ARMUR Audit Report

Prepared for Flare Network

PRESENTED TO

Ilan

PRESENTED BY

Akhil Sharma
Amritansh

Mar 13, 2023

Contents

Initial Airdrop.sol

State Connector.sol

Distribution Treasurysol

Governance Settings.sol

Price Submitter.sol

INITIALAIRDROP.SOL ISSUES

- **Lack of access control:** There is only one role that can perform actions in the contract (Governance). However, there is no role-based access control (RBAC) to limit the actions that the Governance can perform. Anyone who has the Governance address can access and modify the smart contract.

- **Lack of input validation:** The smart contract does not validate input parameters received from external calls, making it susceptible to attacks such as integer overflow, underflow, and array out-of-bounds errors.

- **Reentrancy attack**: The smart contract uses the ReentrancyGuard library to prevent reentrancy attacks, but it is only applied to the airdropStarted modifier. Other methods can still be reentrant, which makes the smart contract vulnerable to attacks that exploit reentrancy.

- **Front-running:** The smart contract is vulnerable to front-running attacks since it does not use mutex locks to manage the execution order of transactions that modify the same state variable.

- **Unprotected receive function:** The smart contract has a receive function that allows anyone to send Ether to the smart contract. The function does not have any input validation or access control, making it vulnerable to denial-of-service (DoS) attacks, where an attacker could drain the contract's Eth balance.

INITIALAIRDROP.SOL ISSUES

- **No withdrawal limit:** The smart contract does not have any withdrawal limit, meaning that anyone who has access to the Governance address can withdraw the entire balance of the smart contract.

- **Lack of event handling:** The smart contract does not emit events for certain actions, such as transfers of Ether or changes to the list of airdrop accounts. This makes it difficult to audit the contract's behavior and detect potential security issues.

- **Susceptible to time manipulation attacks:** The contract's behavior depends on the timestamp, and it is possible to manipulate the timestamp by the miner to gain an unfair advantage or cause the contract to misbehave.

- **Integer arithmetic:** The smart contract uses integer arithmetic to calculate the airdrop amount for each account, and there is a risk of integer overflow or underflow if the amounts involved are too large.

- **Lack of code comments:** The code lacks comments, which makes it difficult for other developers to understand the contract's behavior and modify it safely.

STATE CONNECTOR.SOL ISSUES

- **Lack of comments:** There is no or very limited comments in the code which makes it difficult for the readers to understand the code and its functionality.
- **Security vulnerabilities**: The contract uses Solidity version 0.7.6 which has a few security vulnerabilities like the reentrancy bug, that can be exploited by malicious actors. Also, there are no security checks like input validation, range checks, and overflow checks.
- **Unused storage slots**: The contract uses an array of uint256[16] private gap which is not used in the contract. These unused storage slots can be used by attackers to store their own data, thereby increasing the storage costs for the contract owner.
- **Public constant values:** The public constant values SIGNAL_COINBASE, BUFFER_TIMESTAMP_OFFSET, BUFFER_WINDOW, TOTAL_STORED_BUFFERS, and TOTAL_STORED_PROOFS are hardcoded in the contract. These values can be changed by updating the contract code and redeploying it, which makes the contract less flexible.
- **Lack of error handling:** There is no error handling mechanism in the contract which can cause the contract to behave unexpectedly or fail.

STATE CONNECTOR.SOL ISSUES

- **Unused variables**: There are variables in the contract which are defined but not used. These variables can be removed to optimize the contract.

- **Lack of access control:** The functions in the contract can be accessed by anyone, which can be a security risk. Adding access control mechanisms like role-based access control can make the contract more secure.

- **Lack of unit tests:** There are no unit tests written for the contract which makes it difficult to verify the correctness of the code. Writing unit tests can help identify and fix bugs before the code is deployed to the mainnet.

- **Lack of events**: The contract emits only two events. Emitting more events for different actions in the contract can help with debugging and monitoring.

- **No upgradeability**: The contract is not upgradeable which means that any bugs or vulnerabilities cannot be fixed without redeploying the entire contract, which can be costly and time-consuming. Implementing upgradeability can help the contract owner fix any issues without redeploying the entire contract.

- **Hardcoded values:** The values used in the contract like timestamps and addresses are hardcoded which can make the contract less flexible and difficult to maintain. Using variables instead of hardcoded values can make the contract more flexible and easier to maintain.

DISTRIBUTION TREASURY.SOL

ISSUES

- **Reentrancy Attack:** Although there is a reentrancy guard in place, it only prevents reentry within the same block, which means that a malicious attacker can still potentially exploit the contract's pullFunds() function by repeatedly calling it from another contract or function that initiates a new transaction in the same block.

- **Integer Overflow/Underflow:** There is no check for integer overflow/underflow when adding the MAX_PULL_FREQUENCY_SEC value to the lastPullTs timestamp. If an attacker sends a large enough value for MAX_PULL_FREQUENCY_SEC, it could potentially cause an integer overflow/underflow and cause the function to revert or behave unexpectedly.

- **Lack of input validation:** The pullFunds() function does not perform any input validation on the _amountWei parameter, which means that an attacker could potentially pass in a negative value or a value that is greater than the contract's balance.

- **Lack of access control:** Although there is a modifier onlyDistribution, it only restricts access to the pullFunds() function. Other functions in the contract, such as setDistributionContract(), can be called by anyone with the onlyGovernance modifier. This may lead to unauthorized access to critical functions in the contract.

- **Potential Denial of Service Attack:** The MAX_PULL_AMOUNT_WEI constant value is very high, which means that an attacker could potentially exhaust the contract's balance by repeatedly calling the pullFunds() function with the maximum allowed value.

DISTRIBUTION TREASURY.SOL ISSUES

- **Potential gas limit issues:** The _sendFunds() function uses a low-level call, which may cause issues with the gas limit. If the gas limit is exceeded, the transaction will revert and any funds transferred will be lost.
- **Lack of event logging**: There are no events emitted in the contract, which means that it may be difficult to track the contract's state changes and debug any issues that arise.
- **Potential DoS attack:** The receive() function allows anyone to send funds to the contract, which may lead to a denial of service attack if an attacker repeatedly sends small amounts of ether to the contract, causing the contract to run out of gas.
- **Lack of Error message specificity:** Although the contract has error messages, they are not specific enough. For example, the error message "send funds failed" does not provide enough information about why the funds could not be sent.
- **Lack of detailed comments:** Although the contract has some comments, they do not provide enough details about how the contract works and what each function does. This may make it difficult for developers to understand and modify the contract in the future.

GOVERNANCE SETTINGS.SOL

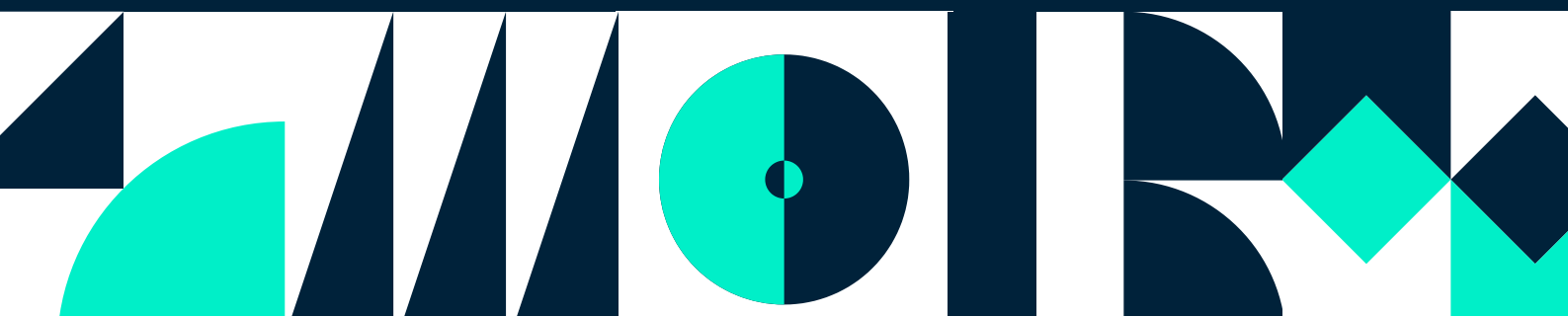
ISSUES

- **Ownership**: There is no ownership functionality defined in this smart contract. Therefore, the governance address is not assigned to anyone, making it vulnerable to attacks. An attacker may compromise the governance address if it is not protected by an owner, and gain control over the entire system.

- **Timelock**: The timelock functionality is defined in the smart contract but not implemented in any function. The timelock functionality is required to prevent unauthorized transactions. The absence of timelock in the functions of the smart contract makes it vulnerable to unauthorized modifications, which may lead to a security breach.

- **Access Control**: The access control mechanism for the functions of the contract is not adequate. The only check for governance access is in the setExecutors function, and there is no check for access control in other functions. A better mechanism for access control is necessary to ensure that only authorized individuals can call the functions that could alter the governance settings.

- **Public Variables**: The governance address and timelock are public variables, which means they can be accessed and modified by anyone. It is necessary to limit the visibility of these variables to ensure that only authorized individuals can access and modify them.



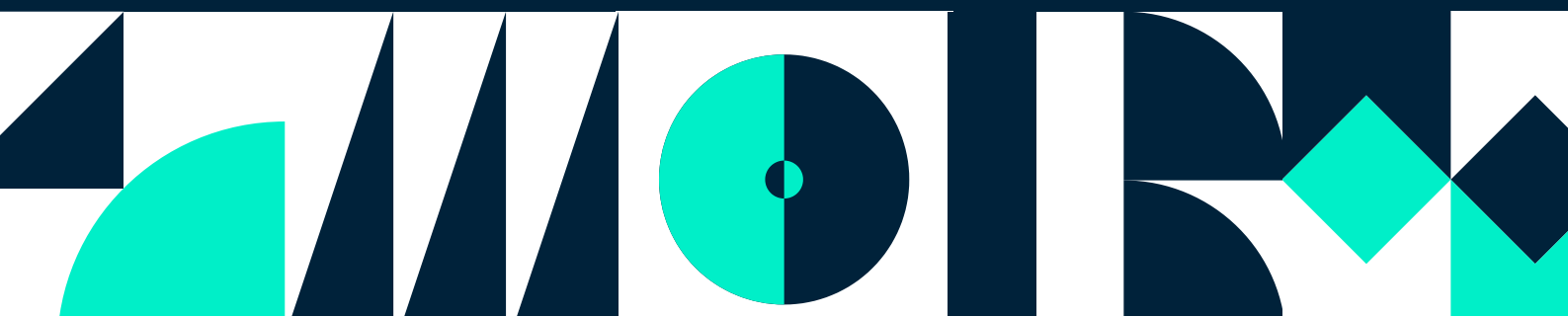
GOVERNANCE SETTINGS.SOL

ISSUES

- **No Input Validation**: There is no input validation for the setExecutors function, which may lead to an attacker adding arbitrary addresses as executors. This may cause a security breach in the governance settings.

- **Reentrancy:** There is no reentrancy guard implemented in any of the functions, which makes the smart contract vulnerable to reentrancy attacks. A reentrancy attack is a type of attack that occurs when a contract calls another contract that maliciously reenters the calling contract, and it is repeated until the contract is drained of its funds.

- **Dependency Risk:** The smart contract is dependent on the IGovernanceSettings interface, and the source code for this interface is not included in this smart contract. Therefore, there may be vulnerabilities in the interface that may impact the security of this smart contract.



PRICE SUBMITTER.SOL

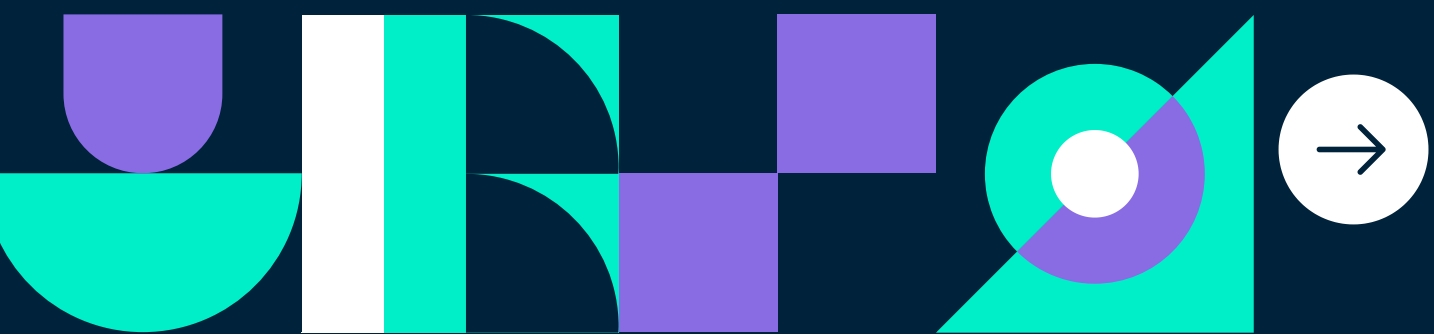
ISSUES

- **Centralization**: The contract is dependent on a few trusted addresses and an FTSO registry and manager, which could potentially act maliciously or be hacked, leading to the loss of user funds.

- **Insecure random number generation**: The random number generator used in this contract has a minimal value of 2^128, which may not be sufficient to provide a secure random number. An attacker may be able to predict the random number and manipulate the price submission, leading to unfair price determination and loss of user funds.

- **Insufficient input validation:** The contract does not validate the input length when setting trusted addresses, which may result in an array overflow or out-of-bounds error. Additionally, the contract does not validate the length of the ftso indices and prices array, which may result in a mismatch with the provided voter information.

- **Incomplete address mapping:** The contract only maps addresses in one direction, which may lead to a situation where a user is mistakenly believed to be whitelisted or trusted, leading to the submission of invalid prices.



PRICE SUBMITTER.SOL

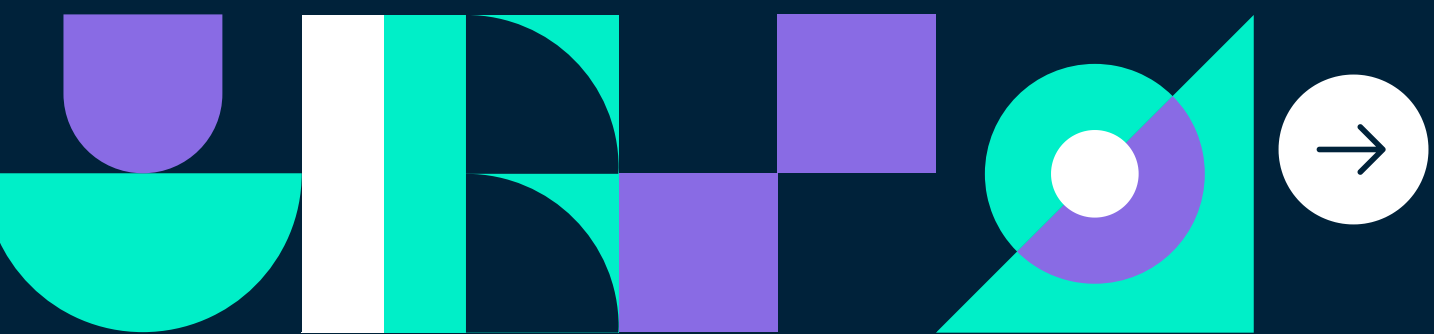
ISSUES

- **Lack of error handling:** The contract does not handle errors sufficiently, leading to potential attacks or loss of user funds. For example, an invalid price may be submitted or a duplicate submit in epoch may occur, and the contract will simply revert without providing information on the cause of the error.

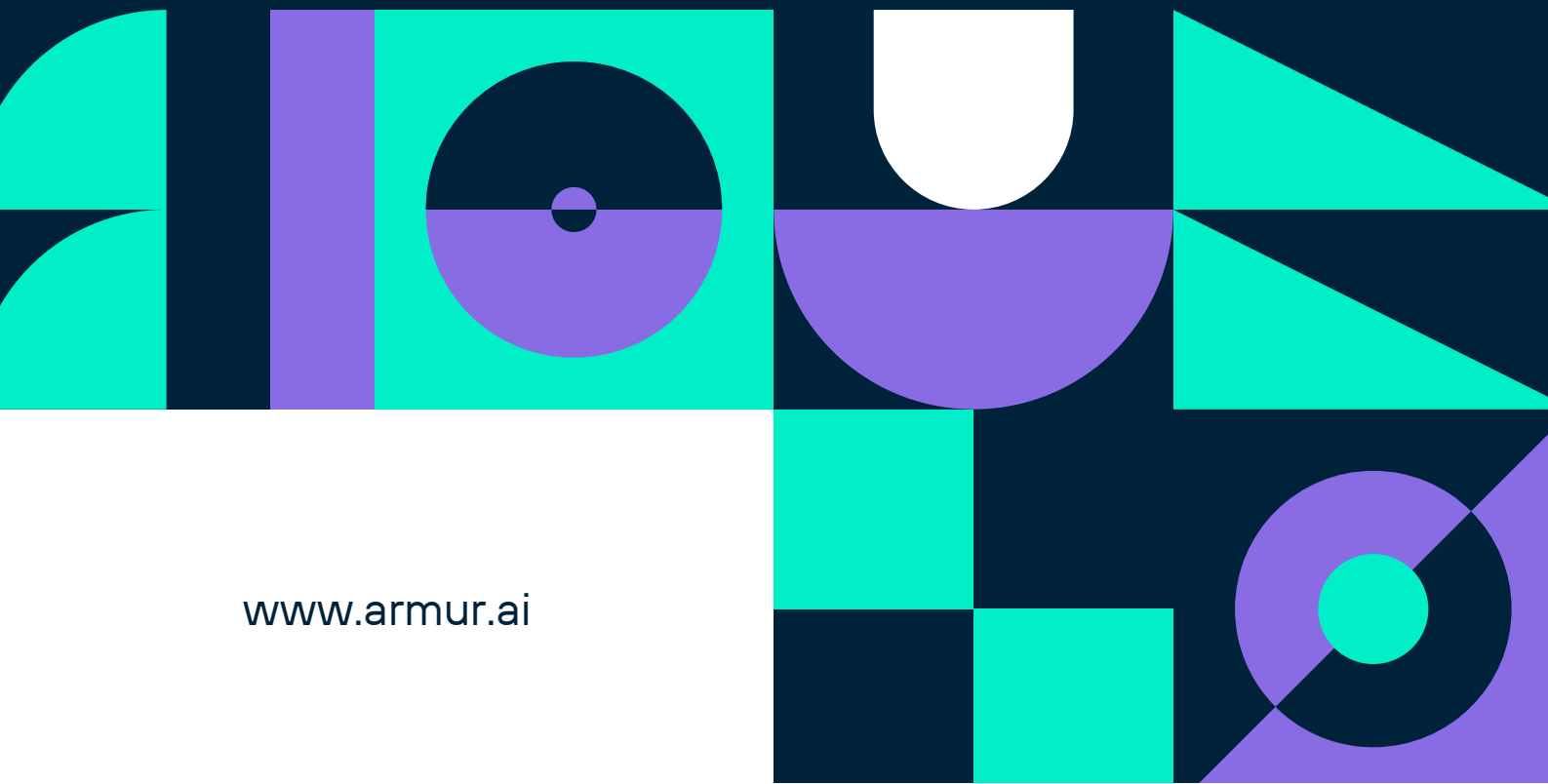
- **Insecure data storage:** The contract stores epochVoterHash values in a public mapping, which may be accessible to attackers and potentially manipulated, leading to the submission of invalid prices and the loss of user funds.

- **Lack of upgradeability:** The contract is not upgradeable, which may lead to the loss of user funds in the event of a security flaw being discovered after deployment.

- **Lack of access control:** The contract does not implement access control mechanisms such as role-based access control or permissions, making it difficult to prevent unauthorized access to sensitive contract functions or data.



Thanks.



www.armur.ai