CERTIK

Preliminary Comments

RociFi

Feb 10th, 2022

Table of Contents

Summary

Overview

- Project Summary
- Audit Summary
- Vulnerability Summary
- Audit Scope

Findings

- BBR-01 : Compiler Error
- BBR-02 : Missing Interface Inheritance
- CMM-01 : Struct Tight-Packing
- CMM-02 : `_amount` Can Equal Zero
- CMM-03 : Centralization Related Risks In `CollateralManager.sol`
- CRF-01 : Variable Declare as Immutable
- CRF-02 : Centralization Related Risks In `ERC20PaymentStandard.sol` And `ERC20CollateralPayment.sol`
- ERC-01 : Unsafe Conversion From `uint256` To `int256`
- ERC-02 : Centralization Related Risks In `ERC20CollateralPayment.sol`
- ERC-03 : Unused Local Variables In `ERC20CollateralPayment.sol`
- ERP-01 : Logical Issue In Function `payment()`
- ERP-02 : `awaitingCollection` And `paymentComplete` Are Equal
- ERP-03 : Stack Too Deep
- ERP-04 : Unused State Variable
- ERP-05 : Missing Input Validation
- ICR-01 : Outside Of The Scope
- ICR-02 : Inaccurate Comment
- ICR-03 : `stakingIndexes[_id]` Is Not Updated
- ICR-04 : Inconsistency Between Comment and Code In `Investor.sol`
- ICR-05 : `minPayment` Should Not Be Equal To Zero
- LFR-01 : Unclear Comment
- LFR-02 : Proper Usage for `storage` and `memory`
- LFR-03 : Redundant Code
- LFR-04 : Centralization Related Risks In `LiquidityFarm.sol`
- PFP-01 : Unused Local Variables In `PriceFeed.sol`
- PFP-02 : Misleading Function Name

PFP-03 : Centralization Related Risks In `PriceFeed.sol`
PSR-01 : Natspec Comments Are Inconsistent With Code
PSR-02 : Missing Error Messages
PSR-03 : Leftover Tokens Due To Rounding
PSR-04 : External Call Inside a Loop
PSR-05 : Centralization Related Risks In `PaymentSplitter.sol`
RCV-01 : Inconsistency Between Comment and Code In `RociCreditTokenV2.sol`
RCV-02 : Susceptible to Signature Malleability
RCV-03 : Redundant Statements
RCV-04 : Comparison to A Boolean Constant
RCV-05 : Token id And Associated Bundle Can Be Changed
RCV-06 : Token Counter Will Be Incorrect After Reminting Tokens
RCV-07 : Centralization Related Risks In `RociCreditTokenV2.sol`
RFC-01 : Third Party Dependencies In `Trader.sol`, `UniTrader.sol`
RFC-02 : Wrong Number Of Parameters
RFC-03 : Function Visibility Optimization
RFC-04 : Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call
RFC-05 : Unlocked Compiler Version
RFC-06 : Unused Return Value
RFC-07 : Missing Emit Events
RFC-08 : External Dependencies Risk
RFC-09 : Check Effect Interaction Pattern Violated
RFC-10 : Lack of Natspec Comments
RFP-01 : `beforeTrade()` Are Not Implemented
RFP-02 : Check Effect Interaction Pattern Violated
RMM-01 : Centralization Related Risks In `RevenueManager.sol`
SDB-01 : Useless Pausable Feature
SDB-02 : Third Party Dependencies In `ScoreDB.sol`
SDB-03 : Natspec Comments Are Inconsistent With Code
SDB-04 : Centralization Related Risks In `ScoreDB.sol`
TIC-01 : Declare Variables `constant`
TRF-01 : Potential Risk Of Loss Or Theft Of Tokens
TRF-02 : Centralization Related Risks In `Trader.sol`
UTR-01 : Potential Sandwich Attacks
UTR-02 : Unused Event
UTR-03 : Unusued Function ` uniZap()`

Appendix

Disclaimer

About

Summary

This report has been prepared for RociFi to discover issues and vulnerabilities in the source code of the RociFi project 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;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	RociFi
Platform	Custom
Language	Solidity
Codebase	https://github.com/loan-wolf/
Commit	aaca2f5b4eca1149ef140bfc3017ed295f35366e

Audit Summary

Delivery Date	Feb 10, 2022
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Mitigated	Resolved
Critical	3	2	0	1	0	0	0
Major	10	1	0	9	0	0	0
Medium	2	2	0	0	0	0	0
Minor	11	11	0	0	0	0	0
Informational	24	24	0	0	0	0	0
Discussion	12	12	0	0	0	0	0

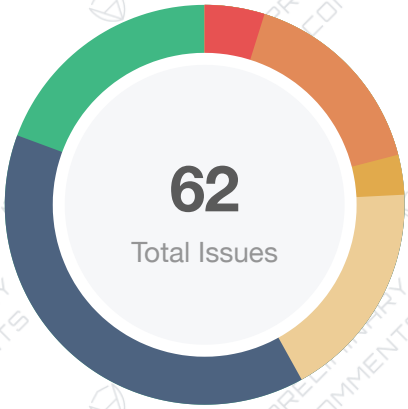
Audit Scope

ID	File	SHA256 Checksum
BBR	projects/RociFi/contracts/Bonds/Bonds.sol	8b288ac6452e07abd3a9da69b15058bffab88a9f8f1ed6db524c5f72f151a383
RCT	projects/RociFi/contracts/NFCS/RociCreditTokenInterface.sol	9f602e8025732b73a4f9d93d8b1569e999cd972e780b0271a45660987016d846
RCV	projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol	c66eb1d39cf866ff6953948abc5332ae765877b89d31ee15c9c8df07b48e28cc
IIO	projects/RociFi/contracts/Oracle/IInvestor.sol	d58363691e4b9a9c41d342c86d5c9eb4bc6059d3068e17c98c860404b8360b34
IRC	projects/RociFi/contracts/Oracle/IRociCreditToken.sol	0508402b4f02372c9c10c100f8fb52556943def048f9d43ec8e3b5c8dfa5ca12
SDB	projects/RociFi/contracts/Oracle/ScoreDB.sol	1f52c894b7114ddb8295ab89002ea5a347499aaefa543f3420440ed064ac9885
SDI	projects/RociFi/contracts/Oracle/ScoreDBInterface.sol	ad5868f4b880c83e53d1e003bbf724302ddae45c05f82163faf077e83a4890bc
PFP	projects/RociFi/contracts/PriceFeed/PriceFeed.sol	39a86f60cee2b4e9ec646ea46669c5bab244db7ae82bc917f139a3d66649df99
PSR	projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol	eea486a107089d71e3f504e7afbbb98a4b0e18017bf930dbfd430c403dfb95de
RMM	projects/RociFi/contracts/RevenueManager/RevenueManager.sol	0e36e7356406b0a820451fbc997edf9015edf025f6cff08866fb8b81ba723a21
SSR	projects/RociFi/contracts/RevenueManager/StakingStub.sol	89873464da69ed9e46ad7394be125230b1067fe599a6c2a52e41325eb9de31ed
CMM	projects/RociFi/contracts/collateralManager/CollateralManager.sol	9e8505b968ae47f5f3b7b6985841a6701031a9ca23a5289f2f0a49e4beb76c93
LFR	projects/RociFi/contracts/farming/LiquidityFarm.sol	c0792bf929843c3cc6390e68ddbd9b3f241f0129f10868c5968e2c4915a6f353
IPS	projects/RociFi/contracts/interfaces/revManager/IPaymentSplitter.sol	fbe1304d541e7480afb00efc5578a1a78ec30297375fbfed6da22c3c5eb79d48
IRM	projects/RociFi/contracts/interfaces/revManager/IRevenueManager.sol	45e7970d0002978eeb6da096f6354a9272d3c1724c8d887612d8c7fb34de7fbf

ID	File	SHA256 Checksum
IUV	projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
IUP	projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Pair.sol	29c75e69ce173ff8b498584700fef76bc81498c1d98120e2877a1439f0c31b5a
IUR	projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
IUF	projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38
IBR	projects/RociFi/contracts/interfaces/IBonds.sol	564c7f7482d4e4c970d2d570aee22072743e197e91823999ca6734cef954d643
ICM	projects/RociFi/contracts/interfaces/ICollateralManager.sol	34b4ab851a0a521952d1621672d1a16e09f2025a613acf2e6f6d75f325675513
IER	projects/RociFi/contracts/interfaces/IERC20PaymentStandard.sol	ba6bd19f154c4ce8c19f90ae63802a25f10e92394de301230b0ffe583ba7cb4f
IPF	projects/RociFi/contracts/interfaces/IPriceFeed.sol	6ae796bfc7f7b1df53c23e22c19f1ed7b88118f0541d50c2acdbb5ffa6e777dd
ITR	projects/RociFi/contracts/interfaces/ITrader.sol	af40b2ad388321f47b254e618dd61fb40ec29502425ae84ae47a3ddf1033d411
SOI	projects/RociFi/contracts/interfaces/ScoreOracleInterface.sol	b21933a0e59e8eeef85050dd2fba000e11066f9985c857c8323f3c31c91e20a0
ERC	projects/RociFi/contracts/investorContract/ERC20CollateralPayment.sol	efa60564c2ad571226392bd0289e9d709d407b437f4200577078e10a738fd0e2
ERP	projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol	d79daba954149d30cc21dff0c7499426473ef90145339637ff38cf853ba9b6ff
ICR	projects/RociFi/contracts/investorContract/Investor.sol	9e447525837ea3ca0f0bc2d5ca4648d4033f25746dbb404f1da7c0567c1ee601
TIC	projects/RociFi/contracts/investorContract/TraderInvestor.sol	b8ea93106eaf58812fb2e353ad6e9c40e8b27f6422d004f7770a29ca74754530
TRF	projects/RociFi/contracts/trader/Trader.sol	4b5c6b5cccd07a963180d97d3492f6a5d13a3f3043e6f957bdc39a9ff40b5648

ID	File	SHA256 Checksum
UTR	projects/RociFi/contracts/trader/UniTrader.sol	95054428f52f754679108a8b91688421fd617ab131cf56b03d1337f729622585
GRF	projects/RociFi/contracts/Globals.sol	d846d540fa481c9c9f9210fc03d5d3a36d27fe26fdabf2c8c7d2715537dc2614

Findings



Critical	3 (4.84%)
Major	10 (16.13%)
Medium	2 (3.23%)
Minor	11 (17.74%)
Informational	24 (38.71%)
Discussion	12 (19.35%)

ID	Title	Category	Severity	Status
BBR-01	Compiler Error	Compiler Error	Discussion	Pending
BBR-02	Missing Interface Inheritance	Control Flow, Volatile Code	Informational	Pending
CMM-01	Struct Tight-Packing	Language Specific, Gas Optimization	Informational	Pending
CMM-02	`_amount` Can Equal Zero	Gas Optimization	Informational	Pending
CMM-03	Centralization Related Risks In `CollateralManager.sol`	**Centralization / Privilege**	**Major**	Acknowledged
CRF-01	Variable Declare as Immutable	Gas Optimization	Informational	Pending
CRF-02	Centralization Related Risks In `ERC20PaymentStandard.sol` And `ERC20CollateralPayment.sol`	**Centralization / Privilege**	**Major**	Acknowledged
ERC-01	Unsafe Conversion From `uint256` To `int256`	Volatile Code	Discussion	Pending
ERC-02	Centralization Related Risks In `ERC20CollateralPayment.sol`	**Centralization / Privilege**	**Major**	Acknowledged
ERC-03	Unused Local Variables In `ERC20CollateralPayment.sol`	Gas Optimization	Informational	Pending
ERP-01	Logical Issue In Function `payment()`	Logical Issue, Inconsistency	Critical	Pending

ID	Title	Category	Severity	Status
ERP-02	`awaitingCollection` And `paymentComplete` Are Equal	Gas Optimization, Logical Issue	Informational	Pending
ERP-03	Stack Too Deep	Compiler Error	Informational	Pending
ERP-04	Unused State Variable	Gas Optimization	Informational	Pending
ERP-05	Missing Input Validation	Volatile Code	Minor	Pending
ICR-01	Outside Of The Scope	Volatile Code	Informational	Pending
ICR-02	Inaccurate Comment	Inconsistency	Discussion	Pending
ICR-03	`stakingIndexes[_id]` Is Not Updated	Logical Issue, Volatile Code	Discussion	Pending
ICR-04	Inconsistency Between Comment and Code In `Investor.sol`	Inconsistency, Logical Issue	Discussion	Pending
ICR-05	`minPayment` Should Not Be Equal To Zero	Logical Issue	Major	Pending
LFR-01	Unclear Comment	Inconsistency	Informational	Pending
LFR-02	Proper Usage for `storage` and `memory`	Gas Optimization	Informational	Pending
LFR-03	Redundant Code	Volatile Code	Informational	Pending
LFR-04	Centralization Related Risks In `LiquidityFarm.sol`	**Centralization / Privilege**	**Major**	Acknowledged
PFP-01	Unused Local Variables In `PriceFeed.sol`	Gas Optimization	Informational	Pending
PFP-02	Misleading Function Name	Logical Issue	Discussion	Pending
PFP-03	Centralization Related Risks In `PriceFeed.sol`	**Centralization / Privilege**	**Major**	Acknowledged
PSR-01	Natspec Comments Are Inconsistent With Code	Inconsistency	Discussion	Pending
PSR-02	Missing Error Messages	Coding Style	Informational	Pending
PSR-03	Leftover Tokens Due To Rounding	Mathematical Operations, Logical Issue	Minor	Pending

ID	Title	Category	Severity	Status
PSR-04	External Call Inside a Loop	Control Flow	Minor	Pending
PSR-05	Centralization Related Risks In `PaymentSplitter.sol`	**Centralization / Privilege**	**Major**	Acknowledged
RCV-01	Inconsistency Between Comment and Code In `RociCreditTokenV2.sol`	Inconsistency	Discussion	Pending
RCV-02	Susceptible to Signature Malleability	Volatile Code	Medium	Pending
RCV-03	Redundant Statements	Volatile Code, Gas Optimization	Informational	Pending
RCV-04	Comparison to A Boolean Constant	Language Specific	Informational	Pending
RCV-05	Token id And Associated Bundle Can Be Changed	Data Flow	Minor	Pending
RCV-06	Token Counter Will Be Incorrect After Reminting Tokens	Logical Issue	Minor	Pending
RCV-07	Centralization Related Risks In `RociCreditTokenV2.sol`	**Centralization / Privilege**	**Major**	Acknowledged
RFC-01	Third Party Dependencies In `Trader.sol`, `UniTrader.sol`	Volatile Code	Minor	Pending
RFC-02	Wrong Number Of Parameters	Language Specific, Volatile Code	Discussion	Pending
RFC-03	Function Visibility Optimization	Gas Optimization	Informational	Pending
RFC-04	Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call	Volatile Code	Minor	Pending
RFC-05	Unlocked Compiler Version	Language Specific	Informational	Pending
RFC-06	Unused Return Value	Volatile Code	Minor	Pending
RFC-07	Missing Emit Events	Coding Style	Informational	Pending
RFC-08	External Dependencies Risk	Volatile Code	Minor	Pending
RFC-09	Check Effect Interaction Pattern Violated	Logical Issue	Informational	Pending

ID	Title	Category	Severity	Status
RFC-10	Lack of Natspec Comments	Coding Style	Informational	Pending
RFP-01	`beforeTrade()` Are Not Implemented	Volatile Code	Discussion	Pending
RFP-02	Check Effect Interaction Pattern Violated	Logical Issue	Minor	Pending
RMM-01	Centralization Related Risks In `RevenueManager.sol`	**Centralization / Privilege**	**Major**	Acknowledged
SDB-01	Useless Pausable Feature	Volatile Code, Logical Issue	Discussion	Pending
SDB-02	Third Party Dependencies In `ScoreDB.sol`	Volatile Code	Minor	Pending
SDB-03	Natspec Comments Are Inconsistent With Code	Inconsistency	Discussion	Pending
SDB-04	Centralization Related Risks In `ScoreDB.sol`	**Centralization / Privilege**	**Major**	Acknowledged
TIC-01	Declare Variables `constant`	Gas Optimization	Informational	Pending
TRF-01	Potential Risk Of Loss Or Theft Of Tokens	Logical Issue	Critical	Pending
TRF-02	Centralization Related Risks In `Trader.sol`	**Centralization / Privilege**	**Critical**	Acknowledged
UTR-01	Potential Sandwich Attacks	Logical Issue, Control Flow	Medium	Pending
UTR-02	Unused Event	Gas Optimization	Informational	Pending
UTR-03	Unusued Function `_uniZap()`	Volatile Code, Gas Optimization	Informational	Pending

BBR-01 | Compiler Error

Category	Severity	Location	Status
Compiler Error	Discussion	projects/RociFi/contracts/Bonds/Bonds.sol: 45~46	Pending

Description

As you can see here: https://github.com/ethereum/solidity/blob/develop/Changelog.md#084-2021-04-21, the

```
*`@notice` tag on non-public state variables and local variable declarations*
```

is allowed only since solidity 0.8.4.

This means the contract cannot compile for versions: `0.8.0`, `0.8.1`, `0.8.2`, `0.8.3`; because of the two linked lines.

The error message returned is:

```
Error: Documentation tag @notice not valid for non-public state variables.
```

Recommendation

This problem can be solved by, for example, using one of these options:

- declaring **explicitly** the mapping `staking` as `public`, this will make the program compile for any `0.8.x` version;
- locking the version of the contract above 0.8.4;
- removing the NatSpec comments on this mapping.

BBR-02 | Missing Interface Inheritance

Category	Severity	Location	Status
Control Flow, Volatile Code	Informational	projects/RociFi/contracts/Bonds/Bonds.sol: 4	Pending

Description

The `Bonds.sol` contract imports `IBonds.sol` which imports `IERC1155Receiver.sol` but `Bonds.sol` does not inherit from `IERC1155Receiver.sol`.

Recommendation

We advise making `Bonds.sol` inheriting from `IERC1155Receiver.sol`.

CMM-01 | Struct Tight-Packing

Category	Severity	Location	Status
Language Specific, Gas Optimization	Informational	projects/RociFi/contracts/collateralManager/CollateralManager.sol: 16~18	Pending

Description

The highlighted variables and structure could be tight-packed.

Recommendation

We advise using the **tight variable packing** pattern to rewrite the highlighted code in order to reduce the gas cost.

CMM-02 | `_amount` Can Equal Zero

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/collateralManager/CollateralManager.sol: 82	Pending

Description

In `CollateralManager.sol`, in function `deposit()` allow a user to deposit 0 tokens. When

```
collateralLookup[msg.sender][_loanId].amount == 0
```

this allows to rewrite multiple times the `block.timestamp` and `_erc20` parameter of a `collateral`.

Recommendation

We advise adding a check to prevent calling `deposit()` with `_amount = 0`.

CMM-03 | Centralization Related Risks In `CollateralManager.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/collateralManager/CollateralManager.sol: 33~37, 43~50	Acknowledged

Description

In the contract `CollateralManager.sol`, the role `owner` has authority over the following functions:

- `addAcceptedCollateral()`;
- `removeAcceptedCollateral()`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and for example, change accepted collateral and renounce ownership so the contract cannot use new collateral or remove old collateral, this could sabotage the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

CRF-01 | Variable Declare As Immutable

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol: 20~22 projects/RociFi/contracts/investorContract/TraderInvestor.sol: 20	Pending

Description

The linked variables assigned in the constructor can be declared with `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since will not be stored in storage. Still, values will directly insert the values into the runtime code.

Recommendation

We recommend using an immutable state variable for these variables.

CRF-02 | Centralization Related Risks In `ERC20PaymentStandard.sol` And `ERC20CollateralPayment.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol: 117~129 projects/RociFi/contracts/investorContract/ERC20CollateralPayment.sol: 69~78	Acknowledged

Description

In the contract `ERC20PaymentStandard.sol`, the address `bondContract` has authority over the following functions:

- `issueBonds()`;
- `addInterest()`;

In the contract `ERC20CollateralPayment.sol`, the address `bondContract` has authority over the `issueBonds()` function.

Any compromise to the `bondContract` address may allow a hacker to take advantage of this authority and:

- issue new Bonds which could block some potential id;
- add as much interest as he/she wants to any incomplete loan. This could sabotage the contract even if the hack is temporary.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

ERC-01 | Unsafe Conversion From `uint256` To `int256`

Category	Severity	Location	Status
Volatile Code	Discussion	projects/RociFi/contracts/investorContract/ERC20CollateralPayment.sol: 99, 140, 180~182, 186~188	Pending

Description

In theory, directly converting a `uint256` into an `int256` could result in a wrong result because of integer overflow/underflow.

Recommendation

Even if the logic of this contract makes it very unlikely to happen, we advise adding a check to avoid any wrong result.
Using the function `toUint256()` from the `SafeCast.sol` library from OpenZeppelin can also solve the problem.

ERC-02 | Centralization Related Risks In `ERC20CollateralPayment.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/investorContract/ERC20CollateralPayment.sol: 51~53, 59~62, 109~113	Acknowledged

Description

In the contract `ERC20CollateralPayment.sol`, the role `owner` has authority over the following functions:

- `changeManager()`;
- `changePriceFeed()`;
- `liquidateByAdmin()`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and for example:

- transfer ownership to an address he/she controls;
- change the `manager` address and sabotage the contract;
- change the `priceFeed` address to manipulate the prices.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations; AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

ERC-03 | Unused Local Variables In `ERC20CollateralPayment.sol`

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/investorContract/ERC20CollateralPayment.sol: 91~92, 94, 200	Pending

Description

The linked variables are defined inside a function but are never used.

Recommendation

We advise using all the local variables defined inside a function or to remove them.

ERP-01 | Logical Issue In Function `payment()`

Category	Severity	Location	Status
Logical Issue, Inconsistency	Critical	projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol: 241, 251	Pending

Description

In the contract `ERC20PaymentStandard.sol` the function `payment()` requires that the amount paid is greater than the `minPayment` of the loan, except if:

```
ln.totalPaymentsValue - ln.paymentComplete < ln.minPayment
```

The comments say that you can pay less than the minimum value if it is the last payment. However, nothing in the **require** statement enforces that.
Indeed as long as the aforementioned condition is met, it is possible to call `payment()` with any value for `_erc20Ammount` even 0.

Hence we have the following potential attack scenario:

- Bob pay the loan until `ln.totalPaymentsValue - ln.paymentComplete < ln.minPayment`,
- every time the `paymentDueDate` is about to be reached, Bob call `payment()` again with the smallest value for `_erc20Ammount` (probably 0),
- `paymentDueDate` is then updated to `block.timestamp + ln.paymentPeriod`,
- as long as he keeps doing that :
  - `_isLate()` function (from `ERC20PaymentStandard`) will always return `false`,
  - `isDelinquent()` will always return `false`,
  - `getLoanInfo()` will always return `(ln.interestRate, ln.accrualPeriod)` so no penalty is counted, this implies in `Bonds.sol`:
    - `unstake()`, `getAccruances()`, `getInterest()` which will affect the number of tokens minted.

Since doing that will very likely increase the `totalPaymentsValue` of a loan, this scenario might not be beneficial for the attacker. However, by contagion effect, this can considerably affect the good performance of the whole system.

Recommendation

We advise adding some restrictions to avoid multiple payments below the `minPayment`.

ERP-02 | `awaitingCollection` And `paymentComplete` Are Equal

Category	Severity	Location	Status
Gas Optimization, Logical Issue	Informational	projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol: 199~200, 250, 252	Pending

Description

Each new `loan` structure is initialized with :

```
199 awaitingCollection: 0,
200 paymentComplete: 0
```

The only time these variables have their values changed is in the function `payment`:

```
250 loanLookup[_id].awaitingCollection += _erc20Ammount;
251 .....
252 loanLookup[_id].paymentComplete += _erc20Ammount; //Increase paymentComplete
```

This implies that we always have :

`loanLookup[_id].awaitingCollection = loanLookup[_id].paymentComplete`.

Recommendation

We advise removing one of the two variables and replacing all its occurrences with the other one. This would also solve the **Stack Too Deep** compiler error.

ERP-03 | Stack Too Deep

Category	Severity	Location	Status
Compiler Error	Informational	projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol : 187~201	Pending

Description

With optimization disabled the compiler returns this error :

```
Stack too deep when compiling inline assembly: Variable headStart is 1 slot(s) too deep inside the stack.
```

Recommendation

We advise optimizing this function to avoid deployment issues or enable optimization options when deploying the contract.
Moreover, since `awaitingCollection` and `paymentComplete` are two equal variables changing all the occurrences of `awaitingCollection` by `paymentComplete` and removing `awaitingCollection` would solve the problem.

ERP-04 | Unused State Variable

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol: 17	Pending

Description

The linked state variables are never used in the codebase.

Recommendation

We advise removing the unused variable.

ERP-05 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol: 49~51	Pending

Description

The given input is missing the check for the non-zero address.

Recommendation

We advise adding the check for the passed-in values to prevent unexpected errors.

ICR-01 | Outside Of The Scope

Category	Severity	Location	Status
Volatile Code	Informational	projects/RociFi/contracts/investorContract/Investor.sol: 4	Pending

Description

The testing contracts are outside of the scope of the audit, these entities are treated as black boxes and their functional correctness is assumed.

Recommendation

We encourage the team to test and check these contracts to find and mitigate the possible side effects.

ICR-02 | Inaccurate Comment

Category	Severity	Location	Status
Inconsistency	Discussion	projects/RociFi/contracts/investorContract/Investor.sol: 34	Pending

Description

The comments of the constructor in `Investor.sol` say that:

```
34  * @dev creates a uniswap pair of the two tokens upon construction
```

However no uniswap pair will be created.

Recommendation

We recommend the team review the design and update either comments or code implementation to ensure consistent logic between code and comment.

ICR-03 | `stakingIndexes[_id]` Is Not Updated

Category	Severity	Location	Status
Logical Issue, Volatile Code	Discussion	projects/RociFi/contracts/investorContract/Investor.sol: 149	Pending

Description

In the contract `Investor.sol`, calling the function `payment()` will unstake the bonds precedently staked and restaked the remaining bonds. The `stake()` function will return an `uint256` which should be used as the new staking index for the specific `_id`.
However here this value is stored in the local variable `stakeIndex` and is never used.

Recommendation

We advise adding:

```
stakingIndexes[_id] = stakeIndex;
```

after the restacking happened.

ICR-04 | Inconsistency Between Comment And Code In `Investor.sol`

Category	Severity	Location	Status
Inconsistency, Logical Issue	Discussion	projects/RociFi/contracts/investorContract/Investor.sol: 165~166	Pending

Description

In the `Investor.sol`, the `borrowFulfill()` function creates and issues a new loan by calling external functions. These two lines contradict each other:

```
165         // this function calls the issue function with requires non-delinquency
166         bonds.newLoan(address(this), _id, _hash, _signature);
```

Indeed: `newLoan()` from `Bonds.sol` calls the `issueBonds()` function from an `ERC20PaymentStandard` contract which does not perform any check about delinquency.

The function checking for non-deliquency is the `issueBonds()` function from the contract `ERC20CollateralPayment` which is never called in the `borrowFulfill()` function.

Recommendation

We recommend the team review the design and update either comments or code implementation to ensure consistent logic between code and comment.

ICR-05 | `minPayment` Should Not Be Equal To Zero

Category	Severity	Location	Status
Logical Issue	Major	projects/RociFi/contracts/investorContract/Investor.sol: 106	Pending

Description

Allowing 0 as minimal payment means that a user can always postpone the payment `duedate`, this could strongly interfere with the overall logic of the system.

Recommendation

We advise setting a strictly positive integer as the value of `minPayment`.

LFR-01 | Unclear Comment

Category	Severity	Location	Status
Inconsistency	Informational	projects/RociFi/contracts/farming/LiquidityFarm.sol: 53~61	Pending

Description

According to the comment L53 the function `getPoolInfo()` returns the pool's farm info if it exists. This may suggest that the function will not return anything if the pool does not exist, however, in this case, the function will return zero values.

Recommendation

We recommend the team review the logic to check if it behaves as intended, and update either comment or code implementation to improve the comment accuracy.

LFR-02 | Proper Usage For `storage` And `memory`

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/farming/LiquidityFarm.sol: 74, 89~90	Pending

Description

The linked functions are view functions. Defining the variables as `storage` is not necessary and costs more gas.

Recommendation

We advise changing `storage` to `memory`.

LFR-03 | Redundant Code

Category	Severity	Location	Status
Volatile Code	Informational	projects/RociFi/contracts/farming/LiquidityFarm.sol: 191, 244	Pending

Description

In the `LiquidityFarm` contract:

- the function `_updateDeposit()` is private and is called by the functions `depositZap()` and `deposit()`;
- the function `_updateWithdraw()` is private and is called by `withdraw()` and `withdrawZapAndUnstake()`;

All these functions used the `poolExists` modifier, hence there is redundancy in both groups.

Recommendation

We advise removing the modifier from:

- `depositZap()` and `deposit()`, **or** removing it from `_updateDeposit()`;
- `withdraw()` and `withdrawZapAndUnstake()`, **or** removing it from `_updateWithdraw()`;

to avoid redundant checks

LFR-04 | Centralization Related Risks In `LiquidityFarm.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/farming/LiquidityFarm.sol: 108~129, 133~154	Acknowledged

Description

In the contract `LiquidityFarm.sol`, the role `owner` has authority over the following functions:

- `createFarm()`;
- `updateFarm()`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and for example, update all the pool with bad parameters and renounce ownership so the contract becomes useless.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

PFP-01 | Unused Local Variables In `PriceFeed.sol`

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/PriceFeed/PriceFeed.sol: 32~39	Pending

Description

In the contract `PriceFeed.sol` the function `getLatestPriceUSD()` extract every variables returned by the function `latestRoundData()`, however only the `price` variable is used:

```
32      (
33          uint80 roundID,
34          int price,
35          uint startedAt,
36          uint timeStamp,
37          uint80 answeredInRound
38      ) = priceFeed.latestRoundData();
39      return price;
40  }
```

Recommendation

We advise using all the variables stored in the function or to rewrite the function as follow:

```
    function getLatestPriceUSD(address _token) public view returns (int) {
        require(priceFeedAddresses[_token] != address(0), "This token is not supported");
        AggregatorV3Interface priceFeed =
AggregatorV3Interface(priceFeedAddresses[_token]);
        (, int price,,,) = priceFeed.latestRoundData();
        return price;
    }
```

PFP-02 | Misleading Function Name

Category	Severity	Location	Status
Logical Issue	Discussion	projects/RociFi/contracts/PriceFeed/PriceFeed.sol: 22	Pending

Description

In the contract `PriceFeed.sol` the function `addPriceFeed()` is not reflecting accuratly what the function can do.

Indeed it is possible to add a new pricefeed to a new token, however, it is also possible to change the token address or the pricefeed address of an already existing couple.

Is that intended?

Recommendation

We advise either rewriting the function to make it consistent with its name and comment or changing the name and the comment so they are consistent with the function logic.

PFP-03 | Centralization Related Risks In `PriceFeed.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/PriceFeed/PriceFeed.sol: 22~24	Acknowledged

Description

In the contract `PriceFeed.sol` the role `owner` has authority over the following functions:

- `renounceOwnership()` (from `Ownable`);
- `transferOwnership()` (from `Ownable`);
- `addPriceFeed()`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and for example: renounce ownership after resetting every pricefeeds to the zero address so the `getLatestPriceUSD()` always revert. This would completely sabotage the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

PSR-01 | Natspec Comments Are Inconsistent With Code

Category	Severity	Location	Status
Inconsistency	Discussion	projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 10	Pending

Description

The comment mentions that the contract `PaymentSplitter.sol` should be upgradeable yet there are no contracts imported that adds an upgradeable feature to this contract.

Recommendation

We advise the team to reread the comment and necessary contracts if needed.

PSR-02 | Missing Error Messages

Category	Severity	Location	Status
Coding Style	Informational	projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 38~46, 52~57	Pending

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise including an error message to the linked statement. The following line is suffice:

```
require(_payees.length == _shares.length, "Arguments lengths do not match");
```

PSR-03 | Leftover Tokens Due To Rounding

Category	Severity	Location	Status
Mathematical Operations, Logical Issue	Minor	projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 73	Pending

Description

The arithmetical operations will round down and therefore transfer slightly fewer tokens than the original amount. These tokens would be locked in the balance of the contract. The issue is that over time more tokens will be locked within the contract and there is no way to transfer the tokens out of the contract.

Recommendation

We recommend rewriting the logic of the contract to avoid this problem, for example: implementing a function that would round down the `amount` to the nearest integer before the transfer takes place.

PSR-04 | External Call Inside A Loop

Category	Severity	Location	Status
Control Flow	Minor	projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 73	Pending

Description

In the contract `PaymentSplitter.sol`, the function `payment()` has an external call inside a loop. Calls inside a loop might lead to a denial-of-service attack, especially when an arbitrary contract can be passed as a parameter.

Recommendation

We advise adding more checks to mitigate any risk related to these external calls.

PSR-05 | Centralization Related Risks In `PaymentSplitter.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 38~46	Acknowledged

Description

In the contract `PaymentSplitter.sol`, the role `owner` has authority over the following functions:

- `addShares()`;
- `removeShares()`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and for example, change the payees and their shares and renounce ownership so the contract cannot change these values, this could sabotage the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

RCV-01 | Inconsistency Between Comment And Code In `RociCreditTokenV2.sol`

Category	Severity	Location	Status
Inconsistency	Discussion	projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol: 57	Pending

Description

This comment:

```
57  * @notice Pauses the whole contract; used as emergency response in case a bug is
detected. [OWNER_ONLY]
```

states that all the contract is paused when `pause()` is called, however, there is only one function, `_beforeTokenTransfer()`, using the `whenNotPaused` modifier meaning that it is the only function paused when `pause()` is called.

Recommendation

We recommend the team review the design and update either comments or code implementation to ensure consistent logic between code and comment.

RCV-02 | Susceptible To Signature Malleability

Category	Severity	Location	Status
Volatile Code	Medium	projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol: 162	Pending

Description

The signature malleability is possible within the Elliptic Curve cryptographic system. An Elliptic Curve is symmetric on the X-axis, meaning two points can exist with the same `X` value. In the `r`, `s` and `v` representation this permits us to carefully adjust `s` to produce a second valid signature for the same `r`, thus breaking the assumption that a signature cannot be replayed in what is known as a replay-attack.

Recommendation

We advise to utilize a `recover()` function similar to that of the `ECDSA.sol` implementation of OpenZeppelin.

RCV-03 | Redundant Statements

Category	Severity	Location	Status
Volatile Code, Gas Optimization	Informational	projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol : 99	Pending

Description

In the contract `RociCreditTokenV2.sol`, the function `mintToken()` contains these lines:

```
97  storeBundle(tokenId, bundle);
98  // Set bundle nonce to true
99  _bundleNonce[tokenId] = true;
```

However, the function `storeBundle()` already contains this line:

```
206  _bundleNonce[tokenId] = true;
```

Hence, setting the bundle nonce to true in `mintToken()` is redundant.

Recommendation

We advise removing the redundant code.

RCV-04 | Comparison To A Boolean Constant

Category	Severity	Location	Status
Language Specific	Informational	projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol: 239	Pending

Description

Boolean constants can be used directly and do not need to be compared to true or false.

Recommendation

We advise removing the comparison to the boolean constant:

```
require(_mintedNonce[tokenOwner], "This address does not own a token.");
```

RCV-05 | Token Id And Associated Bundle Can Be Changed

Category	Severity	Location	Status
Data Flow	Minor	projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol: 73	Pending

Description

Every token has an owner and a bundle of users mapped to it. The owner has permission to change the associated bundle by calling the `public` `remintToken()` function and passing different arguments for the parameters.

Recommendation

The function `remintToken()` allows changes to sensitive information. Consider adding logic that would only implement `remintToken()` if certain conditions are met.

RCV-06 | Token Counter Will Be Incorrect After Reminting Tokens

Category	Severity	Location	Status
Logical Issue	Minor	projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol: 93, 299	Pending

Description

The function `mintToken()` mints a token to the user and within the logic the token id is created. The issue is if a user a remints a token, the id will increase up by 1 while the past token was burnt.

Recommendation

We recommend decreasing the token count when the function `burnToken()` is called so that the token id is consistent with the number of existing tokens.

RCV-07 | Centralization Related Risks In `RociCreditTokenV2.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol: 59~61, 66~68, 331~335	Acknowledged

Description

In the contract `RociCreditTokenV2.sol`, the role `owner` has authority over the following functions:

- `pause()`;
- `unpause()`;
- `_authorizeUpgrade()`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and pause the function `_beforeTokenTransfer()` and renounce ownership so it is impossible to unpause it.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

RFC-01 | Third Party Dependencies In `Trader.sol` , `UniTrader.sol`

Category	Severity	Location	Status
Volatile Code	Minor	projects/RociFi/contracts/trader/UniTrader.sol: 19~20 projects/RociFi/contracts/PriceFeed/PriceFeed.sol: 31, 4	Pending

Description

The contract is serving as the underlying entity to interact with third-party **Uniswap** protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of `RociFi` requires interaction with **Uniswap** contracts. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

RFC-02 | Wrong Number Of Parameters

Category	Severity	Location	Status
Language Specific, Volatile Code	Discussion	projects/RociFi/contracts/investorContract/Investor.sol: 119 projects/RociFi/contracts/interfaces/ScoreOracleInterface.sol: 6 projects/RociFi/contracts/Oracle/ScoreDB.sol: 149	Pending

Description

The only implemented `requestUpdatedScoreBorrow()` function in the scope of the audit is in `ScoreDB.sol`, and this function needs six paramters:

```
function requestUpdatedScoreBorrow(
    uint256 tokenId,
    uint256 loanId,
    bytes32 hash,
    bytes memory signature,
    address callBackContract,
    bytes4 functionSelector
) public override {
```

but only four parameters are given here:

```
ScoreOracleInterface(creditOracle).requestUpdatedScoreBorrow(
    _NCFSID,
    id,
    _hash,
    _signature
);
```

Recommendation

Our recommendation depends on the client's intentions. If this function call is directed to the `scoreDB` contract then there is a major issue since the function cannot be called because of the incorrect amount of arguments being passed.

RFC-03 | Function Visibility Optimization

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/Bonds/Bonds.sol: 206~214, 217~225 projects/RociFi/contracts/investorContract/ERC20CollateralPayment.sol: 39~45, 69~78 projects/RociFi/contracts/farming/LiquidityFarm.sol: 108~129, 133~154, 159~165, 168~187, 215~221, 224~240 projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 64~76	Pending

Description

The linked functions are declared as `public` but are not invoked in their contracts. The functions that are never called internally within the contract should have external visibility.

Recommendation

We advise that the functions' visibility specifiers are set to `external`, optimizing the gas cost of the function.

RFC-04 | Unchecked Value Of ERC-20 `transfer()` / `transferFrom()` Call

Category	Severity	Location	Status
Volatile Code	Minor	projects/RociFi/contracts/trader/Trader.sol: 49, 54, 65, 76, 98~99 projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol: 224, 245 projects/RociFi/contracts/investorContract/Investor.sol: 187~189 projects/RociFi/contracts/collateralManager/CollateralManager.sol: 119, 88 projects/RociFi/contracts/RevenueManager/RevenueManager.sol: 41 projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 66, 73	Pending

Description

The linked `transfer()`/`transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of proper ERC-20 implementation.

Recommendation

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

RFC-05 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	projects/RociFi/contracts/trader/Trader.sol: 2 projects/RociFi/contracts/trader/UniTrader.sol: 2 projects/RociFi/contracts/investorContract/TraderInvestor.sol: 2 projects/RociFi/contracts/investorContract/Investor.sol: 2 projects/RociFi/contracts/investorContract/ERC20PaymentStandard.sol: 2 projects/RociFi/contracts/investorContract/ERC20CollateralPayment.sol: 2 projects/RociFi/contracts/interfaces/ScoreOracleInterface.sol: 3 projects/RociFi/contracts/interfaces/ITrader.sol: 2 projects/RociFi/contracts/interfaces/IPriceFeed.sol: 2 projects/RociFi/contracts/interfaces/IERC20PaymentStandard.sol: 2 projects/RociFi/contracts/interfaces/ICollateralManager.sol: 2 projects/RociFi/contracts/interfaces/IBonds.sol: 2 projects/RociFi/contracts/interfaces/revManager/IRevenueManager.sol: 2 projects/RociFi/contracts/interfaces/revManager/IPaymentSplitter.sol: 2 projects/RociFi/contracts/farming/LiquidityFarm.sol: 2 projects/RociFi/contracts/collateralManager/CollateralManager.sol: 2 projects/RociFi/contracts/RevenueManager/StakingStub.sol: 2 projects/RociFi/contracts/RevenueManager/RevenueManager.sol: 2 projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 2 projects/RociFi/contracts/PriceFeed/PriceFeed.sol: 2 projects/RociFi/contracts/Oracle/ScoreDBInterface.sol: 2 projects/RociFi/contracts/Oracle/ScoreDB.sol: 2 projects/RociFi/contracts/Oracle/IRociCreditToken.sol: 2 projects/RociFi/contracts/Oracle/IInvestor.sol: 2 projects/RociFi/contracts/NFCS/RociCreditTokenV2.sol: 2 projects/RociFi/contracts/NFCS/RociCreditTokenInterface.sol: 2 projects/RociFi/contracts/Bonds/Bonds.sol: 2 projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Router02.sol: 1 projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Router01.sol: 1 projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Pair.sol: 1 projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Factory.sol: 1	Pending

Description

The contract has an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. Because the `Bonds.sol` contract will not compile for a version anterior to solidity `0.8.4`, the compiler version could be locked at `0.8.4` using the following line:

```
pragma solidity 0.8.4;
```

RFC-06 | Unused Return Value

Category	Severity	Location	Status
Volatile Code	Minor	projects/RociFi/contracts/trader/UniTrader.sol: 73, 96~97, 137~138 projects/RociFi/contracts/investorContract/Investor.sol: 119~124, 141 projects/RociFi/contracts/investorContract/TraderInvestor.sol: 42	Pending

Description

The linked functions invocations do not stored the return value from an external call in a local or state variable.

Recommendation

We advise ensuring that all the return values of the function calls are used at least for checking that everything went well.

RFC-07 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	projects/RociFi/contracts/investorContract/ERC20CollateralPayment.sol: 51~53, 59~61, 109~113 projects/RociFi/contracts/investorContract/Investor.sol: 65~71 projects/RociFi/contracts/farming/LiquidityFarm.sol: 108~129, 133~154 projects/RociFi/contracts/collateralManager/CollateralManager.sol: 33~37, 43~50 projects/RociFi/contracts/RevenueManager/RevenueManager.sol: 19~23, 29~33 projects/RociFi/contracts/Oracle/ScoreDB.sol: 190, 212, 216 projects/RociFi/contracts/RevenueManager/PaymentSplitter.sol: 39, 52~57	Pending

Description

Functions that affect the status of sensitive variables should be able to emit events as notifications.

Recommendation

Consider adding events for sensitive actions, and emitting them in the function.

RFC-08 | External Dependencies Risk

Category	Severity	Location	Status
Volatile Code	Minor	projects/RociFi/contracts/RevenueManager/RevenueManager.sol: 39 projects/RociFi/contracts/trader/Trader.sol: 49, 53, 65, 76, 98~99	Pending

Description

The linked functions are `external` functions making external calls to arbitrary contracts that are passed as parameters. Functions defined like such introduce great risk to the contract since a hacker can construct a volatile contract to pass to introduce an attack.

Recommendation

We strongly recommend limiting the number of functions that could make an external call to any contract.

We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

RFC-09 | Check Effect Interaction Pattern Violated

Category	Severity	Location	Status
Logical Issue	Informational	projects/RociFi/contracts/investorContract/Investor.sol: 41~59, 155~179 projects/RociFi/contracts/investorContract/TraderInvestor.sol: 32~47 projects/RociFi/contracts/collateralManager/CollateralManager.sol: 78~103 projects/RociFi/contracts/Oracle/ScoreDB.sol: 73~97, 149~178 projects/RociFi/contracts/Bonds/Bonds.sol: 98~126	Pending

Description

The order of external call/transfer storage manipulation and event emissions should follow the **check-effect-interaction pattern**.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

RFC-10 | Lack Of Natspec Comments

Category	Severity	Location	Status
Coding Style	Informational	projects/RociFi/contracts/interfaces/ScoreOracleInterface.sol: 1 projects/RociFi/contracts/interfaces/ITrader.sol: 1 projects/RociFi/contracts/interfaces/IPriceFeed.sol: 1 projects/RociFi/contracts/interfaces/ICollateralManager.sol: 1 projects/RociFi/contracts/interfaces/IBonds.sol: 1 projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Router02.sol: 1 projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Router01.sol: 1 projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Pair.sol: 1 projects/RociFi/contracts/interfaces/uniswap/IUniswapV2Factory.sol: 1 projects/RociFi/contracts/interfaces/revManager/IRevenueManager.sol: 1 projects/RociFi/contracts/interfaces/revManager/IPaymentSplitter.sol: 1 projects/RociFi/contracts/Oracle/IInvestor.sol: 1 projects/RociFi/contracts/Oracle/IRociCreditToken.sol: 1	Pending

Description

Contract code is missing NatSpec comments, which helps understand the code and all the function's parameters. It also increases code readability which is always welcomed.

Recommendation

We advise following these style guides for adding NatSpec comments.

RFP-01 | `beforeTrade()` Are Not Implemented

Category	Severity	Location	Status
Volatile Code	Discussion	projects/RociFi/contracts/trader/UniTrader.sol: 133, 146 projects/RociFi/contracts/trader/Trader.sol: 133	Pending

Description

In the contract `UniTrader.sol`, the function `_beforeTrade()` is called by `_tradeUniswap()` but does not have any implementation. Since `_tradeUniswap()` is a crucial function for the logic of `UniTrader.sol` and `Trader.sol`, it is not possible to be sure of the safety of these contracts.

Also in the contract `Trader.sol`, a function `_beforeTrade()` is supposed to override the one from `UniTrader.sol` however it does not have any implementation either, and it is not called within the contract.

Recommendation

We advise adding an implementation, consistent with the logic of the contracts, to these functions and using the function in `Trader.sol` or removing them.

RFP-02 | Check Effect Interaction Pattern Violated

Category	Severity	Location	Status
Logical Issue	Minor	projects/RociFi/contracts/trader/Trader.sol: 48~55, 64~69, 71~82, 97~103 projects/RociFi/contracts/trader/UniTrader.sol: 42~48, 53~62, 67~90, 95~110, 132~141	Pending

Description

The order of external call/transfer storage manipulation and event emissions should follow the **check-effect-interaction pattern**.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

RMM-01 | Centralization Related Risks In `RevenueManager.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/RevenueManager/RevenueManager.sol: 19~23, 29~33	Acknowledged

Description

In the contract `RevenueManager.sol`, the role `owner` has authority over the following functions:

- `addInvestors()`;
- `removeInvestors()`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and for example, change the accepted investors and renounce ownership so the contract cannot have new investors or remove old ones, this could sabotage the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

SDB-01 | Useless Pausable Feature

Category	Severity	Location	Status
Volatile Code, Logical Issue	Discussion	projects/RociFi/contracts/Oracle/ScoreDB.sol: 229~238	Pending

Description

In the contract `ScoreDB.sol`, the functions:

- `pause()` make impossible to call a function which use the `whenNotPaused` modifier;
- `unpause()` make impossible to call a function which use the `whenPaused` modifier;

These functions could be really useful to block some functions in the contract in case of an emergency. However, no functions in this contract use the aforementioned modifiers, hence calling one of these two functions would not have any effect.

Recommendation

We advise adding the modifiers from the `Pausable.sol` contract on the functions that should be able to be paused.

SDB-02 | Third Party Dependencies In `ScoreDB.sol`

Category	Severity	Location	Status
Volatile Code	Minor	projects/RociFi/contracts/Oracle/ScoreDB.sol: 83~93, 104	Pending

Description

The contract is serving as the underlying entity to interact with third-party **ChainlinkClient** protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the logic of `requestScore()` requires interaction with an oracle on the **ChainLink** decentralization network. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

SDB-03 | Natspec Comments Are Inconsistent With Code

Category	Severity	Location	Status
Inconsistency	Discussion	projects/RociFi/contracts/Oracle/ScoreDB.sol: 71	Pending

Description

An `@notice` comment describes the functionality of a function. The comment:

```
71  * @notice triggers the score update request
```

hints that a score update should take place yet the function `requestScore()` returns a `requestId` of type `bytes32` instead.

Recommendation

We advise the team to look over the function `requestScore()` and check that the underlying logic is correct.

SDB-04 | Centralization Related Risks In `ScoreDB.sol`

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/RociFi/contracts/Oracle/ScoreDB.sol: 190~192, 212~214 , 216~224, 229~231, 236~238	Acknowledged

Description

In the contract `ScoreDB.sol`, the role `owner` has authority over the following functions:

- `setLtv()`;
- `setBaseUri()`;
- `setOracle()`;
- `pause()`;
- `unpause()`;

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and change important parameters that could result in great damages for the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations; AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;

AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

TIC-01 | Declare Variables `constant`

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/investorContract/TraderInvestor.sol: 18~19	Pending

Description

In `TraderInvestor.sol` the variables `tradeAmount` and `margin` could be declared as `constant` since these state variables are never to be changed.

Recommendation

We advise declaring them as `constant`.

TRF-01 | Potential Risk Of Loss Or Theft Of Tokens

Category	Severity	Location	Status
Logical Issue	Critical	projects/RociFi/contracts/trader/Trader.sol: 48~55, 64~68, 71~82, 97~102	Pending

Description

In `Trader.sol` the `trade()` function call `_tradeUniswap()` only if `currentAMM = AMM.uniswap`.

However if `currentAMM != AMM.uniswap` the function does not revert, this could have two consequences:

- the `_in` tokens sent by the `msg.sender` is not traded and stays in the contract, making the sender lose his/her tokens.
- the transfer of the `_out` tokens will still be tried, meaning that if there are `_out` tokens in the contract not belonging to the sender he could steal them.

The fact that `_in` and `_out` addresses are parameters entered by the caller, made the risk of exploit even higher.

Recommendation

We advise rewriting the logic of this contract to avoid the risks described above. This could be achieved, for example, by using a **require** statement so the function reverts if `currentAMM != AMM.Uniswap`.

TRF-02 | Centralization Related Risks In `Trader.sol`

Category	Severity	Location	Status
Centralization / Privilege	Critical	projects/RociFi/contracts/trader/Trader.sol: 38	Acknowledged

Description

In the contract `Trader.sol`, the role `owner` has authority over the following functions:

- `transferOwnership()` (from `Ownable.sol`);
- `renounceOwnership()` (from `Ownable.sol`);
- `setAMM()`.

These privileges could allow a hacker to exploit other problems to divert tokens or just sabotage the contract.
Indeed any compromise to the `owner` account may allow a hacker to take advantage of this authority and for example:

- transfer ownership to an address he/she controls;
- change the `currentAMM` variable so users will send tokens that will be locked in the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

`[RociFi]`: We're going to introduce multisig and timelock for all ownable contracts.

UTR-01 | Potential Sandwich Attacks

Category	Severity	Location	Status
Logical Issue, Control Flow	Medium	projects/RociFi/contracts/trader/UniTrader.sol: 138, 104~105, 79~80	Pending

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction is attacked) a transaction to sell the asset.

The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- `uniswap.addLiquidity()` in `_uniZapTokensToLP()`;
- `uniswap.removeLiquidity()` in `_uniZapLPtoTokens()`;
- `uniswap.swapExactTokensForTokens()` in `_tradeUniswap()`.

Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

UTR-02 | Unused Event

Category	Severity	Location	Status
Gas Optimization	Informational	projects/RociFi/contracts/trader/UniTrader.sol: 28	Pending

Description

The event `UniswapLPWithdrawal` is declared but never emitted.

Recommendation

We recommend removing this event or emitting it in the right places.

UTR-03 | Unusued Function `_uniZap()`

Category	Severity	Location	Status
Volatile Code, Gas Optimization	Informational	projects/RociFi/contracts/trader/UniTrader.sol: 33~37	Pending

Description

In the contract `UniTrader.sol`, the function `_uniZap(address _lpToken, uint _amount, address _caller)` is internal and is never used within the scope of the audit.

Recommendation

We advise using this function or removing it.

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.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

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

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

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 you ("Customer" or 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, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

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.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

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.