Aluna Social

Security Assessment

November 27th, 2020

For :
Aluna Social

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Aluna Social
Description	The codebase contains the Aluna Token, the Aluna Rewards Pool, the Aluna Token Vesting and the Aluna Boost Pools smart contracts.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	pre-audit: 9edf8d49e8c64be2822ee39fe99827542d8acb38 post-audit: 1be3bf61ba4fe764cc8f38b14cd4feb6bdee39b7

Audit Summary

Delivery Date	November 27th, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	September 7th, 2020 - November 27th, 2020

Vulnerability Summary

Total Issues	43
Total Critical	0
Total Major	0
Total Medium	2
Total Minor	3
Total Informational	38

Executive Summary

The report represents the results of our engagement with the Aluna Social on their implementation of the Aluna Token, the Aluna Rewards Pool, the Aluna Token Vesting and the Aluna Boost Pools smart contracts.

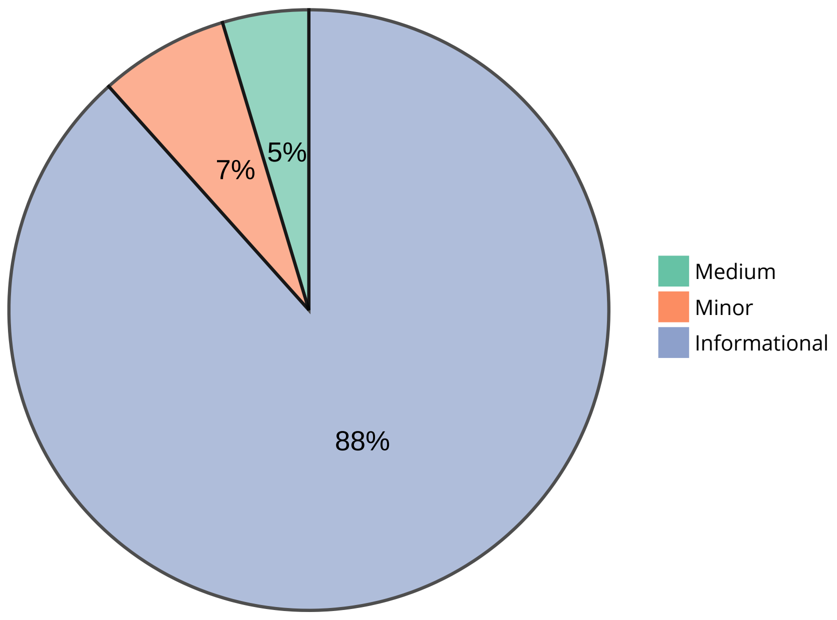
Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the safety of the contract deployement.

Files In Scope

ID	Contract	Location
AGV	AlunaGov.sol	boost-pool/contracts/AlunaGov.sol
ATN	AlunaToken.sol	token/contracts/AlunaToken.sol
AMH	AdditionalMath.sol	boost-pool/contracts/AdditionalMath.sol
ABP	AlunaBoostPool.sol	boost-pool/contracts/AlunaBoostPool.sol
ATV	AlunaTokenVesting.sol	token-vesting/contracts/AlunaTokenVesting.sol
LPT	LPTokenWrapper.sol	boost-pool/contracts/LPTokenWrapper.sol
LPW	LPTokenWrapperWithSlash.sol	boost-pool/contracts/LPTokenWrapperWithSlash.sol
PRR	PaymentReceiver.sol	token/contracts/PaymentReceiver.sol
RPL	RewardsPool.sol	token/contracts/RewardsPool.sol
TRE	Treasury.sol	boost-pool/contracts/Treasury.sol

Findings

Finding Summary



ID	Title	Type	Severity	Resolved
ATN-01	Unlocked Compiler Version	Language Specific	Informational	✓
ATN-02	Introduction of a `constant` Variable	Gas Optimization	Informational	ⓘ
ATN-03	`external` Over `public` Function	Gas Optimization	Informational	✓
ATN-04	Use of `_msgSender()` Function	Volatile Code	Informational	✓
PRR-01	Unlocked Compiler Version	Language Specific	Informational	✓
PRR-02	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
PRR-03	`external` Over `public` Function	Gas Optimization	Informational	✓
PRR-04	Possible Re-Entrancy	Language Specific	Informational	✓
PRR-05	Redundant `require` Statement	Gas Optimization	Informational	✓
RPL-01	Unlocked Compiler Version	Language Specific	Informational	✓
RPL-02	Unused Returned Variable	Volatile Code	Informational	✓
RPL-03	`external` Over `public` Function	Gas Optimization	Informational	✓
RPL-04	Function Optimization	Gas Optimization	Informational	ⓘ
AMH-01	Conditional Optimization	Gas Optimization	Informational	ⓘ
AMH-02	Ambiguous Mathematical Formula	Mathematical Operations	Medium	✓
AMH-03	Unlocked Compiler Version	Language Specific	Informational	✓
LPT-01	`external` Over `public` Function	Gas Optimization	Informational	ⓘ✓

ID	Title	Type	Severity	Resolved
LPT-02	Redundant `import` Statement	Gas Optimization	Informational	✓
LPT-03	State Variables Layout	Gas Optimization	Informational	✓
LPT-04	User-Defined Getters	Gas Optimization	Informational	(!)
LPT-05	Unlocked Compiler Version	Language Specific	Informational	✓
LPW-01	`external` Over `public` Function	Gas Optimization	Informational	(!)✓
LPW-02	Redundant `import` Statement	Gas Optimization	Informational	✓
LPW-03	State Variables Layout	Gas Optimization	Informational	✓
LPW-04	User-Defined Getters	Gas Optimization	Informational	(!)
LPW-05	Unlocked Compiler Version	Language Specific	Informational	✓
TRE-01	Possible Re-Entrancy	Language Specific	Minor	✓
TRE-02	Unused Returned Value	Volatile Code	Informational	✓
TRE-03	Unlocked Compiler Version	Language Specific	Informational	✓
AGV-01	`external` Over `public` Function	Gas Optimization	Informational	(!)✓
AGV-02	Introduction of a `constant` Variable	Gas Optimization	Informational	✓
AGV-03	Possible Re-Entrancy	Volatile Code	Minor	✓
AGV-04	Functions Merge	Gas Optimization	Informational	(!)
AGV-05	State Variables Layout	Gas Optimization	Informational	✓
AGV-06	Unlocked Compiler Version	Language Specific	Informational	✓

ID	Title	Type	Severity	Resolved
ABP-01	Unlocked Compiler Version	Language Specific	Informational	✓
ABP-02	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
ABP-03	Redundant Variable Initialization	Gas Optimization	Informational	✓
ABP-04	`external` Over `public` Function	Gas Optimization	Informational	!
ABP-05	Possible Re-Entrancy	Language Specific	Minor	✓
ABP-06	Ambiguous Mathematical Formula	Mathematical Operations	Medium	✓
ABP-07	Magic Number	Coding Style	Informational	✓
ATV-01	Unlocked Compiler Version	Language Specific	Informational	✓

ATN-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	AlunaToken.sol L1

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.6`.

ATN-02: Introduction of a `constant` Variable

Type	Severity	Location
Gas Optimization	Informational	AlunaToken.sol L13, L14, L15

Description:

The linked variables are assigned values during the initialization phase and are never again updated.

Recommendation:

We advise the team to change the mutability of the linked variables to `constant`.

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation, as they want to follow the initialization pattern and initialize the state variables within the `initialize()` function.

ATN-03: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	AlunaToken.sol L39-L51

Description:

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for the linked function.

Alleviation:

The development team opted to consider our references and used the `external` attribute to the linked function.

ATN-04: Use of `_msgSender()` Function

Type	Severity	Location
Volatile Code	Informational	AlunaToken.sol L49

Description:

Although the sender of the transaction can be accessed via the `msg.sender` global variable, the use of the `_msgSender()` function is preferred.

Recommendation:

We advise the team to use the already exposed `_msgSender()` function of the `Context.sol` contract.

Alleviation:

The development team opted to consider our references and used the exposed `_msgSender()` function to the linked statement.

PRR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	PaymentReceiver.sol L1

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.6`.

PRR-02: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	PaymentReceiver.sol L63

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The development team opted to consider our references and changed to inequality conditional for the linked statement.

PRR-03: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	PaymentReceiver.sol L37-L43, L45-L51, L53-L73, L75-L89

Description:

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for the linked function.

Alleviation:

The development team opted to consider our references and used the `external` attribute to the linked functions.

PRR-04: Possible Re-Entrancy

Type	Severity	Location
Language Specific	Informational	PaymentReceiver.sol L70

Description:

The `payments` mapping gets updated after an external call.

Recommendation:

We advise the team to update the `payments` mapping before depositing to the reward pool, hence following the Solidity safety standards.

Alleviation:

The development team opted to consider our references and updated the state variable before the external call.

PRR-05: Redundant `require` Statement

Type	Severity	Location
Gas Optimization	Informational	PaymentReceiver.sol L96

Description:

The `uint256` parameter `_rewardsPoolPercentage` can take any value from `0` up to `2^256-1`.

Recommendation:

We advise the team to remove the redundant `require` statement.

Alleviation:

The development team opted to consider our references and removed the redundant code.

RPL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	RewardsPool.sol L1

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.5.6`.

RPL-02: Unused Returned Variable

Type	Severity	Location
Volatile Code	Informational	RewardsPool.sol L44, L56

Description:

The returned values of the linked functions are omitted.

Recommendation:

We advise the team to check the returned values of the linked functions, as an internal quality control mechanism.

Alleviation:

The development team opted to consider our references and added a `require` statement to check that the transfer did not fail.

RPL-03: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	RewardsPool.sol L38-L46, L48-L59

Description:

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for the linked functions.

Alleviation:

The development team opted to consider our references and used the `external` attribute to the linked functions.

RPL-04: Function Optimization

Type	Severity	Location
Gas Optimization	Informational	RewardsPool.sol L48-L59

Description:

The `sendRewards()` function seems to offer the same capabilities as the `groupTransfer()` function the is implemented in the Aluna token smart contract.

Recommendation:

We advise the team to use the `groupTransfer()` function of the `AlunaToken.sol`.

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation, as they want the `RewardComplete` event emitted per transfer.

AMH-01: Conditional Optimization

Type	Severity	Location
Gas Optimization	Informational	AdditionalMath.sol L52

Description:

The linked condition redundantly checks the value of the `exponent` parameter, as it will always be different than zero at that point.

Recommendation:

We advise the team to remove the latter condition (`exponent != 0`).

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

AMH-02: Ambiguous Mathematical Formula

Type	Severity	Location
Mathematical Operations	Medium	AdditionalMath.sol L56-L59

Description:

Given the formula in the NatSpec comment section, the `pow()` function is expected to return the value of `a * (b / c)^exponent`. However, due to L56, the `pow()` function returns the value of `a * (b / c)^(exponent+1)`.

Recommendation:

We advise the team to revise the `pow()` function.

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation, as this implementation of the `pow()` function calculates the amount of devaluation of boost price over time.

AMH-03: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	AdditionalMath.sol L21

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

LPT-01: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	LPTokenWrapper.sol L54, L60

Description:

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation:

Use the `external` attribute for functions never called from the contract.

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

LPT-02: Redundant `import` Statement

Type	Severity	Location
Gas Optimization	Informational	LPTokenWrapper.sol L27

Description:

The `Math.sol` contract is redundantly imported.

Recommendation:

We advise the team to remove redundant code.

Alleviation:

The development team opted to consider our references and removed the redundant code.

LPT-03: State Variables Layout

Type	Severity	Location
Gas Optimization	Informational	LPTokenWrapper.sol L37-L40

Description:

The state layout should be as tighly packed as possible to save gas.

Recommendation:

We advise the team to change the structure of the layout to:

```
uint256 private _totalSupply;
IERC20 public stakeToken;
mapping(address => uint256) private _balances;
```

Alleviation:

The development team opted to consider our references and changed the layout of the state variables to strive for a tight packing.

LPT-04: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	LPTokenWrapper.sol L39, L46-L48

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

LPT-05: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	LPTokenWrapper.sol L24

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

LPW-01: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	LPTokenWrapperWithSlash.sol L53, L59, L65

Description:

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation:

Use the `external` attribute for functions never called from the contract.

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

LPW-02: Redundant `import` Statement

Type	Severity	Location
Gas Optimization	Informational	LPTokenWrapperWithSlash.sol L27

Description:

The `Math.sol` contract is redundantly imported.

Recommendation:

We advise the team to remove redundant code.

Alleviation:

The development team opted to consider our references and removed the redundant code.

LPW-03: State Variables Layout

Type	Severity	Location
Gas Optimization	Informational	LPTokenWrapperWithSlash.sol L36-L39

Description:

The state layout should be as tighly packed as possible to save gas.

Recommendation:

We advise the team to change the structure of the layout to:

```
uint256 private _totalSupply;
IERC20 public stakeToken;
mapping(address => uint256) private _balances;
```

Alleviation:

The development team opted to consider our references and changed the layout of the state variables to strive for a tight packing.

LPW-04: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	LPTokenWrapperWithSlash.sol L38, L45-L47

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

LPW-05: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	LPTokenWrapperWithSlash.sol L24

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

TRE-01: Possible Re-Entrancy

Type	Severity	Location
Language Specific	Minor	Treasury.sol L82

Description:

The `ecoFundAmts` mapping gets updated after an external call.

Recommendation:

We advise the team to update the `ecoFundAmts` mapping before transferring the tokens, hence following the Solidity safety standards.

Alleviation:

The development team opted to consider our references and positioned the external call at the end of the function.

TRE-02: Unused Returned Value

Type	Severity	Location
Volatile Code	Informational	Treasury.sol L102

Description:

The `ISwapRouter.swapExactTokensForTokens()` function returns an array `uint256` values, which the `convertToDefaultToken()` function ignores.

Recommendation:

We advise the team to add a `require` statement checking that returned array is not empty.

Alleviation:

The development team opted to consider our references and added a `require` statement to check that returned array is not empty.

TRE-03: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Treasury.sol L24

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

AGV-01: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	AlunaGov.sol L82, L104, L116, L128, L132, L137

Description:

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for the linked functions.

Alleviation:

The development team opted to partially change the codebase based on our references due to inheritance compatibility and time constraints.

AGV-02: Introduction of a `constant` Variable

Type	Severity	Location
Gas Optimization	Informational	AlunaGov.sol L46, L67-L69

Description:

The linked variables are initialized but never updated.

Recommendation:

We advise the team to change the mutability of the linked variables to `constant` to save gas.

Alleviation:

The development team opted to consider our references and changed the mutability of the linked variables to `constant`.

AGV-03: Possible Re-Entrancy

Type	Severity	Location
Volatile Code	Minor	AlunaGov.sol L77-L78

Description:

The `swapRouter` variable gets updated after two external calls.

Recommendation:

We advise the team to update the `swapRouter` variable before approving the two amounts, hence following the Solidity safety standards.

Alleviation:

The development team opted to consider our references and positioned the external calls at the end of the function.

AGV-04: Functions Merge

Type	Severity	Location
Gas Optimization	Informational	AlunaGov.sol L104-L126

Description:

The `voteFor()` and `voteAgainst()` functions can be merged into a single function with an extra boolean parameter to evaluate which scenario to execute.

Recommendation:

We advise the team to implement an external `vote()` function which will be a combination of `voteFor()` and `voteAgainst()` functions:

```
function vote(uint256 id, bool flag) external {
    require(proposals[id].start < block.timestamp , "<start");
    require(proposals[id].end > block.timestamp , ">end");
    uint256 userVotes = AdditionalMath.sqrt(balanceOf(msg.sender));

    if (flag){
        require(proposals[id].againstVotes[msg.sender] == 0, "cannot switch
votes");
        uint256 votes = userVotes.sub(proposals[id].forVotes[msg.sender]);
        proposals[id].totalForVotes = proposals[id].totalForVotes.add(votes);
        proposals[id].forVotes[msg.sender] = userVotes;
    } else {
        require(proposals[id].forVotes[msg.sender] == 0, "cannot switch votes");
        uint256 votes = userVotes.sub(proposals[id].againstVotes[msg.sender]);
        proposals[id].totalAgainstVotes =
proposals[id].totalAgainstVotes.add(votes);
        proposals[id].againstVotes[msg.sender] = userVotes;
    }
    voteLock[msg.sender] = lockPeriod.add(block.timestamp);
}
```

Alleviation:

The Aluna Social development team has acknowledged this exhibit but decided to not apply its remediation, as they opted readability over gas optimization.

AGV-05: State Variables Layout

Type	Severity	Location
Gas Optimization	Informational	AlunaGov.sol L37-L69

Description:

The state layout should be as tighly packed as possible to save gas.

Recommendation:

We advise the team to change the structure of the layout to:

```
    struct Proposal {
        address proposer;
        address withdrawAddress;
        uint256 withdrawAmount;
        mapping(address => uint256) forVotes;
        mapping(address => uint256) againstVotes;
        uint256 totalForVotes;
        uint256 totalAgainstVotes;
        uint256 totalSupply;
        uint256 start; // block start;
        uint256 end; // start + period
        string url;
        string title;
    }

    uint256 public constant MIN_QUORUM_PUNISHMENT = 500;
    uint256 public constant MIN_QUORUM_THRESHOLD = 3000;
    uint256 public constant PERCENTAGE_PRECISION = 10000;
    uint256 public WITHDRAW_THRESHOLD = 1e21
    uint256 public proposalCount;
    uint256 public proposalPeriod = 2 days;
    uint256 public lockPeriod = 3 days;
    uint256 public minimum = 1337e16;

    IERC20 public stablecoin;
    ITreasury public treasury;
    SwapRouter public swapRouter;

    mapping(address => uint256) public voteLock;
    mapping (uint256 => Proposal) public proposals;
```

Alleviation:

The development team opted to consider our references and changed the layout of the state variables to strive for a tight packing.

AGV-06: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	AlunaGov.sol L24

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

ABP-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	AlunaBoostPool.sol L24

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

ABP-02: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	AlunaBoostPool.sol L171, L191, L321

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The development team opted to consider our references and changed to inequality conditionals for the linked statements.

ABP-03: Redundant Variable Initialization

Type	Severity	Location
Gas Optimization	Informational	AlunaBoostPool.sol L49-L50

Description:

The linked `uint256` variables are redundantly initialized to zero, as by simply declaring a `uint256` variable, its default value is zero.

Recommendation:

We advise the team to omit initialization to the linked variables.

Alleviation:

The development team opted to consider our references and omitted the variable initialization to the linked variables.

ABP-04: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	AlunaBoostPool.sol L170, L190, L215

Description:

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for the linked function.

Alleviation:

The development team opted to partially change the codebase based on our references due to inheritance compatibility and time constraints.

ABP-05: Possible Re-Entrancy

Type	Severity	Location
Language Specific	Minor	AlunaBoostPool.sol L323

Description:

The `RewardPaid` event is emitted after an external call.

Recommendation:

We advise the team to emit the `RewardPaid` event before transferring the tokens, hence following the Solidity safety standards.

Alleviation:

The development team opted to consider our references and positioned the external call at the end of the function.

ABP-06: Ambiguous Mathematical Formula

Type	Severity	Location
Mathematical Operations	Medium	AlunaBoostPool.sol L142, L150, L160, L294

Description:

The linked statements synthesize a value increment by a said percentage, yet the addition of a literal, namely `100`, is also part of the procedure.

Recommendation:

We advise the team to either properly document the formula or remove the redundant operation.

Alleviation:

The development team opted to consider our references and updated the in-line comments.

ABP-07: Magic Number

Type	Severity	Location
Coding Style	Informational	AlunaBoostPool.sol L176

Description:

The linked statement contains a magic number, i.e. a literal with no documentation

Recommendation:

We advise the team to add descriptive documentation, explaining the following procedure.

Alleviation:

The development team opted to consider our references and introduced a `constant` variable with a descriptive name.

ATV-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	AlunaTokenVesting.sol L1

Description:

The contract has 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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.12`.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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 exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that 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 and 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.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

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.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.