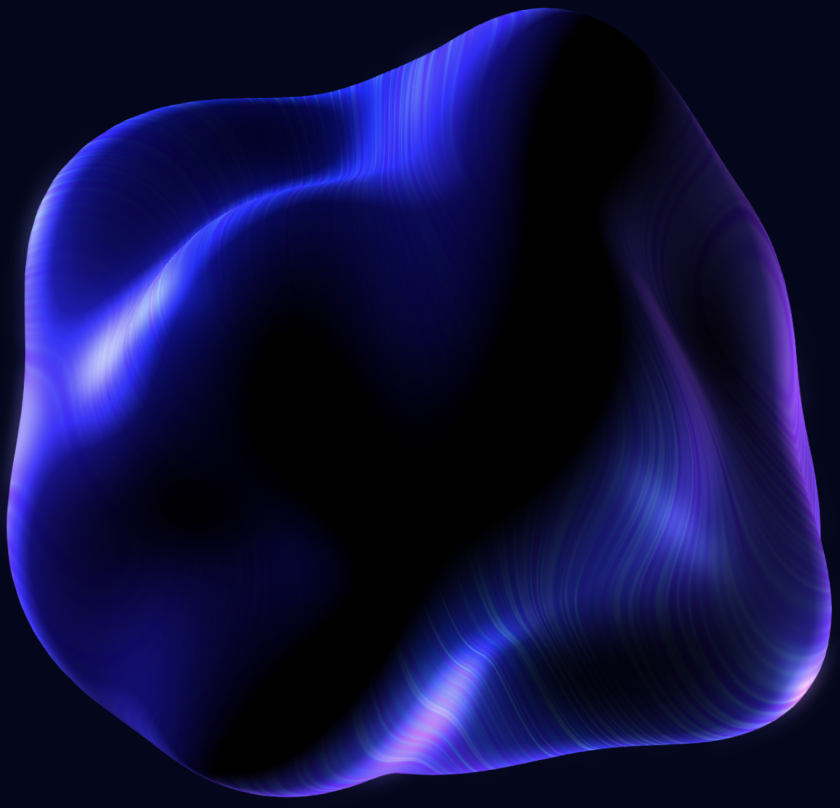
MUNDUS
SECURITY

Security Smart Contract Audit
Lumen Money Chainlink Oracle



MUNDUS
SECURITY

Lumen Money Chainlink Oracle security audit

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.
The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Reference information

Name	Lumen Money Chainlink Oracle
Language	Solidity
Network	NeonEVM
Website	https://www.lumen.money/
Documentation	https://docs.lumen.money/
Repository	https://github.com/Lumen-Money/lumen-protocol
Initial audit commit	3c12a839ab1b1d13fed69a9bde61c92edad4b032

Findings summary

Findings statistics

Severity	Number	Left acknowledged
High	0	0
Medium	2	0
Low	3	1
Informational	1	0
Total	**6**	**1**

Finding Severity breakdown

All vulnerabilities discovered during the source code audit are classified based on their potential severity and have the following classification:

Severity	Description
High	Bugs that can trigger a contract failure or theft of assets. Further recovery is possible only by manual modification of the contract state or replacement of the contract.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Low	Bugs that do not pose significant danger to the project or its users but are recommended to be fixed nonetheless.
Informational	All other non-essential recommendations.

MUNDUS
SECURITY

Project description

Lumen Money Chainlink oracle contract is the main price feed for the Lumen Money lending and borrowing protocol on the NeonEVM network.

Scope of work

Path
oracle/contracts/ChainlinkResilientOracle.sol

Findings

ID	Severity	Description	Status
01	**Medium**	Price feed's `updatedAt > block.timestamp` is allowed	**fixed**
02	**Medium**	`uDecimals` and `fDecimals` give room for human configuration error	**fixed**
03	Low	No check for `asset != address(0)`	ack.
04	Low	The oracle contract is not actually resilient	**fixed**
05	Low	No way to change `_accessControlManager`	**fixed**
06	Info	Unused functions	**fixed**

MUNDUS
SECURITY

Source code audit

ID-01. **Medium**: Price feed's updatedAt > block.timestamp is allowed

Description

The _getChainlinkPrice function (L144) allows the last updated time of the Chainlink price feed updatedAt to be more recent than the Neon EVM's block.timestamp. The comments state that Neon EVM's block.timestamp may lag behind the Solana price feeds updatedAt thus justifying calculation of deltaTime as updatedAt - block.timestamp (L158). This attempt to account for the possible time lag is problematic for the following reason. The protocol is effectively able to look into the future, compared to other protocols deployed on the Neon EVM network. This creates opportunity for unjustified price arbitrage and should not be allowed.

Recommendation

Remove the current deltaTime calculation and add block.timestamp > updatedAdd requirement (see code snippet below).

```
- uint256 deltaTime = block.timestamp > updatedAt
-     ? block.timestamp - updatedAt
-     : updatedAt - block.timestamp;

+ require(block.timestamp > updatedAt);
+ uint256 deltaTime = block.timestamp - updatedAt;
```

Alleviation

The Lumen Money team provided the following explanation as to why the given recommendation is not applicable to the case of NeonEVM network.

There are two execution flows for a transaction: once off-chain by the NeonEVM Proxy. At this stage, the NeonEVM executes and bundles the transactions into a block. After that, the prepared transaction is submitted to Solana, where it executes on the Solana Chain again. NeonEVM not only commits its state to Solana, as any other L2 does, but transpiles the transaction to be executed directly on the Solana Chain in Stage 2. Depending on the congestion of Solana and NeonEVM, it takes some time to reach the second stage. As we read the data (`price` and `updatedAt`) directly from Solana contracts, their account data can change in between Stage 1 and Stage 2. This is why the `block.timestamp` is almost always less than the `updatedAt` value from the Solana contract.

Given the explanation above, the updated recommendation is to split the `maxStalePeriod` parameter into two separate parameters, e.g. `maxStalePeriodChainlink` and `maxStalePeriodNeonEVM`, and compare the time deltas accordingly. The two possible stale periods of different nature, i.e. Chainlink's update lag and NeonEVM's state lag, should be treated differently.

As of commit `1dc2fe46729c3993fdb7a67963ac0ee372c9feb5` the **ChainlinkResilientOracle** contract contains a new parameter `MAX_SOLANA_FUTURE_TIME` and an explicit check `(L148-152)` for the two different cases of stale periods (see code snippet below).

ID-02. **Medium**: uDecimals and fDecimals give room for human configuration error

Description

The TokenConfig data type (L29) contains setting for the underlying asset decimals uDecimals and another setting for the price feed decimals fDecimals. These parameters are configured manually and have direct impact on the resulting internal price calculation. Manual configuration always bears the risk of human error and should be avoid when possible. In the case of token and price feed decimals it is perfectly possible to avoid such risks.

Recommendation

Remove the uDecimals and fDecimals parameters and substitute the corresponding logic with calls to asset.decimals() and AggregatorV3Interface(feed).decimals().

Alleviation

The Lumen Protocol team kept the uDecimals and fDecimals parameters justifying this decision with gas optimization concerns. However, as of commit e9a98ab023e998b238d753e52b095259c7b01b4a, the Lumen Protocol team added additional checks for to the setTokenConfig function (L81-85). These checks remove the possibility of human error during token info configuration.

ID-03. Low: No check for asset != address(0)

Description

The _getPriceInternal function (L121) contains no check for asset != address(0).

Recommendation

Add notNullAddress(asset) modifier to the _getPriceInternal function.

Alleviation

The Lumen Protocol team acknowledges this issue.

ID-04. Low: The oracle contract is not actually resilient

Description

The **ChainlinkResilientOracle** is not actually resilient as it relies exclusively on the correctness of Chainlink's price feed data. This is due to absent logic TWAP logic and manual price configuration logic in the `updatePrice` (L108) and `updateAssetPrice` (L112) functions. Additionally, the comments in the contract's header contain misleading information concerning the oracle's resilience.

Recommendation

Change the name and corresponding information in the comments to simple price oracle.

Alleviation

As of commit `e9a98ab023e998b238d753e52b095259c7b01b4a`, the misleading info in the comments was fixed.

ID-05. Low: No way to change `_accessControlManager`

Description

There is no implemented logic to change the `_accessControlManager` address. This may be problematic in the event of changing the `_accessControlManager` across the protocol.

Recommendation

The imported **AccessControlledV8** contract (L8) contains all the necessary methods implementing the `_accessControlManager` change. Consider changing the **ChainlinkResilientOracle** to inherit from **AccessControlledV8** (see code snippet below).

```
- contract ChainlinkResilientOracle is ResilientOracleInterface
+ contract ChainlinkResilientOracle is AccessControlledV8,
ResilientOracleInterface
```

Alleviation

As of commit e9a98ab023e998b238d753e52b095259c7b01b4a, the oracle contract was changed to inherit from AccessControlledV8.

ID-06. Info: Unused functions

Description

The updatePrice (L108) and updateAssetPrice (L112) functions contain no logic and should be removed (see issue ID-04).

Alleviation

As of commit e9a98ab023e998b238d753e52b095259c7b01b4a, the unused functions were removed.

Disclaimers

Mundus disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical disclaimers

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.