.

Alleviation

The client heeded the advice and added `Timelock` in commit `525366e32044ae22e006b34b717f1406287fd483`.

[**Tranchess Team**]: Add `Timelock` that wraps around OpenZeppelin's TimelockController implementation to delay `MINTER_ROLE` and `PRIMARY_MARKET_ROLE` related proposals and their execution. In the beginning, a centralized admin will assume the `Timelock`'s Proposer role, while no limitation on the Executor role. In the future, the Proposer role would be transferred to Governor contract for autonomous governance.

ECP-01 | Required Validation on Quote Asset

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/exchange/Exchange.sol: 114	ⓘ Acknowledged

Description

The quote asset implemented at address `quoteAssetAddress_`, which in an input of `constructor`, should be non-deflationary. Otherwise, it might lead to conflicts between recorded balance and real balance of quote asset.

Recommendation

We advise the client to carefully review the quote asset before adding it to the contract `Exchange`.

Alleviation

[**Tranchess Team**]: We understand the issue and confirm that the quote asset is USDC or similar stable coins. We will carefully review its contract and make sure that it is non-deflationary.

ECP-02 | Missing Check for Duplicated Identifier

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/exchange/Exchange.sol: 244, 293	✓ Resolved

Description

Calling `placeBid` or `placeAsk` with a existing combination of `conversionID`, `tranche`, `msg.sender` and `clientOrderID` will overwrite the existing identifier, and thus might lead to an incorrect cancellation when calling `cancelBidByClientOrderID` or `cancelAskByClientOrderID`.

Recommendation

We recommend checking if the identifier already exists given `conversionID`, `tranche`, `msg.sender` and `clientOrderID`.

Alleviation

The client heeded the advice and resolved this issue in commit `d84b79a082b00fbe3cdc2f13dd2e3bdda1269b7f` .

ECP-03 | Missing Check for Integer Overflow and Underflow

Category	Severity	Location	Status
Mathematical Operations	● Medium	contracts/exchange/Exchange.sol: 470, 486, 788, 791, 802, 805, 830, 833, 844, 847, 940, 948, 963, 965, 970	✓ Resolved

Description

Integer overflow and underflow are not checked for integer operations at the aforementioned lines.

Recommendation

We recommend using `SafeMath` for integer operations.

Alleviation

The client heeded the advice and resolved this issue in commit `616f4d03c103cf39631f1a6b20749348f935d3a7`.

EOB-01 | Missing Check for Integer Overflow and Underflow

Category	Severity	Location	Status
Mathematical Operations	● Minor	contracts/exchange/ExchangeOrderBook.sol: 41, 43, 68	✓ Resolved

Description

Integer overflow and underflow are not checked for integer operations at the aforementioned lines.

Recommendation

We recommend using `SafeMath` for integer operations.

Alleviation

The client heeded the advice and resolved this issue in commit `17ac07f9357c18322f24a0c73731b7fb71b19ef6` .

ETC-01 | Missing Check for Integer Overflow and Underflow

Category	Severity	Location	Status
Mathematical Operations	● Medium	contracts/exchange/ExchangeTrade.sol: 40~42, 57~59	✓ Resolved

Description

Integer overflow and underflow are not checked for integer operations at the aforementioned lines.

Recommendation

We recommend using `SafeMath` for integer operations.

Alleviation

The client heeded the advice and resolved this issue in commit `616f4d03c103cf39631f1a6b20749348f935d3a7`.

FCK-01 | Missing Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Major	contracts/core/Fund.sol: 667	✓ Resolved

Description

Function `settle` has state updates and event emits after external calls and thus are vulnerable to reentrancy attack.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

The client heeded the advice and resolved this issue in commit `85a83d5d82183663e1a151fa5433cbaeaeb85827` .

FCK-02 | Missing Return Value Handling

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/core/Fund.sol: 727, 753, 759, 765	✓ Resolved

Description

`transfer` and `transferFrom` are not void-returning functions per IERC20 interface. Ignoring the return value might cause some unexpected exception, especially if the callee function does not revert when failing.

Recommendation

We recommend checking return values of `transfer` and `transferFrom` before continuing processing.

Alleviation

The client heeded the advice and resolved this issue in commit `dad1d78e76f1e0c941a0be1e61f8a3bcfc284b7a`.

FCK-03 | Missing Check for Conversion Size

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/core/Fund.sol: 806	✓ Resolved

Description

According to line 89, conversion size should be smaller than 65535.

Recommendation

We recommend adding a check to ensure conversion size will always be smaller than 65535.

Alleviation

[Tranchess Team]: Conversion gets checked once per day, and triggers very rarely. Even if it gets triggered every day, 65535 will take about 180 years.

FCK-04 | Required Validation on Underlying Asset

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/core/Fund.sol: 151	ⓘ Acknowledged

Description

The underlying asset implemented at address `tokenUnderlying_`, which in an input of `initialize`, should be non-deflationary. Otherwise, it might lead to conflicts between recorded balance and real balance of underlying asset.

Recommendation

We advise the client to carefully review the underlying asset before adding it to the contract `Fund`.

Alleviation

[**Tranchess Team**]: We understand the issue and confirm that the underlying asset is WBTC, WETH, or similar mainstream tokens. We will carefully review its contract and make sure that it is non-deflationary.

FCK-05 | Reusable Code

Category	Severity	Location	Status
Language Specific	● Informational	contracts/core/Fund.sol: 492~511	ⓘ Acknowledged

Description

In `Staking.availableBalanceOf`, `Staking.lockedBalanceOf` and `Fund.shareBalanceOf`, calculations of amount after conversion are exactly the same. It is recommended to keep the code DRY by extracting the same logic and reusing the code.

Recommendation

We recommend implementing a new function to perform the amount calculation and call it in the aforementioned functions.

Alleviation

[**Tranchess Team**]: Arguably the code could reuse batchConvert logic, but given its low severity, we decide not to fix this issue in the audited version.

FRC-01 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	contracts/core/FundRoles.sol: 55~57	**Partially Resolved**

Description

The role `PRIMARY_MARKET_ROLE`, which is granted by the role admin, is allowed to mint and burn tokens in the contract `Fund`.

Recommendation

We advise the client to only allow the contract `Fund` to be set as `PRIMARY_MARKET_ROLE` or carefully manage the project's private key and avoid any potential risks of being hacked. We also advise the client to adopt `Timelock` with reason and delay to set a new primary market, Multisig with community-selected 3-party independent co-signers, and/or DAO with transparent governance with the project's community in the project to manage sensitive role accesses.

Alleviation

The client heeded the advice and added `Timelock` in commit `525366e32044ae22e006b34b717f1406287fd483`.

[**Tranchess Team**]: Add `Timelock` that wraps around OpenZeppelin’s TimelockController implementation to delay `MINTER_ROLE` and `PRIMARY_MARKET_ROLE` related proposals and their execution. In the beginning, a centralized admin will assume the `Timelock`’s Proposer role, while no limitation on the Executor role. In the future, the Proposer role would be transferred to the Governor contract for autonomous governance.

IRB-01 | Reusable Code

Category	Severity	Location	Status
Language Specific	● Informational	contracts/core/Governance/InterestRateBallot.sol: 51, 89	ⓘ Acknowledged

Description

The same check is performed twice at the aforementioned lines. It is recommended to keep the code DRY by extracting the same logic and reusing the code.

Recommendation

We recommend implementing a modifier and use it to perform the check.

Alleviation

[**Tranchess Team**]: Given its low severity, we decide not to fix this issue in the audited version.

PMC-01 | Missing Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Major	contracts/core/PrimaryMarket.sol: 92, 119	✓ Resolved

Description

Function `create` and `claim` has state updates and event emits after external calls and thus are vulnerable to reentrancy attack.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

The client heeded the advice and resolved this issue in commit `85a83d5d82183663e1a151fa5433cbaeaeb85827`.

PMC-02 | Missing Return Value Handling

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/core/PrimaryMarket.sol: 94, 122, 126, 267	✓ Resolved

Description

`transfer`, `transferFrom` and `approve` are not void-returning functions per IERC20 interface. Ignoring the return value might cause some unexpected exception, especially if the callee function does not revert when failing.

Recommendation

We recommend checking the return values of `transfer`, `transferFrom` and `approve` before continuing processing.

Alleviation

The client heeded the advice and resolved this issue in commit `dad1d78e76f1e0c941a0be1e61f8a3bcfc284b7a`.

[**Tranchess Team**]: The callee at PrimaryMarket.sol L#123 is `tokenP`, which is part of Tranchess Protocol, and we know for sure that the callee function revert when failing.

PMC-03 | Missing Check for History Creation Rate

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/core/PrimaryMarket.sol: 304, 323	✓ Resolved

Description

The correctness of calculations at the aforementioned lines depends on the daily update of `_historyCreationRate` in `settle`. Checking `_historyCreationRate[oldDay]` is non-zero would be helpful to avoid errors if `settle` is not called correctly.

Recommendation

We recommend checking `_historyCreationRate[oldDay]` is non-zero to ensure it is correctly updated.

Alleviation

[**Tranchess Team**]: Note that `_creationRedemptions[account].day` can only be updated in `_currentCreationRedemption()` at L#338 and written to storage at L#345, so after every time `_creationRedemptions[account]` is modified, `_creationRedemptions[account].day` is the same as `currentDay` at that time.

If `settle` is not called, `currentDay` is not modified and the condition at L#312 (`oldDay < currentDay`) will never meet, and thus no indexing in `_historyCreationRate` with `cr.day`.

At a high level, code at L#324~349 settles creations and redemptions from an account on `cr.day` according to settlement on that day. `cr.day` stores the last trading day when there's some creations or redemptions, and amount of these creations and redemptions are stored in `cr.creatingUnderlying` and `cr.redeemingShares`. `_historyXxxRate[cr.day]` of a specific `cr.day` is used only once when settle is called (so that `currentDay` grows beyond `cr.day`) and a new creation or redemption comes the first time.

SCK-01 | Variable Should Be Declared Constant

Category	Severity	Location	Status
Language Specific	● Informational	contracts/core/Share.sol: 36	✓ Resolved

Description

Variable `decimals` does not depend on input not change after assignment, so it should be declared `constant`.

Recommendation

We recommend declaring `decimals` `constant` and set it to 18 at definition.

Alleviation

The client heeded the advice and resolved this issue in commit `eeae12713dbe6ad482929a62196f71d3679fb5c5`.

SCK-02 | Redundant Conversions of Allowances

Category	Severity	Location	Status
Gas Optimization	● Informational	contracts/core/Share.sol: 113~119, 138~139, 163~169	✓ Resolved

Description

The conversions of allowances from an old version to the latest version are conducted in both of `fund.shareAllowance` and `fund.approve`. Redundant calculations cost unnecessary gas.

Recommendation

The client heeded the advice and resolved this issue in commit `c27305daecfe936b4db5297a30bdf3439775abf5`.

SCP-01 | Reusable Code

Category	Severity	Location	Status
Language Specific	● Informational	contracts/exchange/Staking.sol: 121~147, 155~181	ⓘ Acknowledged

Description

In `Staking.availableBalanceOf`, `Staking.lockedBalanceOf` and `Fund.shareBalanceOf`, calculations of amount after conversion are exactly the same. It is recommended to keep the code DRY by extracting the same logic and reusing the code.

Recommendation

We recommend implementing a new function to perform the amount calculation and call it in the aforementioned functions.

Alleviation

[**Tranchess Team**]: Both `Staking.availableBalanceOf` and `Staking.lockedBalanceOf` are already using the same `fund.batchConvert`, which encapsulate the conversion logics.

SCP-02 | Missing Check for Integer Overflow and Underflow

Category	Severity	Location	Status
Mathematical Operations	● Medium	contracts/exchange/Staking.sol: 497, 515	✓ Resolved

Description

Integer overflow and underflow are not checked for integer operations at the aforementioned lines.

Recommendation

We recommend using `SafeMath` for integer operations.

Alleviation

The client heeded the advice and resolved this issue in commit `616f4d03c103cf39631f1a6b20749348f935d3a7`.

TOC-01 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	contracts/oracle/TwapOracle.sol: 246	ⓘ **Acknowledged**

Description

The role owner is allowed to set price for an epoch if it is not previously set. The newly set price will be in the range (p/10, 10p), where p is price of the previous epoch.

Recommendation

We advise the client to carefully manage the project's private key and avoid any potential risks of being hacked. We also advise the client to adopt `Timelock` with reason and delay to allow the owner to update prices, Multisig with community-selected 3-party independent co-signers, and/or DAO with transparent governance with the project's community in the project to manage sensitive role accesses.

Alleviation

[**Tranchess Team**]: `TwapOracle` by design exposes the elevated operation. Failure to update the oracle price would paralyze the entire system, and thus we decide to deal with the centralization risk to overcome the paralysis risk.

TOC-02 | Missing Check for Integer Overflow and Underflow

Category	Severity	Location	Status
Mathematical Operations	● Medium	contracts/oracle/TwapOracle.sol: 227, 237, 239, 258	✓ Resolved

Description

Integer overflow and underflow are not checked for integer operations at the aforementioned lines.

Recommendation

We recommend using `SafeMath` for integer operations.

Alleviation

[**Tranchess Team**]: For TwapOracle.sol, gas cost is a big deal because at least one transaction is needed every 30 minutes. The contract is carefully optimized for gas cost. As explained at L#198, only the low 64 bits of prices are used, which guarantees that all price-related variables never exceed `(2^64 - 1) * MESSAGE_BATCH_SIZE * PRICE_UNIT`. So, the aforementioned operations never overflow.

VEG-01 | Missing Return Value Handling

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/core/Governance/VotingEscrow.sol: 133, 193	✓ Resolved

Description

`transfer` and `transferFrom` are not void-returning functions per IERC20 interface. Ignoring the return value might cause some unexpected exception, especially if the callee function does not revert when failing.

Recommendation

We recommend checking the return values of `transfer` and `transferFrom` before continuing processing.

Alleviation

[Tranchess Team]: `VotingEscrow`'s "token" is always set to `Chess`, which is an internal ERC20 token with known behavior to revert when failed.

VEG-02 | Missing Check for Integer Overflow and Underflow

Category	Severity	Location	Status
Mathematical Operations	● Medium	contracts/core/Governance/VotingEscrow.sol: 15	✓ Resolved

Description

Integer overflow and underflow are not checked for integer operations in contract `VotingEscrow`.

Recommendation

We recommend using `SafeMath` for integer operations.

Alleviation

The client heeded the advice and resolved this issue in commit `91aa9377950e0e6acd163a892a023ac0d18bac26`.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK