AUDIT REPORT

PROJECT DETAILS

Project Name:

AletheaAI - AIProtocol - Bonding Curve

Project Scope:

1. ai-protocol-contracts/tree/poc/f_tech_curve/contracts/bonding_curves
2. ai-protocol-contracts/blob/release/2.8-base/contracts/token/OpAliERC20v2.sol

Blockchain:

Ethereum (EVM Compatible Chains)

Language:

Solidity

Report Type:

Initial

Project Summary:

Alethea AI is building a decentralized protocol that will enable the creation of interactive and intelligent NFTs (iNFTs). As originators of the iNFT standard, Alethea AI is on the cutting edge of embedding AI animation, interaction and generative AI capabilities into NFTs. Anyone can use the iNFT protocol to Create, Train and Earn from their iNFTs in the world's first Intelligent Metaverse known as Noah's Ark.

TABLE OF CONTENTS

Audit Findings Summary

#	Finding	Risk	Status
1	Insufficient Upper Bounds Check on Fee Percentage	Medium	Pending
2	Inconsistent Interdependency Between Variables	Low	Pending
3	Deprecated Usage of Native Transfers	Low	Pending
4	Unsafe Method For ERC20 Token Transfer	Low	Pending
5	Unsafe External Calls Can Allow Reentrancy	Informational	Pending

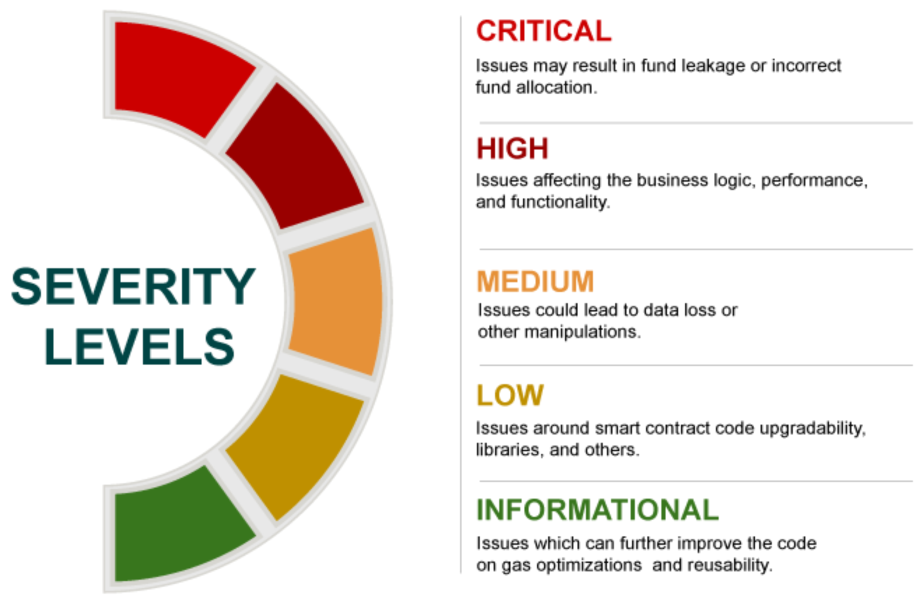
Detailed Results

The contract has gone through several stages of the audit procedure that includes structural analysis, automated testing, manual code review, etc.

All the issues have been explained and discussed in detail below. Along with the explanation of the issue found during the audit, the recommended way to overcome the issue or improve the code quality has also been mentioned.

Security Level Reference

Every issues in this report were assigned a severity level from the following:



CRITICAL
Issues may result in fund leakage or incorrect fund allocation.

HIGH
Issues affecting the business logic, performance, and functionality.

MEDIUM
Issues could lead to data loss or other manipulations.

LOW
Issues around smart contract code upgradability, libraries, and others.

INFORMATIONAL
Issues which can further improve the code on gas optimizations and reusability.

FINDINGS

1. **Issue**:

Insufficient Upper Bounds Check on Fee Percentage

Severity: Medium

Description: The setProtocolFeePercent and setHoldersFeePercent functions lack a validation check for an upper limit on the fee percentage. This oversight allows setting the fee percentage to any value, potentially exceeding 100%.

Impact: Setting excessively high fees, such as 100% or more, could lead to operational issues and user dissatisfaction. It may deter users from interacting with the contract and result in transaction failures if the fee exceeds the transaction amount.

Code-affected:

```
/**
 * @inheritdoc SharesFactory
 */
function setProtocolFeePercent(uint64 _feePercent) public {
        // verify the access permission
        require(isSenderInRole(ROLE_PROTOCOL_FEE_MANAGER), "access denied");
        // verify state change doesn't result into the discrepancy
        require(_feePercent == 0 || protocolFeeDestination != address(0),
"protocolFeeDestination must be set first");
```

```
            // update contract's state
            protocolFeePercent = _feePercent;
            // emit an event
            emit ProtocolFeeUpdated(protocolFeeDestination, _feePercent,
holdersFeePercent, subjectFeePercent);
}
/**
 * @inheritdoc SharesFactory
 */
function setHoldersFeePercent(uint64 _feePercent) public {
            // verify the access permission
            require(isSenderInRole(ROLE_HOLDERS_FEE_MANAGER), "access denied");

            // update contract's state
            holdersFeePercent = _feePercent;

            // emit an event
            emit ProtocolFeeUpdated(protocolFeeDestination, protocolFeePercent,
_feePercent, subjectFeePercent);
}
```

Recommendation: Implement an upper limit validation in the setProtocolFeePercent function. A typical practice is to ensure the fee does not exceed 100%, with a common representation being in basis points (where 10000 basis points = 100%). For example:

```
require(_feePercent <= 10000, "Fee percent too high");
```

2. **Issue**:

Inconsistent Interdependency Between Variables

Severity: Low

Description: The setProtocolFeeDestination function requires that if a non-zero fee destination is set, the protocol fee percent must not be zero. Conversely, the setProtocolFeePercent function requires a non-zero fee destination address to be set before allowing a non-zero fee percentage. This creates a dependency between setting the fee destination and the fee percentage.

Impact: This interdependency restricts the order of operations, potentially limiting flexibility in managing fee settings. It could prevent the independent update of either parameter without first adjusting the other, which may not always be practical or desirable.

Code-affected:

```
/**
 * @inheritdoc SharesFactory
 */
function setProtocolFeeDestination(address _feeDestination) public {
        // verify the access permission
        require(isSenderInRole(ROLE_PROTOCOL_FEE_MANAGER), "access denied");
        // verify state change doesn't result into the discrepancy
        require(_feeDestination != address(0) || protocolFeePercent == 0,
"protocolFeePercent must be set to zero first");

        // update contract's state
```

```
            protocolFeeDestination = _feeDestination;
            // emit an event
            emit ProtocolFeeUpdated(_feeDestination, protocolFeePercent,
holdersFeePercent, subjectFeePercent);
}
/**
 * @inheritdoc SharesFactory
 */
function setProtocolFeePercent(uint64 _feePercent) public {
            // verify the access permission
            require(isSenderInRole(ROLE_PROTOCOL_FEE_MANAGER), "access denied");
            // verify state change doesn't result into the discrepancy
            require(_feePercent == 0 || protocolFeeDestination != address(0),
"protocolFeeDestination must be set first");
            // update contract's state
            protocolFeePercent = _feePercent;
            // emit an event
            emit ProtocolFeeUpdated(protocolFeeDestination, _feePercent,
holdersFeePercent, subjectFeePercent);
}
```

Recommendation:

Consider revising the validation logic in these functions to allow more flexibility. This could involve permitting the independent setting of either the fee percentage or the fee destination, while still ensuring that both parameters are eventually set to valid values before fees are collected.

3. **Issue**:

Deprecated Usage of Native Transfers

Severity: Low

Description: The smart contract complex for bonding curves in the alethea ai-protocol uses the transfer method to transfer ethers to and from the contracts. This pattern of funds transfer has been deprecated in favor of the call method due to the gas limitation potentially resulting in transaction failures.

```
/**
 * @inheritdoc HoldersRewardsDistributor
 */
function claimTheReward() public {
   uint256 claimableAmount = pendingReward(msg.sender);
   require(claimableAmount > 0, "Nothing to claim");

   UserInfo storage userDetail = userInfo[msg.sender];
   // update state variable
   userDetail.unclaimedAmount = 0;
   userDetail.claimedAmount += claimableAmount;
   userDetail.rewardDebt = (userDetail.shares * accRewardPerShare) / 1e18;

   // transfer reward
   if(paymentToken == address(0)) {
      Transfers.transfer(payable(msg.sender), claimableAmount);
   }
```

```
    else {
      require(ERC20(paymentToken).transfer(msg.sender, claimableAmount));
    }


    // emit an event
    emit RewardClaimed(msg.sender, claimableAmount);
  }
```

Recommendation:

It is highly recommended (refer to Open Zeppelin case study) to utilize the call method for transfer all the while ensuring that reentrancy is restricted using the CEI pattern and mutexes.

4. **Issue**:

Unsafe Method For ERC20 Token Transfer

Severity: Low

Description: The smart contract complex for bonding curves in the alethea ai-protocol uses the transfer and transferFrom methods to transfer the token currency. Using transfer has been proven to multiple vulnerabilities that are exploited due to unexpected return values e.g. since the tokens' return values may or may not be handled similarly e.g. the USDT token and few more tokens like that don't returning any boolean for successful transaction or tokens like ERC777 that offer hooks become gateways to vulnerable control flows.

Code Affected:

```
// do the required ERC20 payment token price transfer
require(
        // do not try to transfer zero price
        price == 0 || paymentToken.transferFrom(msg.sender, address(this), price),
        "payment failed"
);
```

Recommendation:

It is highly recommended to utilize the safeERC20 library from openzeppelin for ERC20 token transfers to ensure robustness and compliance with industry best practices.

5. **Issue**:

Unsafe External Calls Can Allow Reentrancy

Severity: Info

Description: Several functions throughout the codebase implement the .call method to interact with external and internal smart contracts in the AIProtocol ecosystem. When manipulating the states of variables within a smart contract, these calls should be safe during the execution for any reentrancy, which can be ensured by following three renowned best practices in the industry. Namely, non-reentrant libraries that utilize mutexes, injecting checks-effects-interactions patterns and pull-over-push design patterns.

Code-affected:

```
/**
 * @inheritdoc TradeableShares
 */
function buyShares(uint256 amount) public payable {
        // delegate to `buySharesTo`
        buySharesTo(amount, msg.sender);
}
/**
 * @inheritdoc TradeableShares
 */
function sellShares(uint256 amount) public {
        // delegate to `sellSharesTo`
        sellSharesTo(amount, payable(msg.sender));
}
```

Recommendation: It's highly recommended that the functions with state manipulation containing external calls be executed under mutexes offered by the Open Zeppelin's implementation of the nonReentrant library.

Disclaimer

ImmuneBytes’s audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program complementing this audit is strongly recommended.

Our team does not endorse the Alethea platform or its product, nor this audit is investment advice.