CERTIK

Security Assessment

Celer Layer2 Finance v2

Jun 25th, 2021

Table of Contents

Summary

This report has been prepared for Celer Layer2 Finance v2 smart contracts, to discover issues and vulnerabilities in the source code of their smart contracts 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	Celer Layer2 Finance v2
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/celer-network/layer2-finance-v2-contracts
Commit	4bd09ee4602ff8dde44a2108916ef2aaef86b4d0

Audit Summary

Delivery Date	Jun 25, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

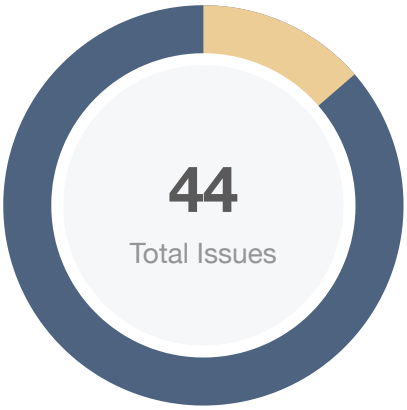
Vulnerability Summary

Vulnerability Level	Total Count	Pending	Partially Resolved	Resolved	Acknowledged	Declined
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	6	0	0	1	0	5
Informational	38	0	0	28	6	4
Discussion	0	0	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum
REG	Registry.sol	63e7695aa10897d9ff5b11016746e4f6312988b3014843289e45d0a70f0c175a
RCK	RollupChain.sol	cc9518b6032b7fcd305bf6e6b712e48e2a2ab76ae30c40092dc930ef4ddb9c5e
TAK	TransitionApplier1.sol	3a83e532ad675a16a5fc2a934f8bbbcd5f6e77a7ba31cf964e697e728cbf1702
TAR	TransitionApplier2.sol	2f4a69406a82fda07ebb796b4d60521dadca2283f180ba9afebb552e076490dc
TDK	TransitionDisputer.sol	23d26abae93df35f026c0368c368117f80119f97862db856e7ec9b507df71196
TEK	TransitionEvaluator.sol	79f5fbbb80d6a7103a4137fa55c0a0ce5ce53e9bae3881fdc269c0910c318aff
DTK	libraries/DataTypes.sol	0a9673af2c843f001bdadfef7e915f42dd4f7334c6c42f9781ed8558a1b2ad37
EMK	libraries/ErrMsg.sol	1c10c33f66b63df49d703cd4cb88f584ea7afb6c816d70bf4890d50846e4eee0
MTK	libraries/MerkleTree.sol	5fdab4525fb4af49e899c61b384828375db8efd8157a97aeb02ae88091f50b05
TRA	libraries/Transitions.sol	a8fabfb145de603eebd98a5a7e0ecb6d8bf2bdb33acd2fd9c760bfd04e8811af
SDK	strategies/StrategyDummy.sol	b7281352edc412121d6c921b0cb4b923c7d9de2ea41d81e2a47aea4a17c3febd

Findings



Critical	0 (0.00%)
Major	0 (0.00%)
Medium	0 (0.00%)
Minor	6 (13.64%)
Informational	38 (86.36%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
DTK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
EMK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
ISK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
IWE-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
MTK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
RCK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
RCK-02	Unnecessary abicoder v2 pragma	Language Specific	Informational	Resolved
RCK-03	Missing visibility specifiers	Language Specific	Informational	Resolved
RCK-04	Unnecessary fallback function	Language Specific	Informational	Resolved
RCK-05	Unrestricted receive function allows anyone to send ETH	Language Specific	Minor	Declined
RCK-06	Arbitrary wETH address parameters	Logical Issue	Minor	Declined
RCK-07	Anyone can induce a withdraw for an arbitrary account	Control Flow	Informational	Declined
RCK-08	Unchecked casting of uint to lower bit count container	Volatile Code	Minor	Declined
RCK-09	Asset not verified to exist in registry before drain transfer	Logical Issue	Minor	Declined

ID	Title	Category	Severity	Status
RCK-10	Owner can drain any ERC-20 when paused	**Centralization / Privilege**	**Informational**	**Acknowledged**
RCK-11	Owner can drain ETH sent to contract when paused	**Centralization / Privilege**	**Informational**	**Acknowledged**
RCK-12	Owner can change block challenge period at any time	**Centralization / Privilege**	**Informational**	**Acknowledged**
RCK-13	Owner can change maximum priority transaction delay at any time	**Centralization / Privilege**	**Informational**	**Acknowledged**
RCK-14	Owner can set net deposit limit for any existing asset at any time	**Centralization / Privilege**	**Informational**	**Acknowledged**
RCK-15	Withdraws larger than net deposit are truncated	Logical Issue	Minor	Declined
RCK-16	Ineffectual statement	Logical Issue	Minor	Resolved
RCK-17	Redundant reading of state variables from storage	Gas Optimization	Informational	Declined
RCK-18	Address parameters not verified	Data Flow	Informational	Declined
RCK-19	Unnecessary local variable declaration	Coding Style	Informational	Resolved
RCK-20	State variables can be declared immutable	Language Specific	Informational	Resolved
REG-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
REG-02	Missing visibility specifiers	Language Specific	Informational	Resolved
REG-03	Redundant reading of state variables from storage	Gas Optimization	Informational	Declined
REG-04	Owner can change strategy implementation at any time	**Centralization / Privilege**	**Informational**	**Acknowledged**
SDK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
TAK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
TAK-02	Unnecessary abicoder v2 pragma	Language Specific	Informational	Resolved
TAR-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved

ID	Title	Category	Severity	Status
TAR-02	Unnecessary abicoder v2 pragma	Language Specific	Informational	Resolved
TDK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
TDK-02	Unnecessary abicoder v2 pragma	Language Specific	Informational	Resolved
TDK-03	Mixed usage of require and returning error messages	Logical Issue	Informational	Resolved
TDK-04	State variables can be declared immutable	Language Specific	Informational	Resolved
TEK-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
TEK-02	Unnecessary abicoder v2 pragma	Language Specific	Informational	Resolved
TEK-03	Missing visibility specifiers	Language Specific	Informational	Resolved
TEK-04	State variables can be declared immutable	Language Specific	Informational	Resolved
TRA-01	Unlocked solidity version pragma	Language Specific	Informational	Resolved
TRA-02	Unnecessary abicoder v2 pragma	Language Specific	Informational	Resolved

DTK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	libraries/DataTypes.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

EMK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	libraries/ErrMsg.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

ISK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	strategies/interfaces/IStrategy.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

IWE-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	interfaces/IWETH.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

MTK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	libraries/MerkleTree.sol: 27	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

RCK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	RollupChain.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

RCK-02 | Unnecessary abicoder v2 pragma

Category	Severity	Location	Status
Language Specific	Informational	RollupChain.sol: 4	Resolved

Description

The linked line explicitly specifies `pragma abicoder v2`, which is unnecessary due to the feature being enabled by default in Solidity v0.8.0 and above.

Recommendation

Consider removing the explicit `pragma abicoder v2` specification on the linked line.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

RCK-03 | Missing visibility specifiers

Category	Severity	Location	Status
Language Specific	Informational	RollupChain.sol: 29, 31	Resolved

Description

The linked lines have missing visibility specifiers, which may result in undesired behavior for sharing between contracts.

Recommendation

Consider adding explicit visibility specifiers to the linked lines.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

RCK-04 | Unnecessary fallback function

Category	Severity	Location	Status
Language Specific	Informational	RollupChain.sol: 131	Resolved

Description

The fallback function in the `RollupChain` contract is unnecessary as it allows nonstandard function selectors and does not contain any actual logic.

Recommendation

Consider removing the fallback function in the `RollupChain` contract.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

RCK-05 | Unrestricted receive function allows anyone to send ETH

Category	Severity	Location	Status
Language Specific	Minor	RollupChain.sol: 133	Declined

Description

The receive function in the `RollupChain` contract allows anyone to send ETH to the contract, which can be unintentional.

Recommendation

Since the receive function is primarily used to receive ETH from the wETH address, consider adding a requirement that the `msg.sender` should only be the well-known wETH address.

Alleviation

The recommendation was not taken into account, with the Celer team responding "We would like to keep the possibility of supporting multiple wETH addresses. For example, Aave has its own WETH."

RCK-06 | Arbitrary wETH address parameters

Category	Severity	Location	Status
Logical Issue	Minor	RollupChain.sol: 156, 190	Declined

Description

The linked lines contain arbitrary address parameters which are intended to represent the wETH address.

Recommendation

Since the wETH address is well-known, consider removing the wETH parameters at the linked lines and utilizing a constant to represent the well-known wETH address instead.

Alleviation

The recommendation was not taken into account, with the Celer team responding "We would like to keep the possibility of supporting multiple wETH addresses. For example, Aave has its own WETH."

RCK-07 | Anyone can induce a withdraw for an arbitrary account

Category	Severity	Location	Status
Control Flow	Informational	RollupChain.sol: 179~182, 190~195	Declined

Description

The external `withdraw` and `withdrawETH` functions in the `RollupChain` contract both take an arbitrary `_account` address parameter instead of utilizing `msg.sender`, which allows anyone to request a withdraw for a particular account without permission for `msg.sender` to do so, which may be undesired by the account owning the funds.

Recommendation

Consider removing the `_account` address parameter and supplying `msg.sender` in its place in the call to the `_withdraw` function.

Alleviation

The recommendation was not taken into account, with the Celer team stating "This is intentional. Since the withdrawal intention information is already recorded on L1. Anyone can pay the gas to confirm the withdrawal."

RCK-08 | Unchecked casting of uint to lower bit count container

Category	Severity	Location	Status
Volatile Code	Minor	RollupChain.sol: 262	Declined

Description

The linked lines may truncate due to casting `uint` values with a higher bit count into `uint` values with a lower bit count without verifying that the `uint` values with higher bit counts are valid within the bounds of a `uint` value with a lower bit count.

Recommendation

Consider adding requirements that the values should be less than or equal to the maximum `uint` value with the lower bit count in order to prevent truncation.

Alleviation

The recommendation was not taken into account, with the Celer team stating "The value is transitions.length, which is the number of transitions submitted as calldata in this single ETH transaction. It is reasonable to assume the number will never reach max_uint32."

RCK-09 | Asset not verified to exist in registry before drain transfer

Category	Severity	Location	Status
Logical Issue	Minor	RollupChain.sol: 417~419	Declined

Description

The external `drainToken` function in the `RollupChain` contract does not verify if the supplied `_asset` address exists within the `registry` prior to the transfer.

Recommendation

Consider requiring the supplied `_asset` address to exist within the `registry` prior to the transfer.

Alleviation

The recommendation was not taken into account, with the Celer team stating "Since drainToken is an emergency function to let the owner withdraw tokens from the contract, we think restrictions of the token address are unnecessary."

RCK-10 | Owner can drain any ERC-20 when paused

Category	Severity	Location	Status
Centralization / Privilege	**Informational**	RollupChain.sol: 417~419	**Acknowledged**

Description

The external `drainToken` function in the `RollupChain` contract allows the owner of the contract to drain any specified ERC-20 token when the contract is in a paused state.

Alleviation

The Celer team stated "Our initial launch will use a single account as the owner. Later the ownership will be transferred to a multi-sign smart contract, which will implement various governance rules, such as advance notification periods for parameter updates."

RCK-11 | Owner can drain ETH sent to contract when paused

Category	Severity	Location	Status
Centralization / Privilege	**Informational**	RollupChain.sol: 427~430	**Acknowledged**

Description

The external `drainETH` function in the `RollupChain` contract allows the owner of the contract to drain any ETH sent to the contract when the contract is in a paused state.

Alleviation

The Celer team stated "Our initial launch will use a single account as the owner. Later the ownership will be transferred to a multi-sign smart contract, which will implement various governance rules, such as advance notification periods for parameter updates."

RCK-12 | Owner can change block challenge period at any time

Category	Severity	Location	Status
Centralization / Privilege	Informational	RollupChain.sol: 436~438	Acknowledged

Description

The external `setBlockChallengePeriod` function in the `RollupChain` contract allows the owner to change the period in which blocks are allowed to be challenged at any time.

Alleviation

The Celer team stated "Our initial launch will use a single account as the owner. Later the ownership will be transferred to a multi-sign smart contract, which will implement various governance rules, such as advance notification periods for parameter updates."

RCK-13 | Owner can change maximum priority transaction delay at any time

Category	Severity	Location	Status
Centralization / Privilege	Informational	RollupChain.sol: 444~446	Acknowledged

Description

The external `setMaxPriorityTxDelay` function in the `RollupChain` contract allows the owner to change the maximum priority transaction delay at any time.

Alleviation

The Celer team stated "Our initial launch will use a single account as the owner. Later the ownership will be transferred to a multi-sign smart contract, which will implement various governance rules, such as advance notification periods for parameter updates."

RCK-14 | Owner can set net deposit limit for any existing asset at any time

Category	Severity	Location	Status
Centralization / Privilege	Informational	RollupChain.sol: 462~466	Acknowledged

Description

The external `setNetDepositLimit` function in the `RollupChain` contract allow the owner to set the net deposit limit for the supplied `_asset` at any time.

Alleviation

The Celer team stated "Our initial launch will use a single account as the owner. Later the ownership will be transferred to a multi-sign smart contract, which will implement various governance rules, such as advance notification periods for parameter updates."

RCK-15 | Withdraws larger than net deposit are truncated

Category	Severity	Location	Status
Logical Issue	Minor	RollupChain.sol: 522~533	Declined

Description

The private `_withdraw` function in the `RollupChain` contract allows the `pendingWithdraws` value for the supplied `_account` and `_asset` to be larger than what is available in `netDeposits` for the supplied `_asset`. This is possible due to setting the `netDeposits` for the supplied `_asset` to zero on L526 and setting the `pendingWithdraws` for the supplied `_account` and `_asset` to zero on L530, effectively truncating the withdraw to only what is available in `netDeposits` for the supplied `_asset` without emitting an event or tracking how much the withdraw was truncated by. The function also returns the amount that was requested in `pendingWithdraws` instead of the actual amount that was available in `netDeposits`, which can affect any functionality relying on the result of the function.

Recommendation

Consider either requiring the `netDeposits` for the supplied `_asset` to be greater than or equal to the `pendingWithdraws` for the supplied `_account` and `_asset`, or consider decrementing the `pendingWithdraws` for the supplied `_account` and `_asset` by what was available in the `netDeposits` for the supplied `_asset` in order to allow a future withdraw to take place to claim the leftover requested amount after a deposit has been made for the supplied `_asset`. Consider also returning the amount that was available in `netDeposits` instead of the amount that was requested in `pendingWithdraws`.

Alleviation

The recommendation was not taken into account, with the Celer team stating "Withdraw will not be truncated. You can see that the `amount` field will not be updated after being set from the storage. It is possible that net deposit is smaller than withdraw amount, because deposits will get earring from the strategies."

RCK-16 | Ineffectual statement

Category	Severity	Location	Status
Logical Issue	Minor	RollupChain.sol: 687	Resolved

Description

The private `_revertBlock` function in the `RollupChain` contract contains an ineffectual statement in a while loop at L687.

Recommendation

Determine if the statement was intended to delete the specified `pendingWithdrawCommits` entry and consider handling the operation correctly or removing the ineffectual statement altogether.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

RCK-17 | Redundant reading of state variables from storage

Category	Severity	Location	Status
Gas Optimization	Informational	RollupChain.sol: 382, 384, 385, 667, 668	Declined

Description

The linked lines read state variables from storage redundantly, which is inefficient.

Recommendation

Consider storing the state variables in a local variable after the initial read from storage in order to save on the overall cost of gas.

Alleviation

The recommendation was not taken into account, with the Celer team stating "We decided to leave the code as it is for simplicity, since registry operations are very rare."

RCK-18 | Address parameters not verified

Category	Severity	Location	Status
Data Flow	Informational	RollupChain.sol: 115~117	Declined

Description

The linked address parameters are not verified to be non-zero before assigning them to state variable which cannot be modified in the future.

Recommendation

Consider adding requirements that the linked address parameters should be non-zero.

Alleviation

The recommendation was not taken into account, with the Celer team stating "These constructor values, which are fine without check because the incorrectly deployed contract will just be discarded."

RCK-19 | Unnecessary local variable declaration

Category	Severity	Location	Status
Coding Style	Informational	RollupChain.sol: 256	Resolved

Description

The linked local variable declaration is unnecessary and its value can be supplied to the call to `blocks.push` on L264.

Recommendation

Consider removing the linked local variable declaration is unnecessary and supplying its value to the call to `blocks.push` on L264.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

RCK-20 | State variables can be declared immutable

Category	Severity	Location	Status
Language Specific	Informational	RollupChain.sol: 29, 31	Resolved

Description

The state variable declared on the linked lines are mutable, yet are only ever assigned to in the contract's constructor.

Recommendation

Consider declaring the state variables on the linked lines as immutable.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

REG-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	Registry.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

REG-02 | Missing visibility specifiers

Category	Severity	Location	Status
Language Specific	Informational	Registry.sol: 9, 10	Resolved

Description

The linked lines have missing visibility specifiers, which may result in undesired behavior for sharing between contracts.

Recommendation

Consider adding explicit visibility specifiers to the linked lines.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

REG-03 | Redundant reading of state variables from storage

Category	Severity	Location	Status
Gas Optimization	Informational	Registry.sol: 37, 40, 38, 53, 54, 56	Declined

Description

The linked lines read state variables from storage redundantly, which is inefficient.

Recommendation

Consider storing the state variables in a local variable after the initial read from storage in order to save on the overall cost of gas.

Alleviation

The recommendation was not taken into account, with the Celer team stating "We decided to leave the code as it is for simplicity, since registry operations are very rare."

REG-04 | Owner can change strategy implementation at any time

Category	Severity	Location	Status
Centralization / Privilege	**Informational**	Registry.sol: 64~74	**Acknowledged**

Description

The owner of the contract can change the implementation address associated with the supplied `_strategyId` at any time

Alleviation

The Celer team stated "Our initial launch will use a single account as the owner. Later the ownership will be transferred to a multi-sign smart contract, which will implement various governance rules, such as advance notification periods for parameter updates."

SDK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	strategies/StrategyDummy.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TAK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	TransitionApplier1.sol: 5	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TAK-02 | Unnecessary abicoder v2 pragma

Category	Severity	Location	Status
Language Specific	Informational	TransitionApplier1.sol: 6	Resolved

Description

The linked line explicitly specifies `pragma abicoder v2`, which is unnecessary due to the feature being enabled by default in Solidity v0.8.0 and above.

Recommendation

Consider removing the explicit `pragma abicoder v2` specification on the linked line.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TAR-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	TransitionApplier2.sol: 5	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TAR-02 | Unnecessary abicoder v2 pragma

Category	Severity	Location	Status
Language Specific	Informational	TransitionApplier2.sol: 6	Resolved

Description

The linked line explicitly specifies `pragma abicoder v2`, which is unnecessary due to the feature being enabled by default in Solidity v0.8.0 and above.

Recommendation

Consider removing the explicit `pragma abicoder v2` specification on the linked line.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TDK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	TransitionDisputer.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TDK-02 | Unnecessary abicoder v2 pragma

Category	Severity	Location	Status
Language Specific	Informational	TransitionDisputer.sol: 4	Resolved

Description

The linked line explicitly specifies `pragma abicoder v2`, which is unnecessary due to the feature being enabled by default in Solidity v0.8.0 and above.

Recommendation

Consider removing the explicit `pragma abicoder v2` specification on the linked line.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TDK-03 | Mixed usage of require and returning error messages

Category	Severity	Location	Status
Logical Issue	Informational	TransitionDisputer.sol: 63, 65, 66, 84, 88, 90, 95~104, 106, 146, 155, 164, 172, 176, 179	Resolved

Description

The external `disputeTransition` function in the `TransitionDisputer` contract contains mixed usage of require statements and returning error messages. In the event of returning an error message, no actual reversion is happening.

Recommendation

Clarify the usage of require versus returning error messages in the `disputeTransition` function.

Alleviation

The Celer team stated "Require statements and error messages have different purposes. A require statement is to check the validity of dispute proof. If it fails, the dispute will fail. An error message is to check the validity of the transition. If an error message is returned, the dispute will succeed."

TDK-04 | State variables can be declared immutable

Category	Severity	Location	Status
Language Specific	Informational	TransitionDisputer.sol: 18	Resolved

Description

The state variable declared on the linked lines are mutable, yet are only ever assigned to in the contract's constructor.

Recommendation

Consider declaring the state variables on the linked lines as immutable.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TEK-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	TransitionEvaluator.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TEK-02 | Unnecessary abicoder v2 pragma

Category	Severity	Location	Status
Language Specific	Informational	TransitionEvaluator.sol: 4	Resolved

Description

The linked line explicitly specifies `pragma abicoder v2`, which is unnecessary due to the feature being enabled by default in Solidity v0.8.0 and above.

Recommendation

Consider removing the explicit `pragma abicoder v2` specification on the linked line.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TEK-03 | Missing visibility specifiers

Category	Severity	Location	Status
Language Specific	Informational	TransitionEvaluator.sol: 17, 18	Resolved

Description

The linked lines have missing visibility specifiers, which may result in undesired behavior for sharing between contracts.

Recommendation

Consider adding explicit visibility specifiers to the linked lines.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TEK-04 | State variables can be declared immutable

Category	Severity	Location	Status
Language Specific	Informational	TransitionEvaluator.sol: 17, 18	Resolved

Description

The state variable declared on the linked lines are mutable, yet are only ever assigned to in the contract's constructor.

Recommendation

Consider declaring the state variables on the linked lines as immutable.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TRA-01 | Unlocked solidity version pragma

Category	Severity	Location	Status
Language Specific	Informational	libraries/Transitions.sol: 3	Resolved

Description

The contract specifies 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 an 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. For example, for version `v0.8.5` the contract should contain the following line:

```
pragma solidity 0.8.5;
```

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

TRA-02 | Unnecessary abicoder v2 pragma

Category	Severity	Location	Status
Language Specific	Informational	libraries/Transitions.sol: 4	Resolved

Description

The linked line explicitly specifies `pragma abicoder v2`, which is unnecessary due to the feature being enabled by default in Solidity v0.8.0 and above.

Recommendation

Consider removing the explicit `pragma abicoder v2` specification on the linked line.

Alleviation

The recommendation was found to be applied in commit `1427c2f4ec601e327d20304ace4df5d1351e3d38`.

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.

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.

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.