Alethea AI Version 3.0 Bonding Curves

Audit Report Version 1.1

Darren Jensen

1st January 2024

Alethea AI Version 3.0 Bonding Curves

Darren Jensen

1st January 2024

Lead Security Researcher: - Darren Jensen

Table of Contents

Introduction

This document outlines the findings for smart contract code review for contracts in ai-protocol-contracts repo at commit SHA a5ce10f0 and specifically focuses on the contracts in the `contracts/bonding_curves` folder and the `OpAliERC20v2` token contract. All associated test files and deployment scripts were also reviewed as part of the scope of work.

The purpose of this audit report (Version 1.1) is to act as a remediation review of changes made following the issues raised in Version 1.0 of this report. Therefore, each issue has been reviewed and a new section added outlining the *remediation plan* for that issue.

Overview

Project Name	Alethea AI BondingCurves, TradeableShares & OpAliERC20v2
Repository	ai-protocol-contracts
Commit SHA	a5ce10f0
Documentation	Provided
Methods	Manual review & CLI review (Mythril, Slither, Solhint)

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behaviour with some of the protocol's functionalities that's not so critical.

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions and the cost of the attack is relatively low to the amount of funds that can be stolen or lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a huge stake by the attacker with little or no incentive.

Actions required by severity level

- **High** - client must fix the issue.
- **Medium** - client should fix the issue.
- **Low** - client could fix the issue.
- **Informational** - client could consider design/UX related decision
- **Recommendation** - client could have an internal team discussion on whether the recommendations provide any UX or security enhancement and if it is technically and economically feasible to implement the recommendations
- **Gas Findings** - client could consider implementing suggestions for better UX

Audit Details

Scope

In this report the auditor has focused on all contracts in the `contracts/bonding_curves` directory plus the token/`OpAliERC20v2.sol` contract as follows:

```
AbstractShares.sol
FriendTechBondingCurve.sol
ProtocolFeeDistributorV1.sol
SharesFactoryV1.sol
BondingCurve.sol
SharesSubjectLib.sol
ERC20Shares.sol
HoldersRewardsDistributor.sol
RewardSystem.sol
TradeableShares.sol
ETHShares.sol
HoldersRewardsDistributorV1.sol
SharesFactory.sol
TypedStructLib.sol
token/OpAliERC20v2.sol
```

White Paper Specifications Document

The auditor reviewed the TradableShares README as provided in the shared repo.

LoC (Lines of Code)

The `cloc` utility was used to determine the lines of code under review. The utility excludes empty lines and comments to leave a count of auditable lines of code in each contract. Since all contracts in scope are under the `contracts/bonding_curves` folder the cloc command was run like so:

```
cloc --md contracts/bonding_curves/*.sol
```

The results were as follows:

Language	files	blank	comment	code
Solidity	15	436	2130	1306
———	———	———	———	———
SUM:	15	436	2130	1306

SHA-256 File Fingerprints

To generate the SHA-256 fingerprint for all the smart contracts in a directory run:

```
shasum -a 256 contracts/bonding_curves/*.sol
```

Which outputs the following:

```
6a31f5a6320cba40bba8b2b7b6c2c1c86e88950bdb76d1464197ee5e6854a050  AbstractShares.sol
4829fc5b38ec0fdbabb6f7a59bf1fc495ac849fed33698a4bc21e1680e66a4ca  Aliases.sol
714f43a2392767fd7ce1b73fbbbe8239c8be7b0ae5f94dd7877fccccc7eb1704  BondingCurve.sol
fb65535284b94cd67251b3b6d017fa78f2b5857ce92f9fa74785f7096f89030e  ERC20Shares.sol
a930f20c86c1554ee927cd73198dcc67cc69c743c5ba60ad6e9d62b1f0771860  ETHShares.sol
69bfe888ea8dde457addf80258cf46770d10ad9656d3d2704bd22a12c8b00ba6  FriendTechBondingCurve.sol
fe0359ef666dd3ed13569163bf2883724e02768e3933e99177ea6376f01cb4de  HoldersRewardsDistributor.sol
c65e4da925acb2bcce15207b5761ca231b93452c164345f7c47f50151f76f3b1  HoldersRewardsDistributorV1.sol
d58d421cc41a960f0534cd9f49151cc0d3c25cf167fa3b177ec5d2ae299db398  ProtocolFeeDistributorV1.sol
e694bc7d2b74334347e25b832d976b3595c3fe3021114665d70039e9bb7ec94c  RewardSystem.sol
5c6ad032d4d723deb8fead7668b2bb5fe1b819dd658fa0fd4bad16260fe583fd  SharesFactory.sol
1ab33785e5f81714390106b4a99ea4226c04004ed6aaff7186d4306b7e2b68da  SharesFactoryV1.sol
015070e328613c594e64a7e218d2d44318327af9d93934a2cd083dda83bf26d4  SharesSubjectLib.sol
91212fafdfa3f2baf2115cb170328ec176c24a70aabcc5280f3d19da8462f7b4  TradeableShares.sol
4de59b2266190e4b891d2d2698a72acd7e45b0ddddb20a75388a7bdca6f81096  TypedStructLib.sol
```

Test Run

The auditor built the contracts using hardhat and ran all the tests under `contracts/bonding_curves` which are all passing.

All the contracts were compiled and the test run executed successfully with all tests passing.

Test Coverage

Below shows the output of the test coverage report for the bonding curve contracts. Its worth noting that the overall bonding curves test coverage has *reduced slightly* due to the intruction of the `Aliases.sol` contract which reports as having zero coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
bonding_curves/	98.01	91.46	94.4	98.43	
AbstractShares.sol	100	90	100	100	
Aliases.sol	0	100	0	0	40,47,65,72
BondingCurve.sol	100	100	100	100	
ERC20Shares.sol	100	90.63	100	100	
ETHShares.sol	100	100	100	100	
FriendTechBondingCurve.sol	100	100	100	100	
HoldersRewardsDistributor.sol	100	100	100	100	
HoldersRewardsDistributorV1.sol	100	85.19	100	100	
ProtocolFeeDistributorV1.sol	96.97	87.5	87.5	97.5	231
RewardSystem.sol	100	85.71	100	100	
SharesFactory.sol	100	100	100	100	
SharesFactoryV1.sol	99	95.08	100	99.17	688
SharesSubjectLib.sol	94.74	83.33	100	95.45	92
TradeableShares.sol	100	100	100	100	
TypedStructLib.sol	100	100	100	100	
———————————	———-	———-	———-	———-	————
All files	97.93	92.51	96.63	97.93	
———————————	———-	———-	———-	———-	————

Linting

Linting is a valuable tool for finding potential issues in smart contract code. It can find stylistic errors, violations of programming conventions, and unsafe constructs in your code. There are many great

linters available, such as Solhint. Linting can help find potential problems, even security problems such as re-entrancy vulnerabilities before they become costly mistakes.

After installation, Solhint can be run via the terminal as follows.

```
solhint 'contracts/bonding_curves/*.sol'
```

This command will run solhint for the contracts in the root of the project directory. I have run the above command and included the output in the Appendix section of this report. The only issue identified by solhint was the length of the line in some instances.

I recommend running the solhint linter, either via a Git commit hook or as part of an integration with the developer IDE so that the recommendations can be checked on every code change. NOTE: the above issues are low priority and would not have any impact if not modified.

Slither Security Analysis

Auditor uses Slither (version 0.9.6) static analyzer tool. The slither cli was run against all contracts in scope of the project.

For each of the above contracts the correct version of `solc` using `solc-select` and then run the `slither` command to analyze specific contract under test:

```
solc-select use 0.8.15

slither --checklist --exclude-informational --exclude-low --solc-remaps
    "@openzeppelin/=node_modules/@openzeppelin/" contracts/
   bonding_curves  2>&1 | tee slither-bonding-curves.md
```

The findings were checked and, if appropriate, included in the Executive Summary of this report. The final Slither report is also available as a separate file included with this report.

Mythril Security Analysis

The Mythril security analysis reports were run using the MythX service in standard mode using the following commands:

```
mythx analyze --mode standard --remap-import "@openzeppelin/=$(pwd)/
   node_modules/@openzeppelin/" --solc-version 0.8.15 contracts/
   bonding_curves/*.sol
```

The analysis was performed via the MythX services and the pdf reports generated and downloaded.

NOTE: One issue is the `totalClaimedReward` variable in `RewardSystem` does not have its visibility set making the default visibility of this variable `internal`. Only other issues were around floating pragma.

Each contract has its own individual report (that can be downloaded as PDF) for each submitted contract to the MythX service. These reports are included in a separate ZIP file along with this main report.

Aderyn Security Analysis

Auditor uses Aderyn (version 0.0.10) static analyzer tool. The `aderyn` cli was run against all contracts in scope of the project.

Run the `aderyn` command to analyze specific contracts under test as follows:

```
1 aderyn .
```

The findings were checked and, if appropriate, included in the Executive Summary of this report. The final Aderyn report is also available as a separate file included with this report.

Executive Summary

Issues found

Severity	Number of issues found	Resolved / Acknowledged
High	0	-
Medium	2	2
Low	4	4
Info	17	17
Gas	3	3
Total	26	26

NOTE: All issues identified in Version 1.0 of this audit have been either resolved or acknowledged by the Alethea AI team.

Findings

Medium

[M-1] Potential to front run in `HoldersRewardsDistributorV1` affecting `__accept` function

Context

Github

Description:

It's possible for a malicious user to affect the `accRewardPerShare` calculation by calling the `receive()` function in the same block before any buy / sell tx is mined. This would set the `lastRewardBlock` to the current block and the `feeAmount` sent into the contract would not be added to the `accRewardPerShare`.

Impact:

Can potentially prevent fees being collected by the contract.

Recommended Mitigation:

Consider protecting this function from front running. However, it's unlikely that an attacker would benefit from such an action other than purposely reducing the rewards for holders.

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: 5d41ce78

[M-2] If owner (via Factory `sharesOwnerAddress` function) is not set in the shares contract then it's not possible to manage

Context:

Github

Description:

In the `__initSharesContract` function of the `SharesFactoryV1` contract the shares contracts are deployed using the `sharesOwnerAddress` as the owner. However, if the `sharesOwnerAddress` is a zero address then the deployed sharers contract will not be possible to manage.

Impact:

It will not be possible to manage the deployed shares contract.

Recommended Mitigation:

Ensure that the `sharesOwnerAddress` is set to a valid address to use as the owner of the shares contracts before calling the `__initSharesContract` function. Moreover, consider checking the deployment scripts to set the `sharesOwnerAddress` via a call to `setSharesOwnerAddress`.

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"This is by design. We want to be able to switch the shares contract deployment process into non-manageable mode."

Low

[L-1] `ProtocolFeeDistributorV1` contract `updateRecipientsList` function does not check for duplicate recipient address

Context:

Github

Description:

It's possible to set duplicate recipients in `updateRecipientsList` function including setting all the recipients to the same address.

Impact:

Duplicates can lead to mis-use of funds.

Recommended Mitigation:

Consider performing a uniqueness check in the input addresses of the `_recipients` array.

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"Yes, this is correct. We believe checking the sum of all the recipient shares to be 100% is enough."

[L-2] Check the shares contract type before setting in `HoldersRewardsDistributorV1`

Context:

Github

Description:

Check the shares contract type before setting in the contract storage variable. If the `paymentToken` address is set in the `HoldersRewardsDistributorV1` contract then the shares contract should be `ERC20Shares` type, otherwise `ETHShares` type.

Impact:

If the wrong shares type is set it can lead to incorect functionality of the protocol.

Recommended Mitigation:

Add a validation for sharesContractAddress points to a shares contract of the expected type (`ERC20Shares` or `ETHShares`).

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"*The finding is valid. A simple fix would break many tests however. We have the tests covering factory behaviour which ensures that the factory cannot do this mistake of deploying different versions of the shares and distributor contracts and attaching them together.*"

[L-3] The `RewardSystem` proxy deployment sets the `rewardSystemType` incorrectly

Context:

Github

Description:

In the deployment script for the `RewardSystem` proxy the type is set to `TRUE` when it should be set to `FALSE` given that an`ERC20` token is set then it suggests this is an `ERC20` `rewardSystemType` deployment.

Impact:

Could lead to a false positive when identifying the reward system type in the protocol.

Recommended Mitigation:

Consider updating the deployment script so that the `rewardSystemType` is set correctly in the deployment.

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: `1fab9d37` stating also:

"This was fixed outside Darren's audit resolution, as part of the Miguel's audit resolution."

[L-4] The `RewardSystem` proxy deployment is missing the ETH `rewardSystemType`

Description:

There does not appear to be any deployment of the `RewardSystem` proxy for the ETH `rewardSystemType`.

Impact:

Could impact the protocol if there is a partial deployoment of contracts or settings.

Recommended Mitigation:

Consider adding a deployment script for the ETH `rewardSystemType`.

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"We plan to deploy only one system eventually – either ETH, or ERC20. We will update the deployment script once we chose."

Informational

[I-1] Validate the bridge address is trusted before updating role

Context:

Github

Description:

The role is applied to the bridge contract before validating the address is a known trusted bridge.

Impact:

If the bridge is untrusted it could cause loss of funds, gas griefing, locked ether or other issues.

Recommended Mitigation:

Consider validating the bridge before assigning the role.

Remediation Plan

ACKOWLEDGED - NOT VALID: The Alethea AI team have acknowledged the issue stating:

"The bridge is set during the deployment and its address is picked from hardhat.config, which is assumed to have the highest trusting authority during the deployment process."

[I-2] Bonding curve price function differs slightly from the FriendTech version

Context:

Github

Description:

Calculation for `sum2` is slightly different to the FriendTech version which sets `sum2` to 0 only when supply is 0 and amount is 1. In the AletheaAI implementation this has been changed to amount is `<=1`.

Impact:

Incorect pricing calulcations in bonding curve.

Proof of Concept:

AletheaAI version

```
uint256 sum2 = s == 0 && a <= 1 ? 0 : (s + a - 1) * (s + a) * (2 * (s +
    a - 1) + 1) / 6;
```

FriendTech version

```
uint256 sum2 = s == 0 && a == 1 ? 0 : (s + a - 1) * (s + a) * (2 * (s +
    a - 1) + 1) / 6;
```

Recommended Mitigation:

Consider aligning the versions to be exactly the same.

Remediation Plan

ACKOWLEDGED : The Alethea AI team have acknowledged the issue stating:

"Yes, the original function is not defined in (0, 0) and we've fixed that."

[I-3] Missing detail in comment for `keccak256` value

Context:

Github

Description:

Missing comment showing the `keccak256` used to generate the hash.

Recommended Mitigation:

Consider adding comments as is for the `SharesSubject` type hash.

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: 791e15b8](https://github.com/AletheaAI/ai-protocol-contracts/commit/791e15b8).

[I-4] Outdated comment in `HoldersRewardsDistributor`

Context:

Github

Description:

The comment regarding the `fallback` is outdated. The encoded data includes an `isBuy` bool and the amount is always positive (`uint256`).

Recommended Mitigation:

Consider updating the comment to reflect existing behaviour.

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: ce53f40a.

[I-5] Naming convention for contract interfaces

Context:

Github

Description:

It's not clear from the name of the contract this is an interface.

Recommended Mitigation:

Consider using 'I' prefix for all interface names. So, for example, renaming `HoldersRewardsDistributor` to `IHoldersRewardsDistributor`.

Remediation Plan

ACKOWLEDGED - NOT VALID: The Alethea AI team have acknowledged the issue stating:

"From the very beginning, we try to stick to JavaScript naming conventions in the areas where explicit naming conventions for Solidity don't exist. The I prefix used by OpenZeppelin and many other projects comes from the C/C++/C#/.NET naming conventions, while we stick to Solidity/JavaScript/Java conventions."

[I-6] TODO comments in contract code

Contaxt:

Github

Github

Description:

`TODO` comment in code suggests a potential change to the `claimTheReward` function.

Recommended Mitigation:

Consider removing the `TODO` comment.

Remediation Plan

SOLVED: The Alethea AI team have removet the TODO comments.

[I-7] Include an `.nvmrc` file to set the node version to 16 for devs

Description:

Developers can benefit from an `.nvmrc` file at the root of the project folder so that can potentially trigger NVM to automatically switch to the version of Node JS as specified in the `.nvmrc` file.

Recommended Mitigation:

Consider adding an `.nvmrc` file to the repo with the contents set to the current version of the Node being used. The file can be generated like so and then added to the project git repo:

```
node -v > .nvmrc
```

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: 84672dc2.

[I-8] `describe` block in test could be an `it` block

Context:

Github

Description:

The `describe` block in this example test could be an `it` block.

Recommended Mitigation:

Consider changing to an `it` block.

Remediation Plan

ACKOWLEDGED - NOT VALID: The Alethea AI team have acknowledged the issue stating:

"The describe block mentioned contains other it blocks inside it and cannot be converted into it itself."

[I-9] Consider using `console.warn`

Context:

Github

Description:

In the gas optimization tests there is a `console.log` output for when the gas used is less than the expected amount. Using `console.warn` may be better to highlight this issue when it is included in the test logs.

Recommended Mitigation:

Consider changing `console.log` to `console.warn`.

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: 625db496.

[I-10] JS version of `get_price` in tests is slightly different to the `getPrice` implementation in `FriendTechBondingCurve`

Context:

Github

Description:

This appears only when the given supply is > 1 and the amount is > 0 then the resulting price from these function calls always differs.

Impact:

Could lead to false positives in test run.

Proof of Concept:

```
describe("Checking impl of get price in JS vs Solidity", function() {
        let bc;
        beforeEach(async function() {
            const FriendTechBondingCurve = artifacts.require("
               FriendTechBondingCurve");
            bc = await FriendTechBondingCurve.new()
        });
        it.only("calling get_price in JS vs getPrice in Solidity", async
             function() {
            const supply = 2;
            const amount = 1;


            priceJs = await get_price(supply, amount);
            console.log("Price (JS): ", priceJs.toString());
            priceSol = await bc.getPrice(supply, amount);
            console.log("Price (SOL): ", priceSol.toString());
        });
    })
```

The result of the above POC test is:

```
Price (JS):  312500000000000
Price (SOL):  250000000000000
```

Recommended Mitigation:

Correct the JS implementation of `get_price` used in the tests to exactly match the implementation of the Solidity implementation of `getPrice`.

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: `b202e1c9`.

[I-11] Missing associated parent or Interface contract

Context:

Github

Description:

There is no corresponding parent or Interface contract as there are with the contracts (e.g. `HoldersRewardsDistrib` has the `HoldersRewardsDistributor` Interface)

Recommended Mitigation:

Consider adding a `IProtocolFeeDistributor` contract for the `ProtocolFeeDistributorV1` .

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"Yes, we stick to the idea of not adding and interface, until it is really required and it becomes hard or inconvenient to avoid having it."

[I-12] `receive` function does not need to be marked `virtual`

Context:

Github(https://github.com/AletheaAI/ai-protocol-contracts/blob/efef8eecd87f18b88ef8ceee7dc68be73664f187/contrac

Description:

The `receive` function in the `ProtocolFeeDistributorV1` contract does not need to be marked as `virtual`.

Recommended Mitigation:

Consider removing the `virtual` keyword.

```
- receive() external payable virtual
+ receive() external payable
```

Remediation Plan

ACKOWLEDGED - NOT VALID: The Alethea AI team have acknowledged the issue.

[I-13] Emit event when `sharesContractAddress` is set in `HoldersRewardsDistributorV1`

Context:

Github

Description:

The `sharesContractAddress` is set in `HoldersRewardsDistributorV1` without any event being fired.

Recommended Mitigation:

Consider emitting an event when `sharesContractAddress` is set in `HoldersRewardsDistributorV1` contract.

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"We do not emit events in constructors and in functions which replace or adjust the constructors."

[I-14] Possible to pass any nonce to `rewindNonce` function that is greater than the current which would leave gaps

Context:

Github

Description:

In the `rewindNonce` function of the `SharesFactoryV1` contract it's possible to pass in any nonce that is greater than the current which would leave gaps.

Recommended Mitigation:

Consider restricting the new nonce to be exactly one greater than the current nonce.

Remediation Plan

ACKOWLEDGED - NOT VALID: The Alethea AI team have acknowledged the issue stating:

"*This is a correct behaviour implied by the function name and its signature. The function allows to discard any number of previously issued signatures.*"

[I-15] Comment mentions `ROLE_PROTOCOL_FEE_MANAGER` but the values set is for `ROLE_SUBJECT_FEE_MANAGER`

Context:

Github

Description:

Comment refers to a different variable name.

Recommended Mitigation:

Consider correcting the comment to properly reflect the variabile name.

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: 70d75806.

[I-16] Potentially missing a deployment dependency in `setup-SharesFactory` script.

Context:

Github

Description:

Perhaps `upgrade-ProtocolFeeDistributorV1` should also be listed as a dependency in this script.

Recommended Mitigation:

Consider adding `upgrade-ProtocolFeeDistributorV1` also be listed as a dependency in this script.

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"*The finding is correct. Right now the setup-SharesFactory script is acts as a placeholder. We will add mentioned dependency once we release a real factory upgrade.*"

[I-17] `determineImplementationType` will always return `ImplementationType.ETH` if any other address is passed

Context:

Github

Description:

If any address is passed into the function (except a valid `ERC20` shares address) then the function will always return `ImplementationType.ETH` result.

Recommended Mitigation:

Consider performing a check for a valid `TradeableShares` contract before defaulting to `ImplementationType.ETH`.

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"*determineImplementationType function is just a hint, it cannot guarantee anything. It is used in event only. From the factory point of view the shares contract having different ERC20 address is completely invalid, it is neither ETH, nor ERC20; but this is not so important at the moment, since the factory is upgradeable we reserve a possibility to upgrade this function if it becomes important.*"

Gas

[G-1] Can reuse sharesSupply value

Context:

Github

Description:

The calculation for `sharesSupply` is performed twice.

Recommended Mitigation:

The `sharesSupply` can be passed directly to the `getPrice` function to save a little gas

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: db0446a4.

[G-2] Function call can be avoided by inline code

Context:

Github

Description:

Calling `pendingReward` in the `claimTheReward` function can be avoided by moving this function code inline and therefore avoiding the function hop which will save some gas.

Recommended Mitigation:

Consider moving the `pendingReward` function logic inline of the `claimTheReward` function to save some gas when calling this function.

Remediation Plan

ACKOWLEDGED: The Alethea AI team have acknowledged the issue stating:

"*pendingReward function is public and can be used externally, thus cannot be removed/inlined.*"

[G-3] Checking issuer address is not `address(0)` multiple times

Context:

Github

Description:

The assert statement (`assert(issuer != address(0));`) comes after a require statement that is run if the issuer is a zero address.

Recommended Mitigation:

Consider removing the `assert` statement on the line shown in the context link above.

Remediation Plan

SOLVED: The Alethea AI team solved the issue in the following commit: ce53f40a6.

Conclusion

All contracts reviewed in scope are well written and organized.

All issues identified in Version 1.0 of this audit have either been fixed or ackowledged by the Alethea AI team. This has been checked and verified by the auditor and notes made under the *remediation plan* of each setion of the findings report. It was also noted that the overall bonding curves test coverage has reduced slightly due to the intruction of the `Aliases.sol` contract.

It's still worth considering further analysis, by the auditor, of the contracts logic and state by performing some specific invariant tests against the contracts such as performing thousands of buy / sell trades and checking that certain invariants hold. This work could be carried out as part of a subsequent report of work by the auditor using invariant testing tools in Foundry.

Appendix

Suyra Contract Interaction & Inheritance Graphs

See Figure 1, 2, 3 & 4 for details.

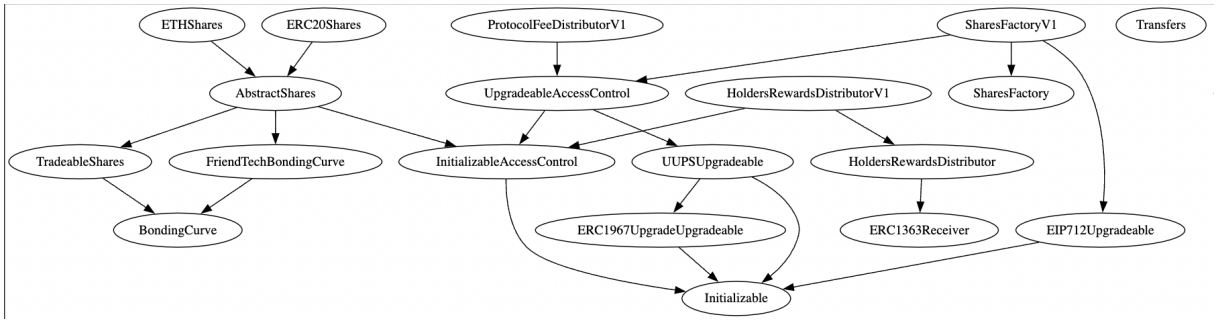

Figure 1: Alethea AI Version 3 Inheritance

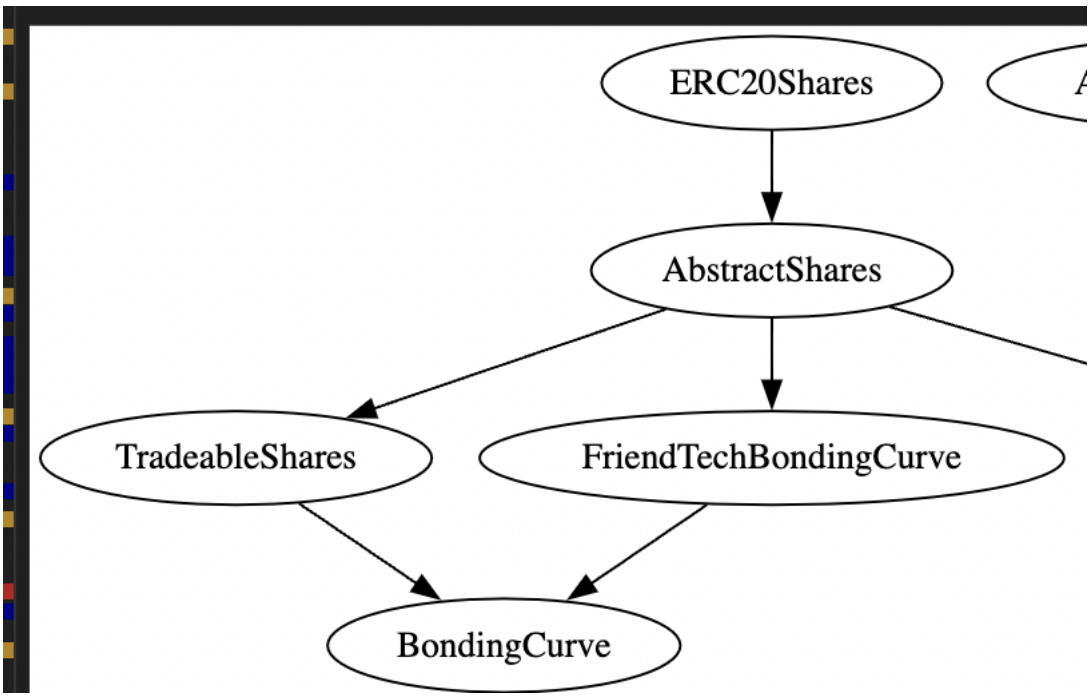

Figure 2: ETH Shares Inheritance

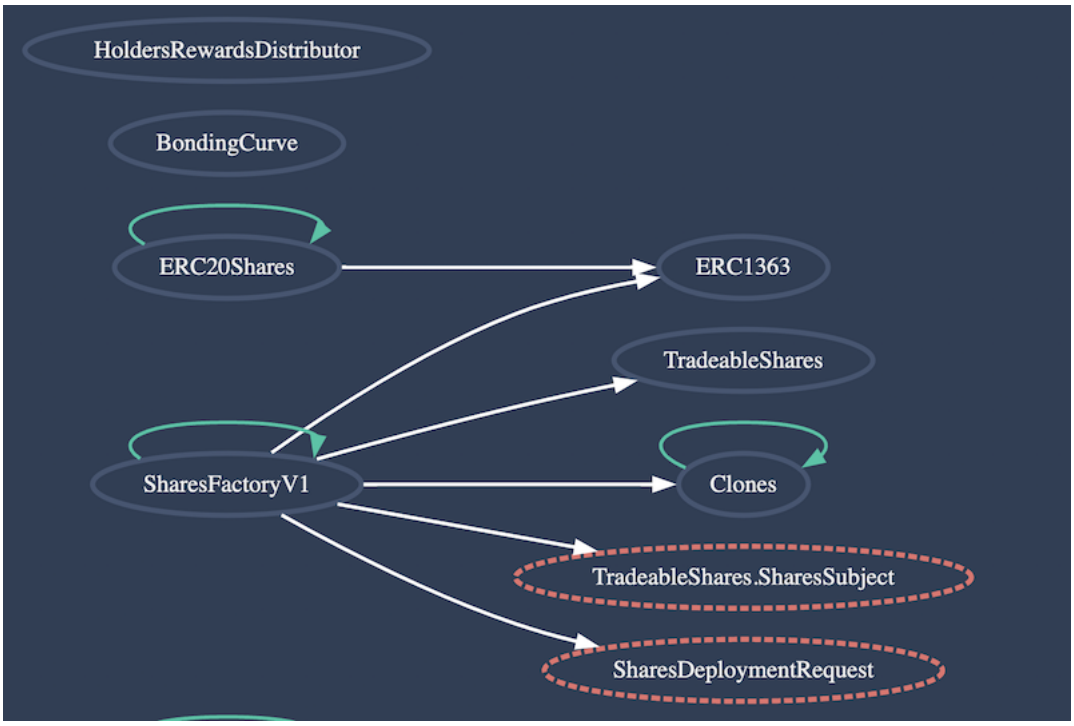

Figure 3: Contract Interactions 1

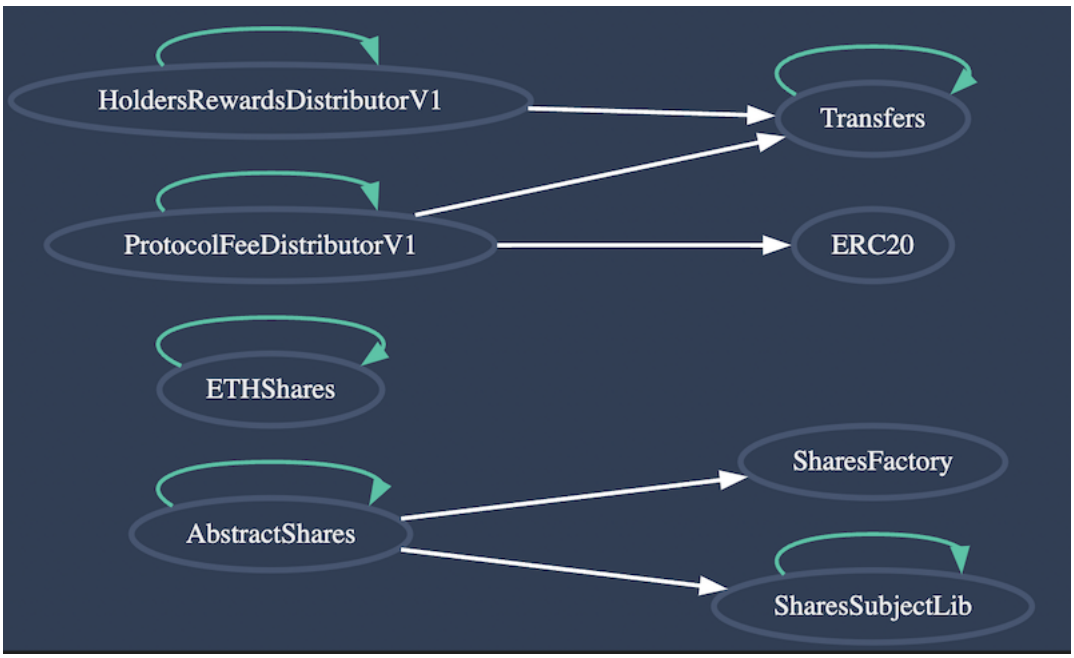

Figure 4: Contract Interactions 2

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held understanding of the auditor's knowledge of security patterns as they relate to the client's contract(s), assuming that blockchain technologies, in particular, will continue to undergo frequent and ongoing development and therefore introduce unknown technical risks and flaws. The scope of the audit presented here is limited to the issues identified in the preliminary section and discussed in more detail in subsequent sections. The audit report does not address or provide opinions on any security aspects of the Solidity compiler, the tools used in the development of the contracts or the blockchain technologies themselves, or any issues not specifically addressed in this audit report.

The audit report makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, the legal framework for the business model, or any other statements about the suitability of the contracts for a particular purpose, or their bug-free status.

To the full extent permissible by applicable law, the auditors disclaim all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. The auditors hereby disclaim, and each client or user of this audit report hereby waives, releases and holds all auditors harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.