A CONSENSYS DILIGENCE AUDIT REPORT

AragonBlack Fundraising

Date	November 2019
Lead Auditor	Martin Ortner
Co-auditors	Sergii Kravchenko

1 Summary

ConsenSys Diligence conducted a security audit on Aragon Fundraising, previously known as Apiary, an application based on the AragonOS framework to facilitate emergent organizations to conduct (bonding curve) fundraising campaigns and provides accountability throughout the lifecycle of a project.

- **Project Name:** AragonBlack Fundraising
- **Client Name:** Aragon Black & The Aragon Association
- **Client Contact:** Olivier Sarrouy (@osarrouy), Louis Giraux (@LouisGrx)
- **Lead Auditor:** Martin Ortner (@tintinweb)
- **Co-auditors:** Sergii Kravchenko
- **Date:** 25 Nov 2019
- **Commit Hash:** #5ad1332
- **Repository:** github/AragonBlack/fundraising

2 Audit Scope

This audit covered the following files:

File Name	SHA-1 Hash
apps/aragon-fundraising/contracts/AragonFundraisingController.sol	aa3a996f66d715dc7a17ca9adbfbbb9035d70a5d
apps/batched-bancor-market-maker/contracts/BatchedBancorMarketMaker.sol	ecdbb49b0d2e0c643287e8d88106e71a866de6b2
apps/presale/contracts/Presale.sol	6210d6ec917ff7d152b54599130cfcf09892e1cd
apps/tap/contracts/Tap.sol	0b97e2d2439da8b04a4465aa0726f5cd8cc9cae4
templates/multisig/contracts/FundraisingMultisigTemplate.sol	910556ecee995901e65eab96a3c96c7b38bf602c

The audit also includes the interaction with relevant parts of the AragonOS Framework (Kernel, ACL and Core DAO components used, as well as, Aragon Applications `Vault`, `TokenManager`, `Voting`, `Finance`, `Agent`).

Besides contracts that mock functionality or are solely used for testing purposes it was agreed that the following components are out of scope:

File Name	SHA-1 Hash
apps/bancor-formula/contracts/BancorFormula.sol	5f492a5d99549ae3ed72eabc1231452e1c59c94e
apps/bancor-formula/contracts/interfaces/IBancorFormula.sol	7712ea788fd71e6dd39db79933ba5c9d17e90528
apps/bancor-formula/contracts/utility/SafeMath.sol	8f4d141d30ff49230f90ca8fe1190905cbe9f218
apps/bancor-formula/contracts/utility/Utils.sol	ae9fca6a54fa1a7287e1013cd2e881456bb2f097

It was also verified that the logic of the 3rd party components were left unchanged (Reference: BancorFormula.sol, IBancorFormula.sol, SafeMath.sol, Utils.sol). The system uses BancorFormula v0.3 while the newer version BancorFormula v4 is already available.

The system implements a granular role-based permission model. While permissions can be assigned to any entity, the audit assumes the trust model established with the set of permissions described with the provided `FundraisingMultisigTemplate.sol`.

2.1 Documentation

The following documentation was available to the audit team:

- Introduction: Aragon Fundraising
- Aragon Documentation
- Inline Code Documentation and Comments

The audit team evaluated that the system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three broad categories:

1. **Security:** Identifying security related issues within the contract.
2. **Architecture:** Evaluating the system architecture through the lens of established smart contract best practices.
3. **Code quality:** A full review of the contract source code. The primary areas of focus include:
    - Correctness
    - Readability
    - Scalability
    - Code complexity
    - Quality of test coverage

3 System Overview

AragonBlack Fundraising is an Aragon Application that implements a fundraising system capable of accepting multiple `ERC20` compliant tokens or `ETH` as collateral. Individuals can participate in a fundraising campaign by contributing to a presale phase or by trading collateral for bonding token. Owning bonding token (`Shareholder Token`) gives individuals voting power in the DAO Shareholder Voting Application but DAO Shareholder can only vote on Board member proposals, they cannot create votings themselves. The bonding token market is governed by a `BancorFormula` based bonding curve market maker. Buy and sell orders have fees attached that are transferred to a `beneficiary` in the system. Bonding tokens can be sold at any time.

Raised collateral is transferred to the fundraising `Reserve` where it can later be withdrawn by the Board/Project Team. Withdrawal by Board members/the Project Team is controlled by a `Tap` contract that limits the number of tokens that can be withdrawn for each whitelisted collateral per time unit. The tap is a control mechanism and incentive for the project team to continuously delivery project milestones in return for funding. The monthly tap amount the Project Team can withdraw is controlled by the Shareholders via the monthly tap increase rate.

The fundraising application is built from five individual Aragon Applications that are described in more detail in the next sections.

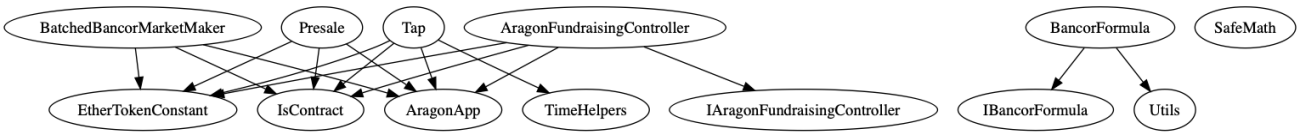
- `AragonFundraisingController`
- `Presale`
- `BatchedBancorMarketMaker` and `BancorFormula`
- `Tap`
- `Reserve`

The system requires and interacts with the following Aragon default applications: `Voting`, `Finance`, `TokenManage`, `Vault`, `Agent`.

3.1 Detailed Design

This section describes the top-level contracts, their inheritance structure, actors, permissions and contract interactions.

Inheritance Structure



Inheritance graph of the AragonBlack Fundraising Contracts (dot)

Call Graph

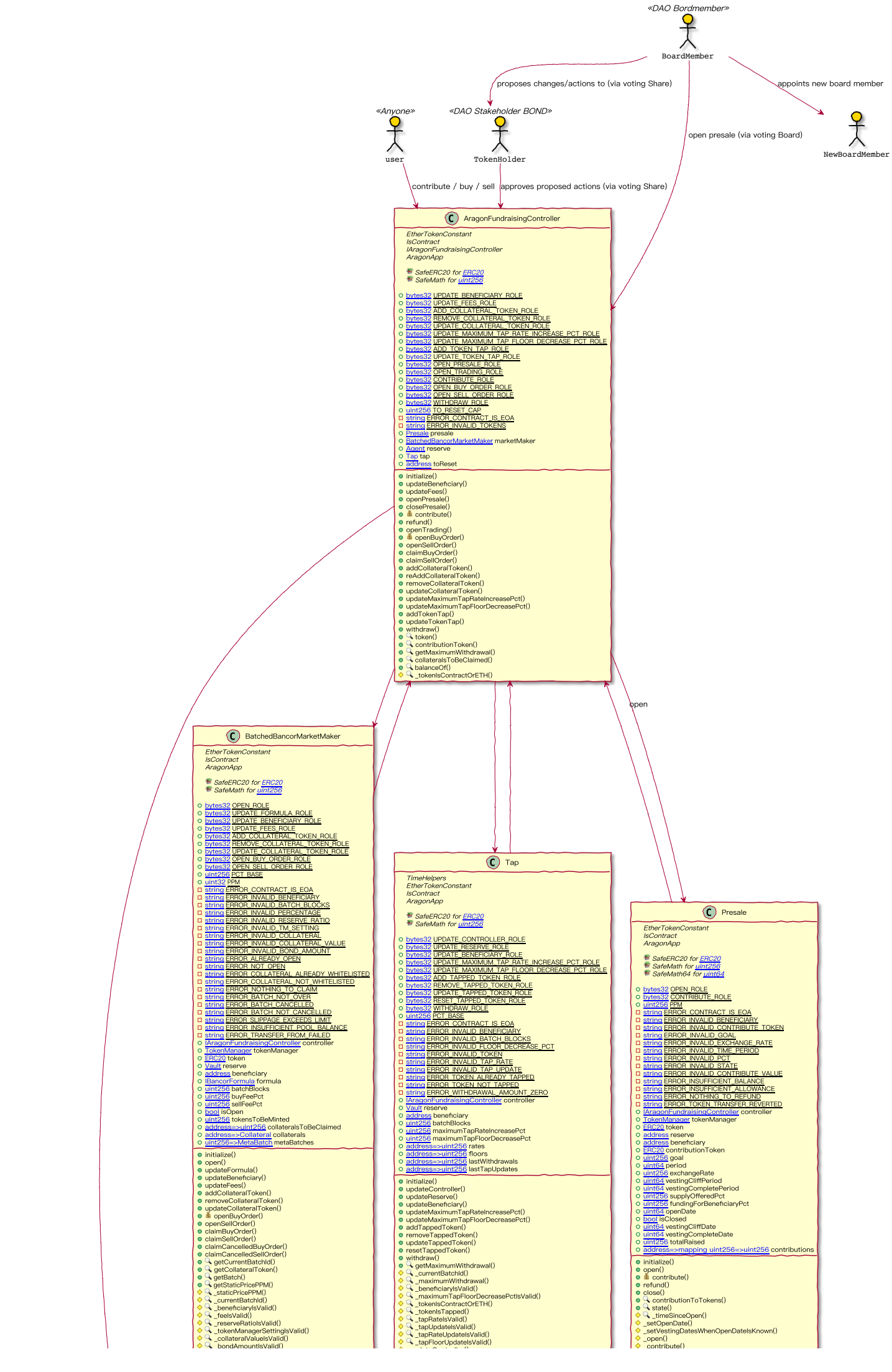
Call graph of the AragonBlack Fundraising Contracts (dot)

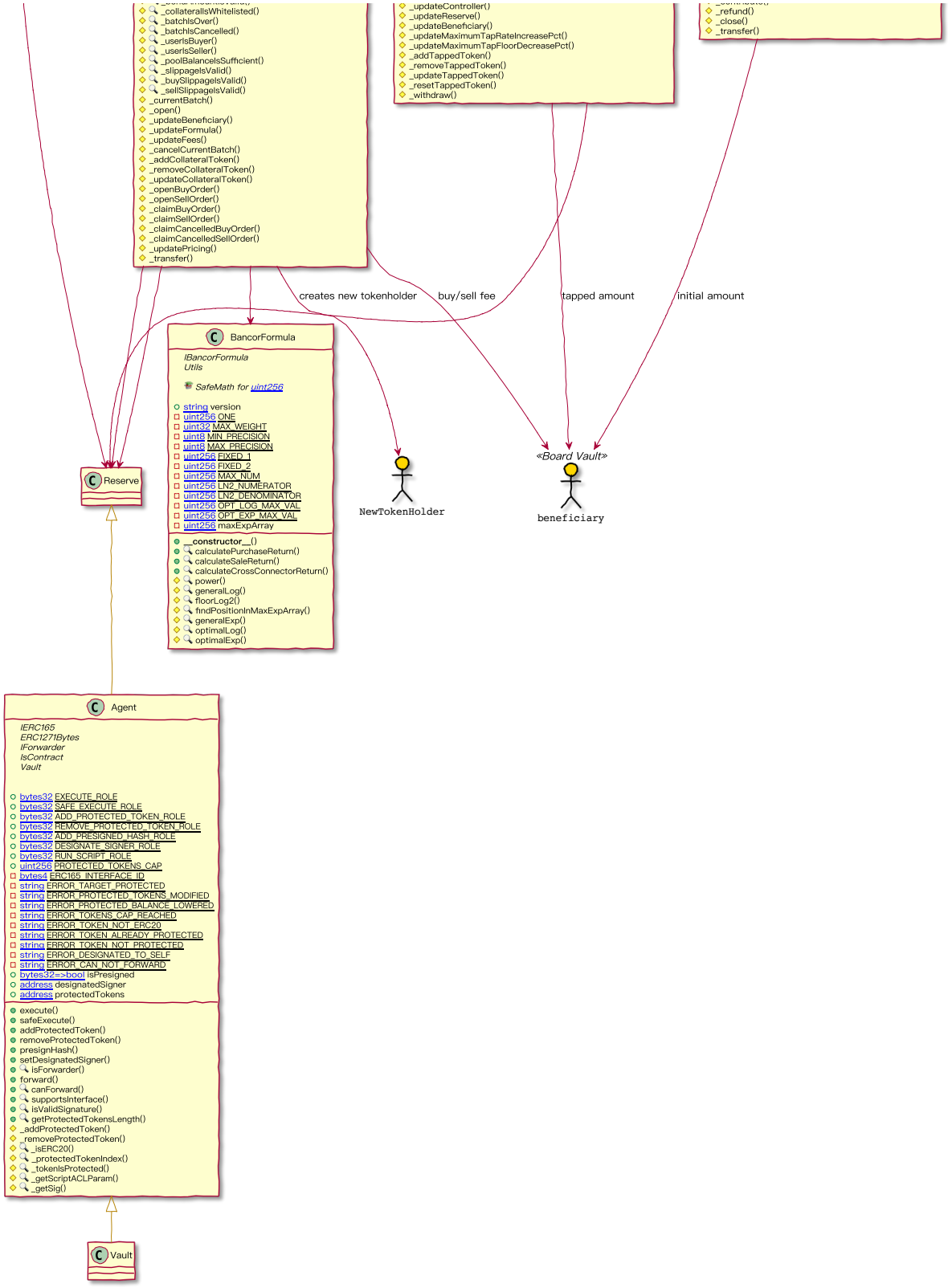
Components

The following graphic depicts the main contracts and their interfaces, actors and high-level interactions with the fundraising application. Aragon default Applications like `Vault`, `Agent`, and `TokenManager` have been left out for simplification. Actors were based on the `FundraisingMultisigTemplate` which operates in a `company-board` like DAO scenario with `Board members` (every member has the same voting power; tokens are not transferable) and `Shareholders` (voting power per account not limited; token transferable). Most of the contracts state-changing functionality is authenticated. However, contributing, buying or selling collaterals and token in the system is unrestricted. Anyone can participate in the fundraising without and there is no functionality to whitelist approved accounts. The main point of interaction is the `AragonFundraisingController` which serves as a stable API to the system. Potential investors interact with this contract to deposit collateral in return for stake in the DAO as a `Shareholder` (NewTokenHolder). All decisions are steered by the Board and Board members can reach consensus to appoint other users as new Board members. Most of the decisions and especially the critical ones (Kernel, ACL, ...) require approval by Shareholders. Shareholders cannot create votes themselves and require at least one Board member to create a vote. This also means that if the Board turns rogue, Shareholders need at least one honest Board member to propose countermeasures they can vote on. Some components specify a beneficiary that receives token from the presale, fees from token exchange in the market maker and tapped funds from the fundraising campaigns vault. The beneficiary can be the DAO Board members Vault (as specified in the `FundraisingMultisigTemplate`). Shareholders do not have access to this vault. Fundraising collaterals are stored in a Reserve, which is a default Aragon Agent

application (Agent is a Vault). Market making is done via the `BancorFormula` in `BatchedBancorMarketMaker`.

Please refer to section 5 - Security Specification for a security-centric view on the system.


«DAO Bordmember»
BoardMember
proposes changes/actions to (via voting Share)
appoints new board member
«Anyone»
«DAO Stakeholder BOND»
open presale (via voting Board)
user
TokenHolder
NewBoardMember
contribute / buy / sell
approves proposed actions (via voting Share)
AragonFundraisingController
EtherTokenConstant
IsContract
IAragonFundraisingController
AragonApp
SafeERC20 for ERC20
SafeMath for uint256
bytes32 UPDATE_BENEFICIARY_ROLE
bytes32 UPDATE_FEES_ROLE
bytes32 ADD_COLLATERAL_TOKEN_ROLE
bytes32 REMOVE_COLLATERAL_TOKEN_ROLE
bytes32 UPDATE_COLLATERAL_TOKEN_ROLE
bytes32 UPDATE_MAXIMUM_TAP_RATE_INCREASE_PCT_ROLE
bytes32 UPDATE_MAXIMUM_TAP_FLOOR_DECREASE_PCT_ROLE
bytes32 ADD_TOKEN_TAP_ROLE
bytes32 UPDATE_TOKEN_TAP_ROLE
bytes32 OPEN_PRESALE_ROLE
bytes32 OPEN_TRADING_ROLE
bytes32 CONTRIBUTE_ROLE
bytes32 OPEN_BUY_ORDER_ROLE
bytes32 OPEN_SELL_ORDER_ROLE
bytes32 WITHDRAW_ROLE
uint256 TO_RESET_CAP
string ERROR_CONTRACT_IS_EOA
string ERROR_INVALID_TOKENS
Presale presale
BatchedBancorMarketMaker marketMaker
Agent reserve
Tap tap
address toReset
initialize()
updateBeneficiary()
updateFees()
openPresale()
closePresale()
contribute()
refund()
openTrading()
openBuyOrder()
openSellOrder()
claimBuyOrder()
claimSellOrder()
addCollateralToken()
reAddCollateralToken()
removeCollateralToken()
updateCollateralToken()
updateMaximumTapRateIncreasePct()
updateMaximumTapFloorDecreasePct()
addTokenTap()
updateTokenTap()
withdraw()
token()
contributionToken()
getMaximumWithdrawal()
collateralsToBeClaimed()
balanceOf()
_tokenIsContractOrETH()
open
BatchedBancorMarketMaker
EtherTokenConstant
IsContract
AragonApp
SafeERC20 for ERC20
SafeMath for uint256
bytes32 OPEN_ROLE
bytes32 UPDATE_FORMULA_ROLE
bytes32 UPDATE_BENEFICIARY_ROLE
bytes32 UPDATE_FEES_ROLE
bytes32 ADD_COLLATERAL_TOKEN_ROLE
bytes32 REMOVE_COLLATERAL_TOKEN_ROLE
bytes32 UPDATE_COLLATERAL_TOKEN_ROLE
bytes32 OPEN_BUY_ORDER_ROLE
bytes32 OPEN_SELL_ORDER_ROLE
uint256 PCT_BASE
uint32 PPM
string ERROR_CONTRACT_IS_EOA
string ERROR_INVALID_BENEFICIARY
string ERROR_INVALID_BATCH_BLOCKS
string ERROR_INVALID_PERCENTAGE
string ERROR_INVALID_RESERVE_RATIO
string ERROR_INVALID_TM_SETTING
string ERROR_INVALID_COLLATERAL
string ERROR_INVALID_COLLATERAL_VALUE
string ERROR_INVALID_BOND_AMOUNT
string ERROR_ALREADY_OPEN
string ERROR_NOT_OPEN
string ERROR_COLLATERAL_ALREADY_WHITELISTED
string ERROR_COLLATERAL_NOT_WHITELISTED
string ERROR_NOTHING_TO_CLAIM
string ERROR_BATCH_NOT_OVER
string ERROR_BATCH_CANCELLED
string ERROR_BATCH_NOT_CANCELLED
string ERROR_SLIPPAGE_EXCEEDS_LIMIT
string ERROR_INSUFFICIENT_POOL_BALANCE
string ERROR_TRANSFER_FROM_FAILED
IAragonFundraisingController controller
TokenManager tokenManager
ERC20 token
Vault reserve
address beneficiary
IBancorFormula formula
uint256 batchBlocks
uint256 buyFeePct
uint256 sellFeePct
bool isOpen
uint256 tokensToBeMinted
address=>uint256 collateralsToBeClaimed
address=>Collateral collaterals
uint256=>MetaBatch metaBatches
initialize()
open()
updateFormula()
updateBeneficiary()
updateFees()
addCollateralToken()
removeCollateralToken()
updateCollateralToken()
openBuyOrder()
openSellOrder()
claimBuyOrder()
claimSellOrder()
claimCancelledBuyOrder()
claimCancelledSellOrder()
getCurrentBatchId()
getCollateralToken()
getBatch()
getStaticPricePPM()
_staticPricePPM()
_currentBatchId()
_beneficiaryIsValid()
_feeIsValid()
_reserveRatioIsValid()
_tokenManagerSettingIsValid()
_collateralValueIsValid()
_bondAmountIsValid()
Tap
TimeHelpers
EtherTokenConstant
IsContract
AragonApp
SafeERC20 for ERC20
SafeMath for uint256
bytes32 UPDATE_CONTROLLER_ROLE
bytes32 UPDATE_RESERVE_ROLE
bytes32 UPDATE_BENEFICIARY_ROLE
bytes32 UPDATE_MAXIMUM_TAP_RATE_INCREASE_PCT_ROLE
bytes32 UPDATE_MAXIMUM_TAP_FLOOR_DECREASE_PCT_ROLE
bytes32 ADD_TAPPED_TOKEN_ROLE
bytes32 REMOVE_TAPPED_TOKEN_ROLE
bytes32 UPDATE_TAPPED_TOKEN_ROLE
bytes32 RESET_TAPPED_TOKEN_ROLE
bytes32 WITHDRAW_ROLE
uint256 PCT_BASE
string ERROR_CONTRACT_IS_EOA
string ERROR_INVALID_BENEFICIARY
string ERROR_INVALID_BATCH_BLOCKS
string ERROR_INVALID_FLOOR_DECREASE_PCT
string ERROR_INVALID_TOKEN
string ERROR_INVALID_TAP_RATE
string ERROR_INVALID_TAP_UPDATE
string ERROR_TOKEN_ALREADY_TAPPED
string ERROR_TOKEN_NOT_TAPPED
string ERROR_WITHDRAWAL_AMOUNT_ZERO
IAragonFundraisingController controller
Vault reserve
address beneficiary
uint256 batchBlocks
uint256 maximumTapRateIncreasePct
uint256 maximumTapFloorDecreasePct
address=>uint256 rates
address=>uint256 floors
address=>uint256 lastWithdrawals
address=>uint256 lastTapUpdates
initialize()
updateController()
updateReserve()
updateBeneficiary()
updateMaximumTapRateIncreasePct()
updateMaximumTapFloorDecreasePct()
addTappedToken()
removeTappedToken()
updateTappedToken()
resetTappedToken()
withdraw()
getMaximumWithdrawal()
_currentBatchId()
_maximumWithdrawal()
_beneficiaryIsValid()
_maximumTapFloorDecreasePctIsValid()
_tokenIsContractOrETH()
_tokenIsTapped()
_tapRateIsValid()
_tapUpdateIsValid()
_tapRateUpdateIsValid()
_tapFloorUpdateIsValid()
Presale
EtherTokenConstant
IsContract
AragonApp
SafeERC20 for ERC20
SafeMath for uint256
SafeMath64 for uint64
bytes32 OPEN_ROLE
bytes32 CONTRIBUTE_ROLE
uint256 PPM
string ERROR_CONTRACT_IS_EOA
string ERROR_INVALID_BENEFICIARY
string ERROR_INVALID_CONTRIBUTE_TOKEN
string ERROR_INVALID_GOAL
string ERROR_INVALID_EXCHANGE_RATE
string ERROR_INVALID_TIME_PERIOD
string ERROR_INVALID_PCT
string ERROR_INVALID_STATE
string ERROR_INVALID_CONTRIBUTE_VALUE
string ERROR_INSUFFICIENT_BALANCE
string ERROR_INSUFFICIENT_ALLOWANCE
string ERROR_NOTHING_TO_REFUND
string ERROR_TOKEN_TRANSFER_REVERTED
IAragonFundraisingController controller
TokenManager tokenManager
ERC20 token
address reserve
address beneficiary
ERC20 contributionToken
uint256 goal
uint64 period
uint256 exchangeRate
uint64 vestingCliffPeriod
uint64 vestingCompletePeriod
uint256 supplyOfferedPct
uint256 fundingForBeneficiaryPct
uint64 openDate
bool isClosed
uint64 vestingCliffDate
uint64 vestingCompleteDate
uint256 totalRaised
address=>mapping uint256=>uint256 contributions
initialize()
open()
contribute()
refund()
close()
contributionToTokens()
state()
_timeSinceOpen()
_setOpenDate()
_setVestingDatesWhenOpenDateIsKnown()
_open()
_contribute()

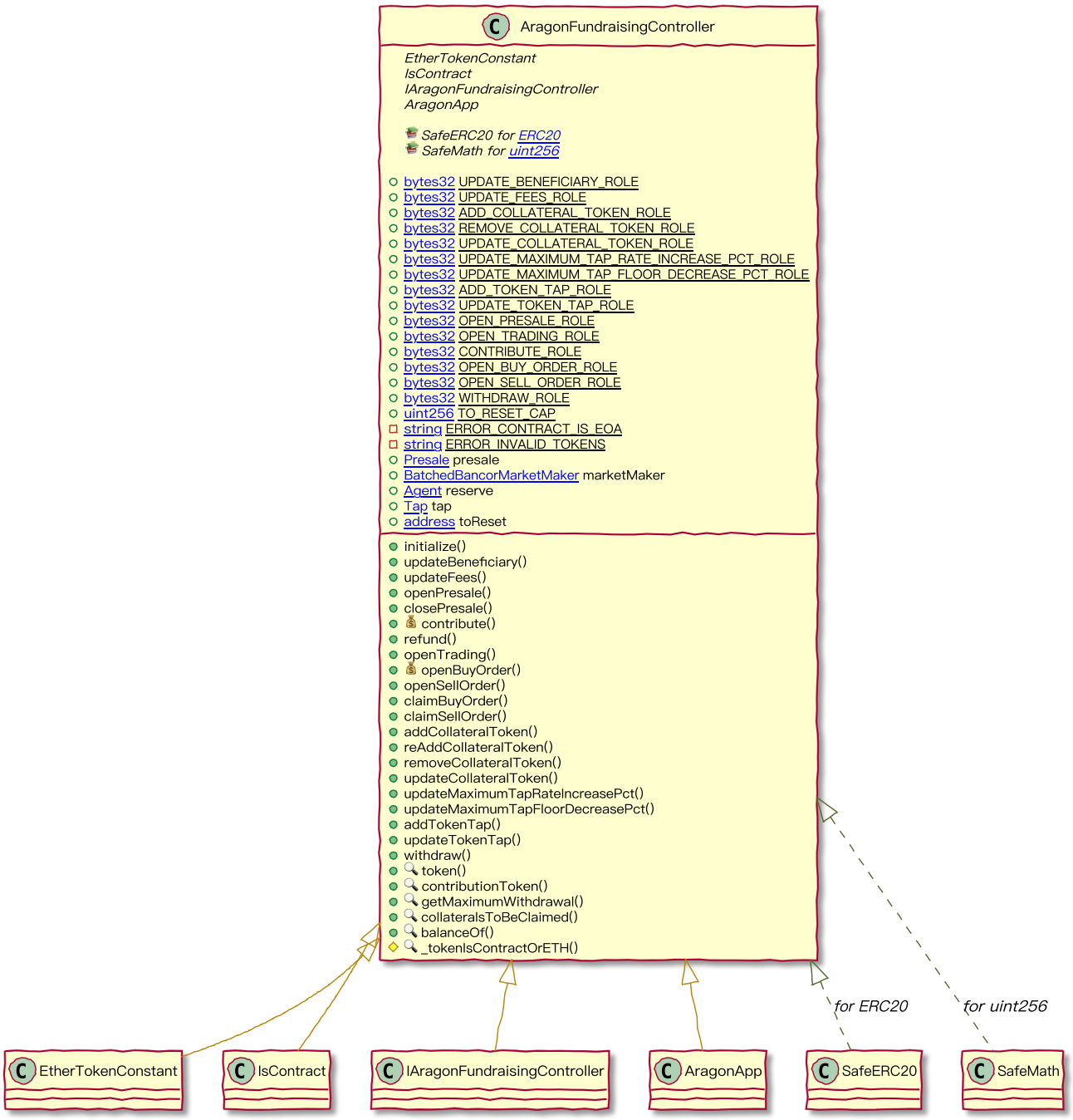


High-Level Application Overview

AragonFundraisingController

The external interface of the application and the main entry point for interaction with the system. The contract provides a stable API to the fundraising application and allows individuals to contribute to fundraising campaigns (presale, token market). It allows Board members and Stakeholder to change parameters or perform actions and interacts with the four main back-end components: `Presale`, `BatchedBancorMarketMaker`, `Tap`, `Reserve`.

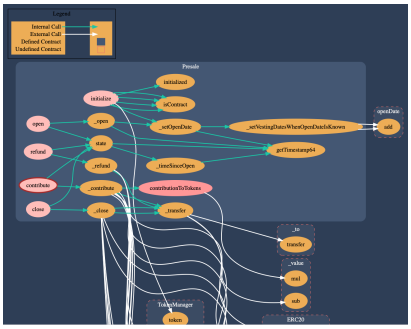
Call graph of the AragonBlack Fundraising Controller (dot)



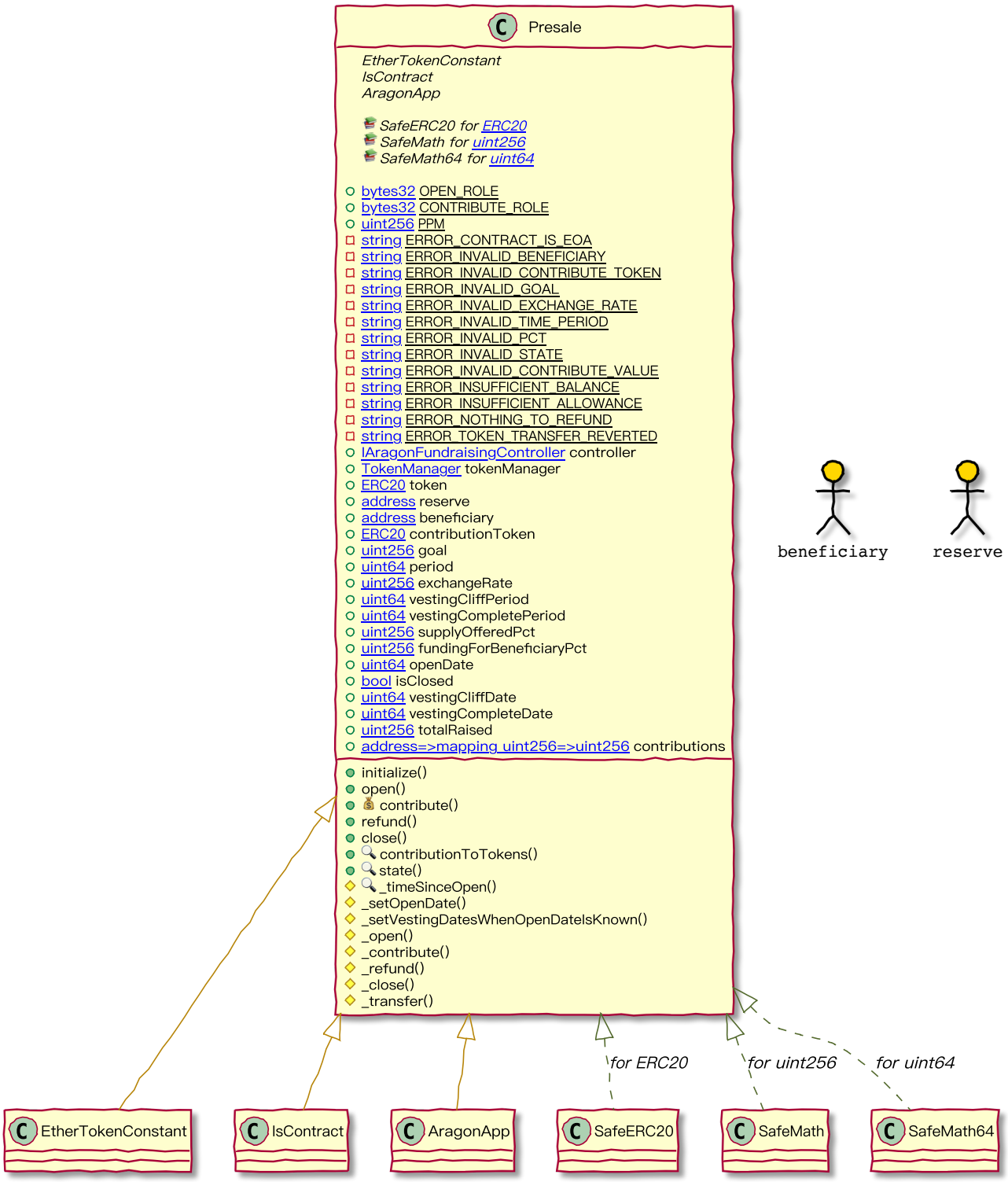
Controller

Presale

Allows the Project Team/Board to start the Fundraising Campaign with a time-limited token presale that sells token at a fixed rate with configurable token vesting for contributors. The presale can only be started once.



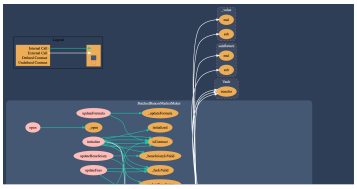
Call graph of the AragonBlack Presale Contract (dot)



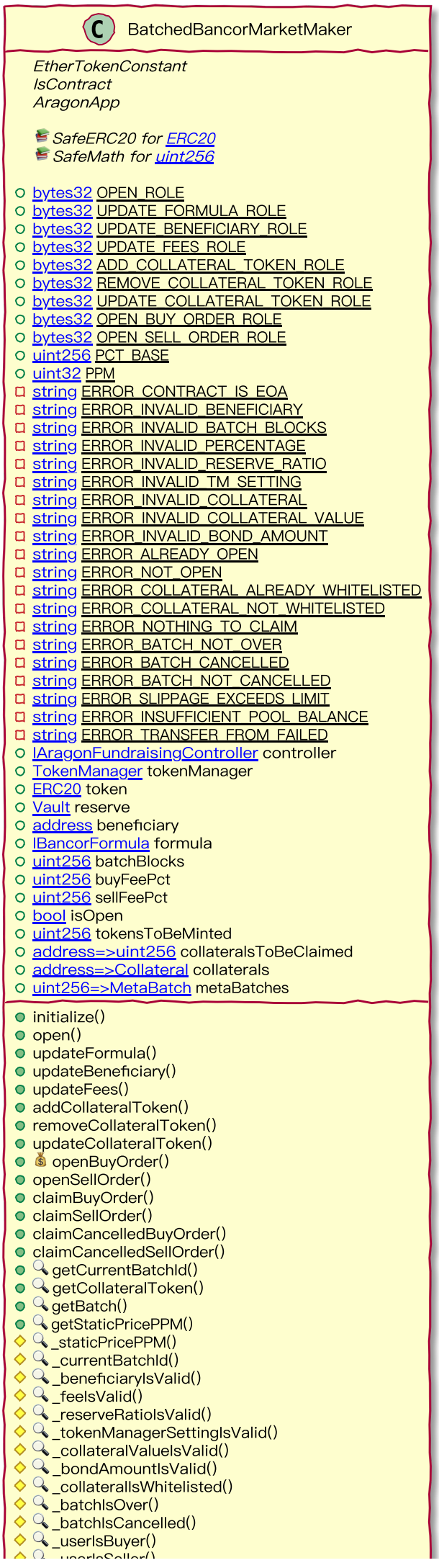
Presale

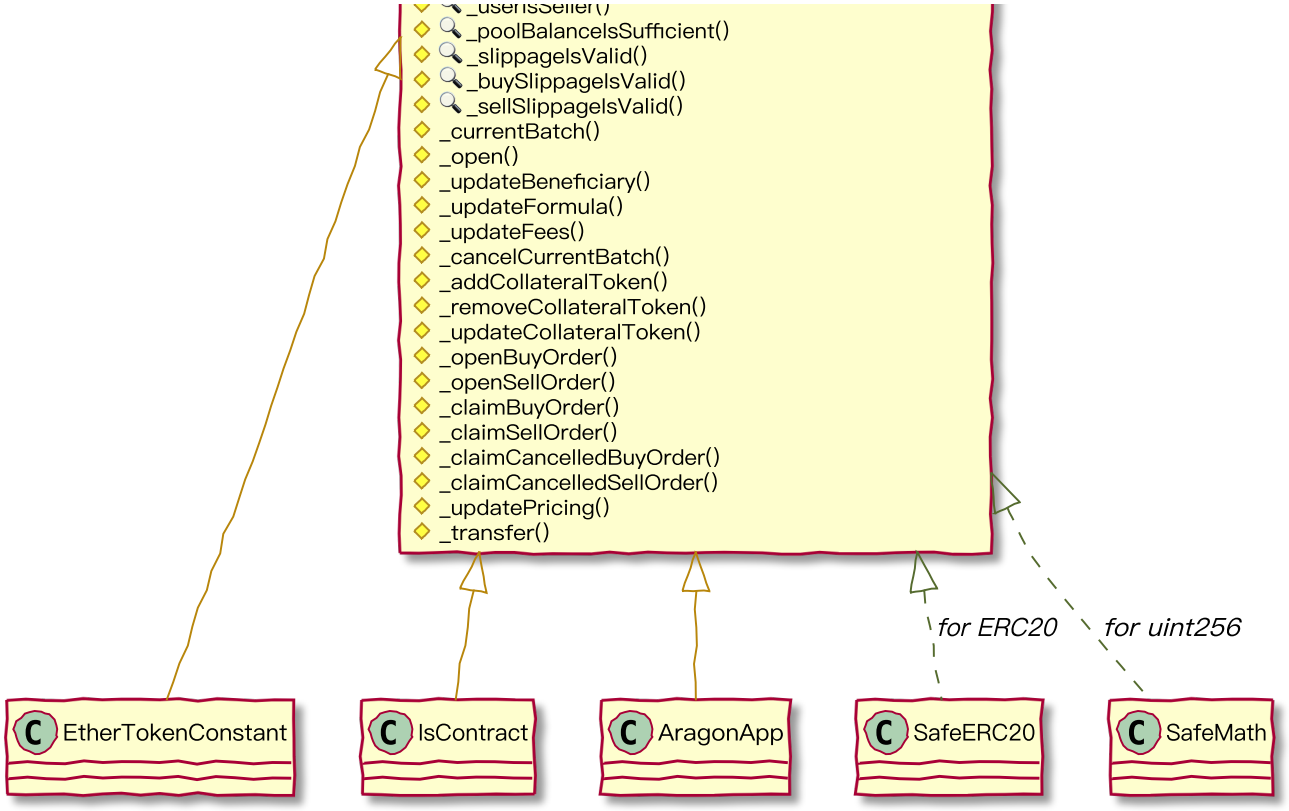
BatchedBancorMarketMaker

Implements the Shareholder token market logic based on the 3rd party `BancorFormula` and the fee economics. It is an AragonApp that is acting as the market making contract for token transfers and facilitates the buying and selling of bonding tokens investors receive for their investments. The contract interfaces with the `BancorFormula` and is critical to the system. It also collects fees and transfers the invested token or `ETH` to the pool or fee recipient.



Call graph of the AragonBlack MarketMaker Contract (dot)

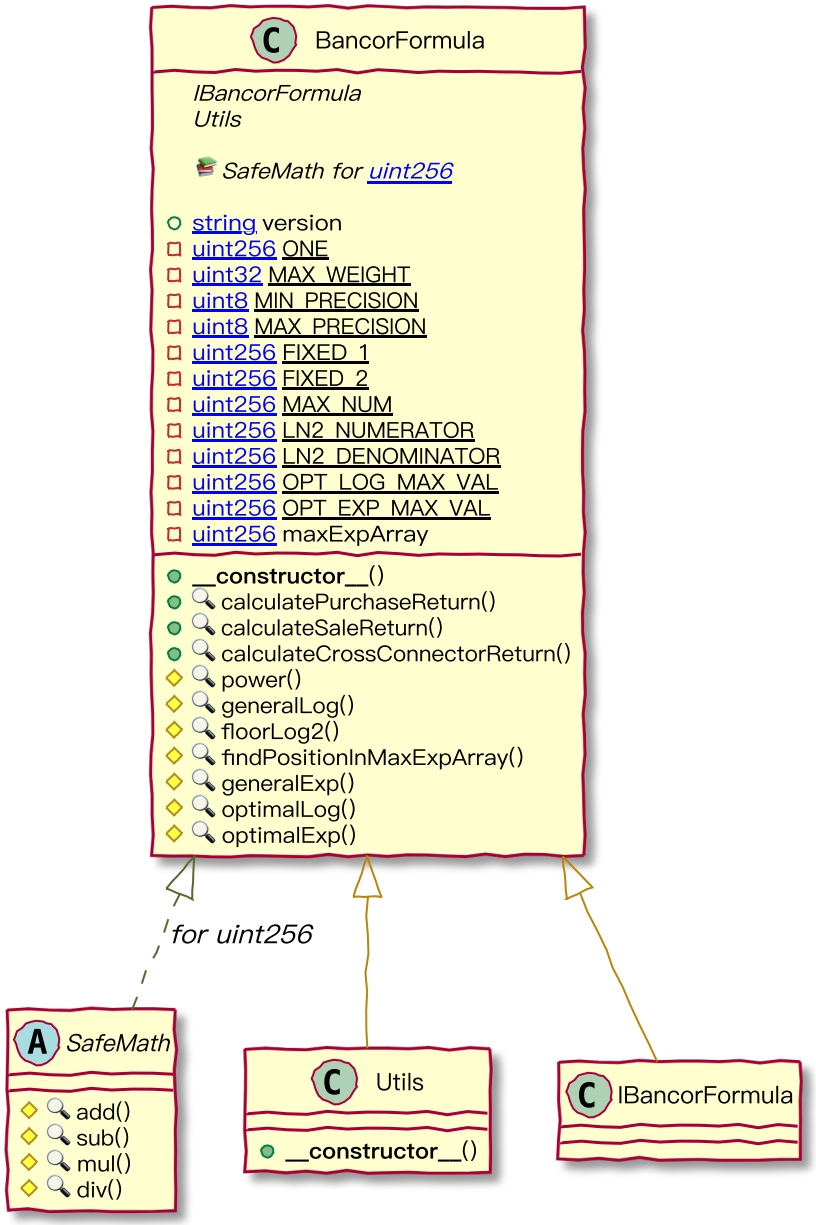

BatchedBancorMarketMaker
EtherTokenConstant
IsContract
AragonApp
SafeERC20 for ERC20
SafeMath for uint256
bytes32 OPEN_ROLE
bytes32 UPDATE_FORMULA_ROLE
bytes32 UPDATE_BENEFICIARY_ROLE
bytes32 UPDATE_FEES_ROLE
bytes32 ADD_COLLATERAL_TOKEN_ROLE
bytes32 REMOVE_COLLATERAL_TOKEN_ROLE
bytes32 UPDATE_COLLATERAL_TOKEN_ROLE
bytes32 OPEN_BUY_ORDER_ROLE
bytes32 OPEN_SELL_ORDER_ROLE
uint256 PCT_BASE
uint32 PPM
string ERROR_CONTRACT_IS_EOA
string ERROR_INVALID_BENEFICIARY
string ERROR_INVALID_BATCH_BLOCKS
string ERROR_INVALID_PERCENTAGE
string ERROR_INVALID_RESERVE_RATIO
string ERROR_INVALID_TM_SETTING
string ERROR_INVALID_COLLATERAL
string ERROR_INVALID_COLLATERAL_VALUE
string ERROR_INVALID_BOND_AMOUNT
string ERROR_ALREADY_OPEN
string ERROR_NOT_OPEN
string ERROR_COLLATERAL_ALREADY_WHITELISTED
string ERROR_COLLATERAL_NOT_WHITELISTED
string ERROR_NOTHING_TO_CLAIM
string ERROR_BATCH_NOT_OVER
string ERROR_BATCH_CANCELLED
string ERROR_BATCH_NOT_CANCELLED
string ERROR_SLIPPAGE_EXCEEDS_LIMIT
string ERROR_INSUFFICIENT_POOL_BALANCE
string ERROR_TRANSFER_FROM_FAILED
IAragonFundraisingController controller
TokenManager tokenManager
ERC20 token
Vault reserve
address beneficiary
IBancorFormula formula
uint256 batchBlocks
uint256 buyFeePct
uint256 sellFeePct
bool isOpen
uint256 tokensToBeMinted
address=>uint256 collateralsToBeClaimed
address=>Collateral collaterals
uint256=>MetaBatch metaBatches
initialize()
open()
updateFormula()
updateBeneficiary()
updateFees()
addCollateralToken()
removeCollateralToken()
updateCollateralToken()
openBuyOrder()
openSellOrder()
claimBuyOrder()
claimSellOrder()
claimCancelledBuyOrder()
claimCancelledSellOrder()
getCurrentBatchId()
getCollateralToken()
getBatch()
getStaticPricePPM()
_staticPricePPM()
_currentBatchId()
_beneficiaryIsValid()
_feeIsValid()
_reserveRatioIsValid()
_tokenManagerSettingIsValid()
_collateralValueIsValid()
_bondAmountIsValid()
_collateralIsWhitelisted()
_batchIsOver()
_batchIsCancelled()
_userIsBuyer()
beneficiary



MarketMaker

BancorFormula

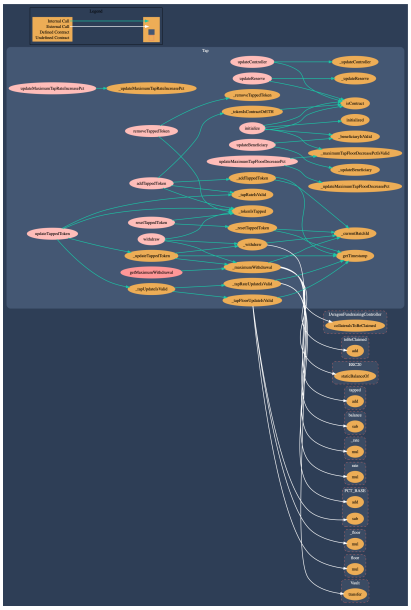
`BancorFormula.sol` is the implementation of the Bancor bonding curve formula.



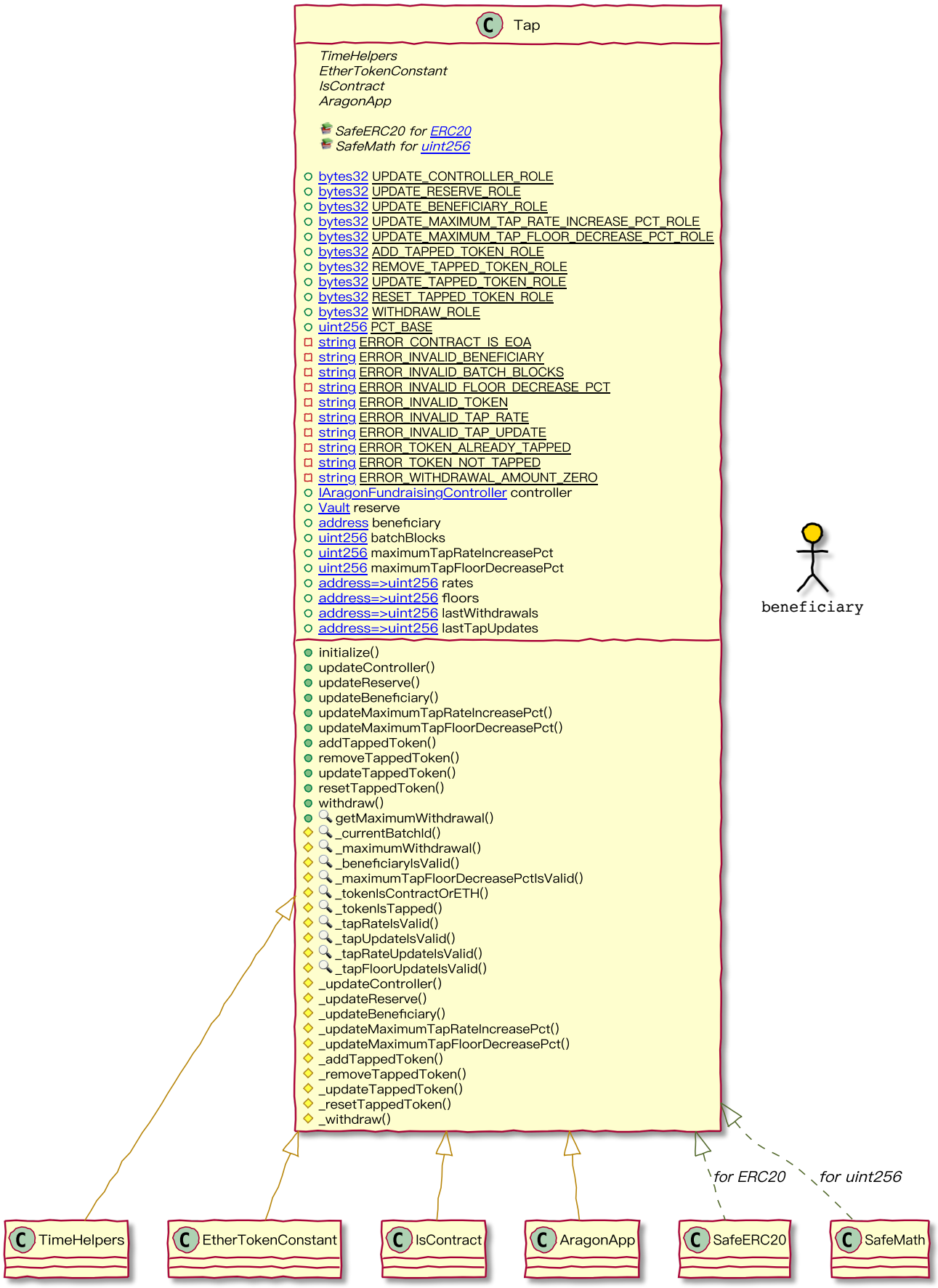
BancorFormula

Tap

Limits the amount of collaterals the Board can withdraw from the `Reserve`.



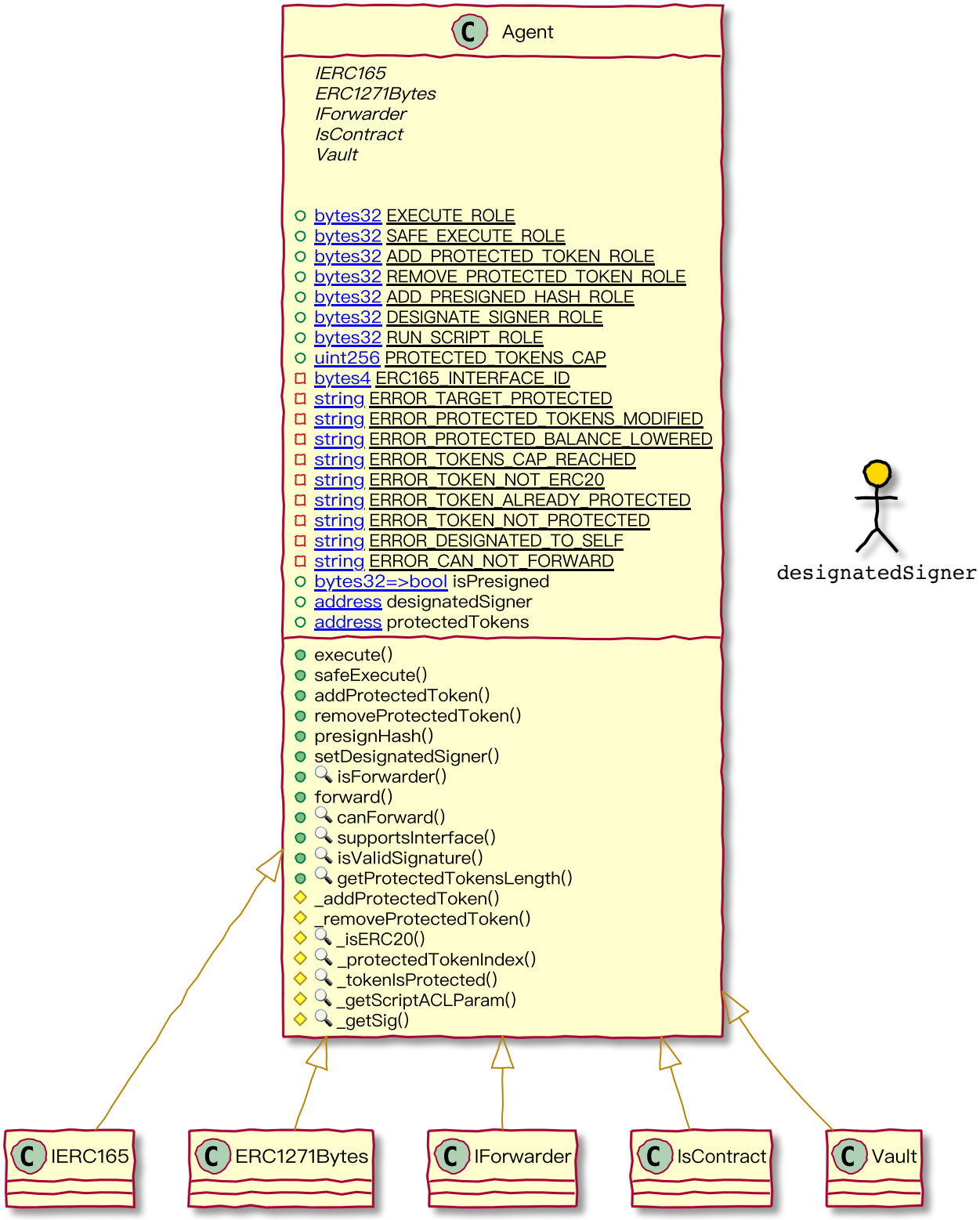
Call graph of the AragonBlack Tap Contract (dot)



Tap

Reserve (Agent)

Sometimes referred to as `Pool`, is an Aragon `Agent` application that acts as a value store for raised collateral and can be used to interact with other entities. The `Agent` application inherits the Aragon `Vault` application and was recently updated with `SAFE_EXECUTE` and the possibility to add/remove tokens that are protected by this method. `SAFE_EXECUTE` allows to call other contracts on behalf of the reserve and ensures that protected token balances cannot change during `safe_execute` calls. Neither Board nor Shareholders have direct access to the funds stored in the `Reserve`.



Pool / Agent

Actors

There are three main groups of actors in the AragonBlack Fundraising system:

- **FundraisingMultisigTemplate** - A DAO deployer can interact with the template to deploy a new fundraising DAO. The template initially has permissions to configure the DAO but transfers all ownership in the final step of the deployment. The template contract should not remain in control of the DAO after its deployment. Initial configuration of the template contract must be verified before interacting with it.
- **The DAO deployer** - A user that interacts with the template contract to deploy a new DAO. This user should not directly remain in control of any of the deployed DAO components.
- **Board members** - Also known as the fundraising Project Managers. The board controls the DAO but can only propose changes to the Stakeholders. Stakeholders have to approve the changes. The board may have its Vault that is managed by a Finance application. Funds can only be withdrawn from this Vault via Board vote. Board members can also tap into the fundraising campaigns Reserve but this access and the allowed withdrawal amount for collateral is restricted by the fundraising Tap contract.
- **Beneficiary** - Usually a Vault managed by the Board. Receives initial tokens from the presale, trading fees from the MarketMaker and funds withdrawn from the reserve.
- **Shareholders** - Are the investors in the fundraising DAO. They are a control instance that approves changes or actions proposed by Board members.
- **Any Ethereum Account** - With the current permission setup deployed with the provided DAO template anyone can participate in the token presale to receive vested tokens or buy tokens from the bonding curve based batched market maker. Buying tokens gives voting power in the DAO and turns the Account into a Shareholder. However, the permissions can also be assigned to a KYC provider that whitelists investors in the future.

Security focussed information about actors in this system can be found in section 5 - Security Specification/Actors.

3.2 FundraisingMultisigTemplate

The fundraising DAO template is based on the Company-Board default DAO-Template. We have also audited this default DAO-Template as part of the Aragon DAO-Templates

Audit, please refer to this report for general security information. A security analysis of the template is provided as part of section 5 - Security Specification.

The template is well structured and aligned with the coding style of the Aragon default DAO-Templates. Wherever possible it makes use of functionality provided by the Aragon `BaseTemplate` which is part of the Aragon default DAO-Templates repository. New fundraising enabled DAOs can be deployed in four steps:

1. `prepareInstance`

- creates a new DAO
- creates the BOARD token
- installs Board apps (TokenManager, Voting, Vault (beneficiary), Finance)
- mints one BOARD token per member for the initial group of board members
- cache the DAO and apps for the multi-step deployment approach

2. `installShareApps`

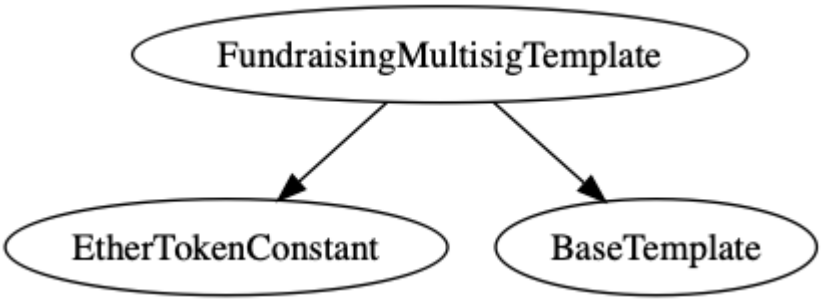
- creates the SHARE token
- installs Share apps (TokenManager, Voting)
- sets up board permissions (TokenManager, Voting, Vault, Finance)
- caches newly installed apps
- `installFundraisingApps`
  - installs fundraising applications (Presale, MarketMaker, Tap, Controller)
  - sets up share permissions (TokenManager, Voting)
  - sets up fundraising permissions (Reserve, Presale, MarketMaker, Tap, Controller)
- `finalizeInstance`
  - sets up initial collaterals (`DAI` as protected and tapped token, `ANT` as protected collateral but not as a tapped token, `ETH` is not whitelisted by default)
  - sets up EvmScriptRegistryPermission
  - transfers root permissions from template to Voting_Share
  - registers application id
  - clears the cache

The `DAOFactory`, `ENS Registry`, `MiniMeTokenFactory`, `aragonID`, and the `DAI` and `ANT` token contract addresses are specified when deploying the template contract. Users should make sure the initial configuration of the template is safe (no malicious factory or

collateral token contracts) when using a 3rd party template contract to deploy a new DAO.

A visual representation of the permission setup deployed with the DAO can be found in section 5 - Security Specification.

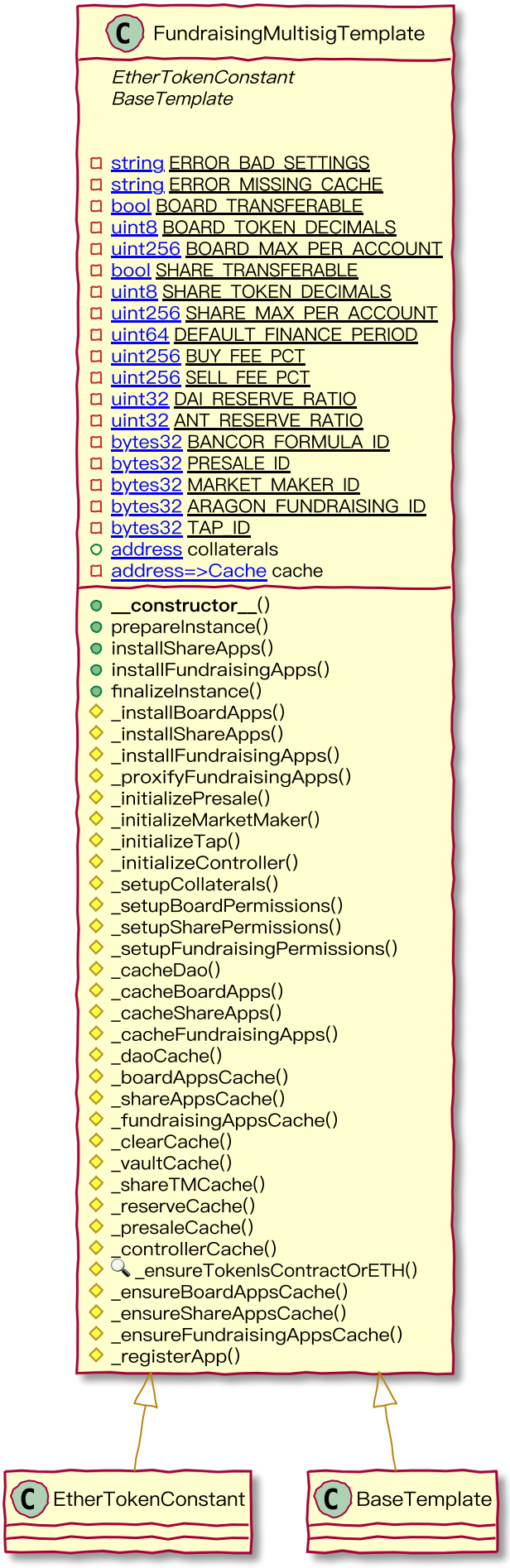
Inheritance Structure



Inheritance graph of the AragonBlack Fundraising Contracts (dot)

Call Graph

Call graph of the AragonBlack Fundraising Contracts (dot)



FundraisingMultisigTemplate

4 Key Observations/Recommendations

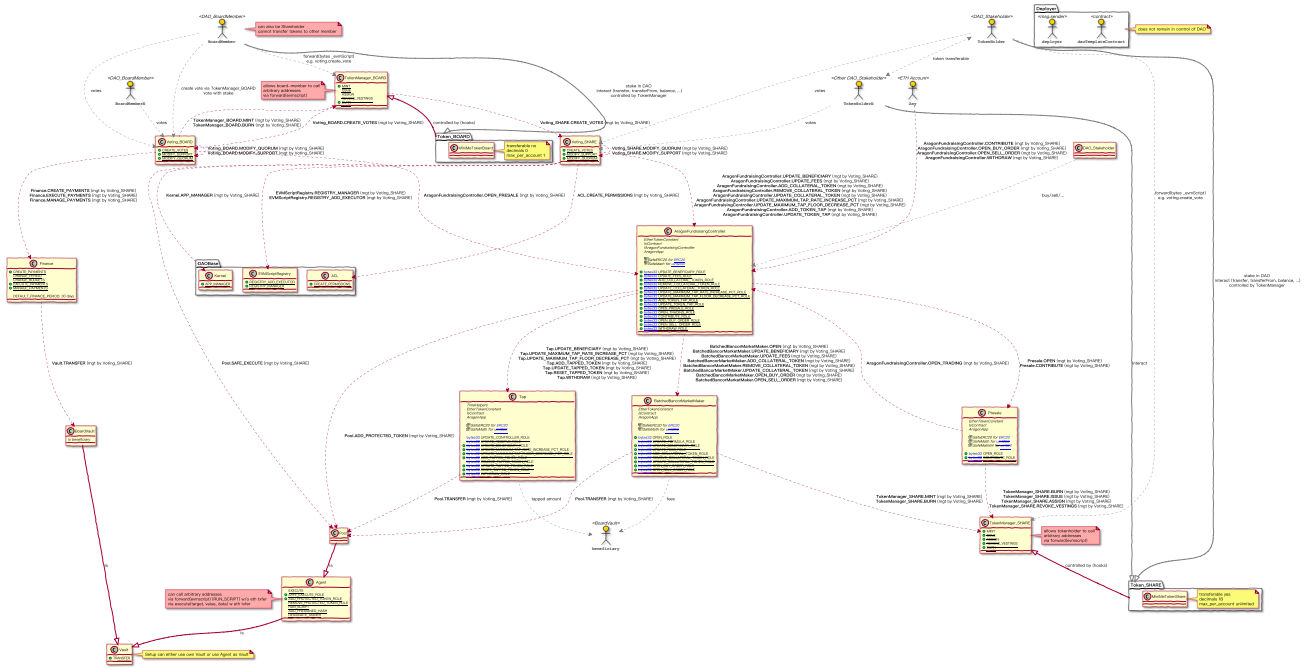
- The Aragon Applications and the Template are well structured, following Aragon design specifications and coding style. This greatly improves code readability, maintainability, security and is effectively reducing effort to review the application and template. The template relies on functionality to set up a new DAO provided with the DAO-Templates repository instead of inventing new mechanisms. Both the template and the application code itself are good examples of a coding style that supports maintainability and security.
- The project team provided system documentation and auditable specification documents. It is typically suggested to make documentation available that clearly outlines at least the following information:
    - Application Name, Version and Outline - is available
    - Roles & Capabilities ideally grouped into logical actors (e.g. Investors, Project Managers, ...) - is available
    - Set-Up and initialization details - is available
    - Caveats & Limitations - can be improved
    - Security Considerations, Common Pitfalls or Secure Setup information - can be improved
    - A description of an example Application Lifecycle - can be improved
    - A reference to an example kit to deploy the Aragon Black Fundraising application with a DAO - template is available
- Scripts to bootstrap the repository and compile & run test suites are provided with the project.
- Security best practices are followed where they make sense.
- Common design patterns are used where required instead of inventing new interfaces and designs.
- 3rd party modules (`BancorFormula` and its components) have not been logically modified.
- Arithmetic operations are performed using SafeMath in most cases where it makes sense.
- Staticcalls are enforced for `ERC20.balance` calls.
- Smart contracts are following the Unix design philosophy as recommended by Aragon. Contracts are split into logical building-blocks.

- Inline documentation for contracts and exposed methods are available and outline the functionality and expected input parameters.
- The application relies on AragonOS framework functionality and is following the Aragon application development guide.
- Exposed functionality is protected with the AragonOS framework ACL system creating a fine-grained role system that can be customized by DAO's.
- Critical functionality of MarketMaker and Presale is protected from reentrancy.
- The application is set to compile with an outdated solidity version lacking the latest security improvements. The solidity version cannot easily be changed as it is dictated by the underlying AragonOS framework.
- The `AragonFundraisingController` 's purpose is to forward and keep in sync other contracts. This forces it to be in sync with all of the other contracts it interacts with, effectively writing a lot of boilerplate to serve as the interface for the whole application. This is a problem because if some functionality is added or removed in any of the market maker, reserve or tap contracts, the controller needs to be updated. Another different way is to remove the controller altogether and have the web interface call the right contract for each functionality. For the methods that keep the collateral tokens (`addCollateralToken`) and the beneficiary (`updateBeneficiary`) in sync, a different contract can be used as a ledger having the collateral token list and the beneficiary available (or even split in 2 contracts: one ledger for collateral tokens and the other for the beneficiary); each of the contracts (market maker, reserve and tap) will be able to retrieve the list of collateral tokens or the beneficiary. Managing the collateral tokens or the beneficiary will be done directly in these contracts.
- The authentication system could be changed from `auth()` to `authP()` allowing to set rules on the method parameters. Please note that this might increase complexity and must be complemented with unit-tests accordingly.
- The price of the Shareholder tokens gets lower with every second because of the Tap withdrawal allowance increasing with the same rate, which decreases the pool balance that is used for the Share token price calculation.
- The project does not yield any security-relevant compiler warnings in scope for this audit.
- The test-suite passes without failing test-cases.

- Test coverage is not complete and failed to generate coverage statistics for the Presale contract. The GitHub/AragonBlack: CI Integration does not collect coverage statistics for Presale. Any contract system that is used on the Mainnet should have as a minimum requirement a 100% test coverage. In particular, it's useful to include negative test cases ensuring that undesirable changes are detected early and kept in-line with the specification. Security-focused test cases verifying that permissions are set-up according to the security-specification and trust-model should be consistently implemented. Methods that require special permissions must be tightly covered. None of the contracts that comprise the contract system should be exempt from testing/coverage. For details see section Test Coverage Measurement.

5 Security Specification

This section describes the behavior of the system under audit from a security perspective. It is best combined with the overview given in section 3 - Components. Please note that this document is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team. Furthermore, the information contained in this section can be used for internal security activities and we recommend documenting and building-upon the trust model that has been established.



Template Permission Overview

5.1 Actors

The relevant actors are as follows:

- **FundraisingMultisigTemplate**
- **The DAO deployer**
- **Board members**
- **Beneficiary**
- **Shareholders**
- **Any Ethereum Account**

5.2 Components

- **Voting Board** - Any Board member can create votings on the Shareholder voting application. Board consensus but no Shareholder approval is required to open the presale or spend funds from the Board Vault.
- **Voting Shareholder** - Critical permissions are assigned to the Shareholder voting application. An entity that controls the Shareholder voting application (e.g. whales with enough stake to control the vote quorum/support) and colludes with one Board member may be able to configure critical parameters of the fundraising application (whitelisted tokens, tap amount, tap increase rate, granting and revoking permissions, managing applications for the DAO). The Shareholders cannot initiate a votings themselves.
- **DAO Kernel** - Board members rely on Shareholder approval to manage the DAO applications and permissions.
- **FundraisingController** - Main point of interaction that owns critical roles with the fundraising applications components.
- **Reserve** - Fundraising value store (`ETH` and whitelisted collateral tokens).
- **Tap** - Controls funds stored in **Reserve**.
- **Presale** - Initial value store for contributions in the presale phase. Value gets transferred to beneficiary/reserve when presale goal is reached and presale is closed. Mints Shareholder token.
- **MarketMaker and BancorFormula** - Define exchange rates, prices, and fees. Mints Shareholder token.
- **Beneficiary** - Board/Project Team value store. Usually, an Aragon Vault application but can be any account.

5.3 Trust Model

The trust model and security observations aim to bring transparency about security-relevant characteristics of the system, help to understand trust assumptions and

describe potential high-level threats to the system. The goal is to spark security discussions, document them as part of a continuous process and use them as input for internal SDL security practices.

It is based on the permission setup provided with the `FundraisingMultisigTemplate`. The audit team would like to note that the system can be deployed with various configurations. Other templates (DAO scenarios) than the one audited as part of this work might not enforce secure defaults or a safe permission setup.

Deployment

Before the Fundraising DAO can be used it has to be deployed using a template contract. This template contract can be provided by Aragon or third parties. We would like to emphasize that both the template contract code and its initial configuration as well as its dependencies (especially Factory Contracts) must be verified and should be audited. The template contract nor factory contracts or a third party should remain in control of any of the newly deployed DAOs components.

- The **FundraisingMultisigTemplate** can be deployed by anyone. The template deployer does not remain in direct control of the template but it can be indirectly controlled via the templates default configuration and factories being used (e.g. DAOFactory, MiniMeFactory, TokenContracts).

- The **DAO deployer** is an account that interacts with the **FundraisingMultisigTemplate** to deploy a new DAO. It is initiating the four DAO deployment steps outlined in section 3 - System Overview.

- In the course of the deployment of a DAO, permissions are assigned to the **FundraisingMultisigTemplate**. For example, `_createDAO` initially assigns `Kernell.APP_MANAGER_ROLE` and `Acl.CREATE_PERMISSIONS_ROLE` to the template. When minting tokens `BaseTemplate._mintTokens()` temporarily assigns the `TokenManager.MINT_ROLE` to itself and removes the permission after the tokens have been minted. The **FundraisingMultisigTemplate** temporarily assigns `Controller.ADD_COLLATERAL_TOKEN_ROLE` to itself and transfers this permission to Voting_Share after whitelisting `DAI` and `ANT` as collateral. When finalizing the new DAO the **FundraisingMultisigTemplate** transfers `Kernell.APP_MANAGER_ROLE` and `Acl.CREATE_PERMISSIONS_ROLE` to Voting_Share effectively revoking its access from the newly deployed DAO.

- The **DAO deployer** is not granted any permissions during the deployment of the new DAO but it is in control of configuration options and the initial set of Board

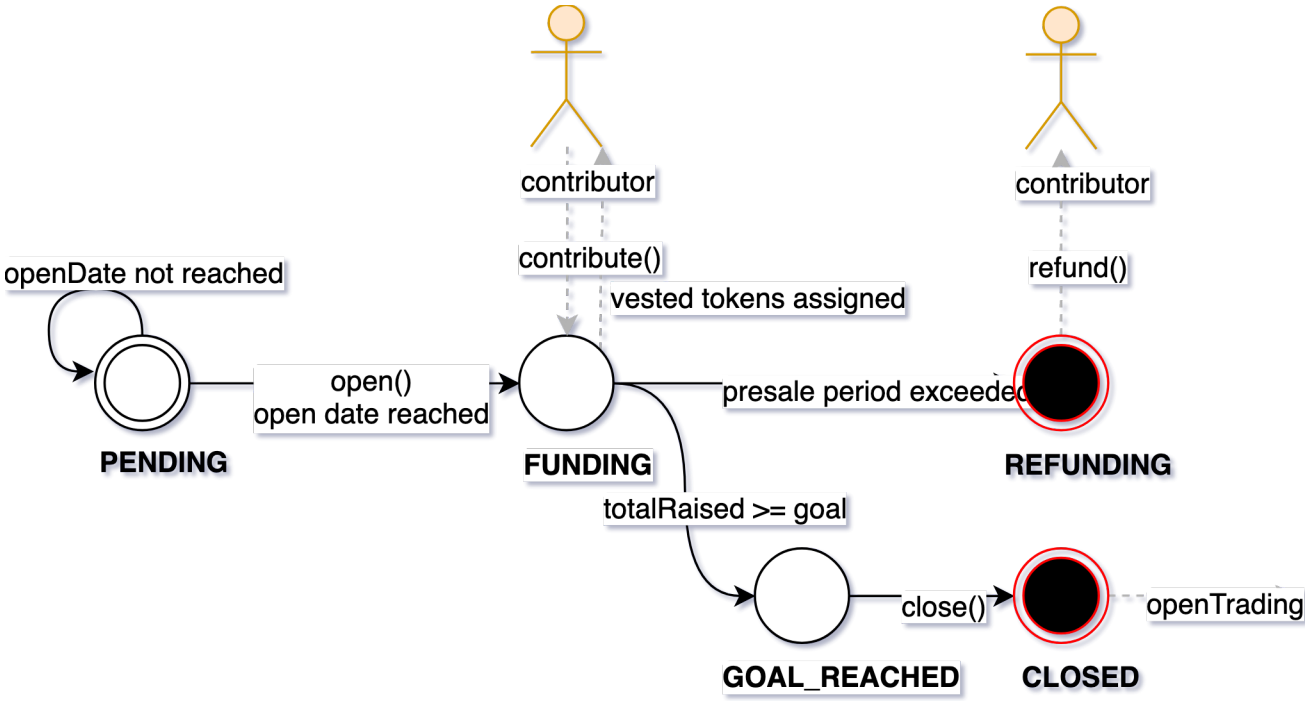
members. DAO users must verify the configuration settings of the newly deployed DAO before participating in it.

- An initial group of **Board members** is assigned to the DAO by the **DAO deployer**.
- The Board Vault is initially set as the **Beneficiary** for fees, presale tokens and tap withdrawal by the **FundraisingMultisigTemplate**.

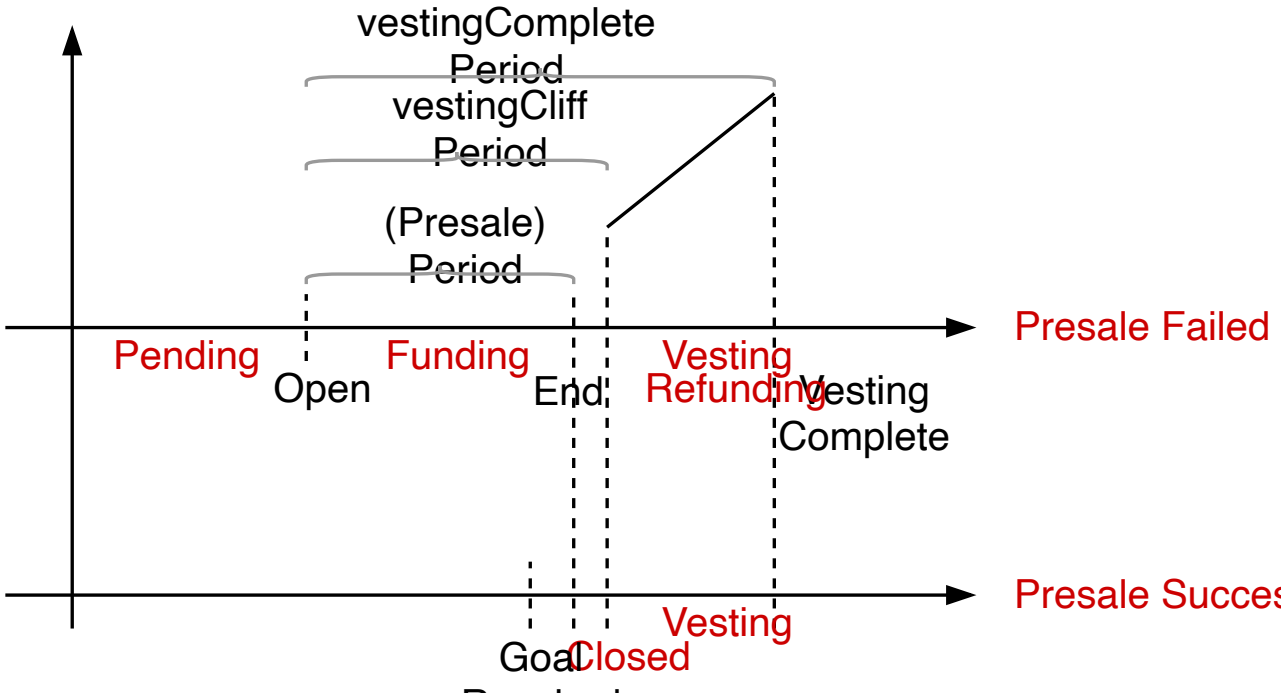
Presale

The fundraising campaign is preceded by a presale phase. With the scenario deployed by the **FundraisingMultisigTemplate** this step is mandatory and reaching the presale goal ultimately enables investors to buy or sell tokens from the MarketMaker contract. If the presale fails the fundraising DAO can only be abandoned, contributors are requested to withdraw their contributions directly from the presale contract. The presale contract is a value store and keeps funds until the goal is reached. Reaching the goal splits contributions to an amount initially assigned to the beneficiary with the rest being transferred to the fundraising reserve.

The following two images depict the presale stages and the timing configuration including the token vesting.



Presale Stages



Presale Timing

The presale proceeds in the following stages:

- `PENDING` - The presale is not yet open. Contributions are not yet accepted.
- `FUNDING` - The presale is open. Contributions are accepted.
- `REFUNDING` - The presale duration ended but the goal has not been reached. Contributors can claim a refund.
- `GOAL REACHED` - The presale duration ended and the goal has been reached. Waiting for someone to close the presale.
- `CLOSED` - The presale has reached its goal and it has been closed by someone, transferring a number of tokens to the beneficiary (Board Vault) and the rest to the Fundraising reserve. Shareholder tokens are minted and assigned vested to contributors. Trading with the MarketMaker is finally opened.

The following properties have been identified:

- The initial presale configuration is critical, set by the **DAO deployer** and must be verified by participants before contributing to a presale. For example, the **DAO deployer** may initially configure the presale to transfer 100% of contributed token to the **Beneficiary** account instead of providing it as collateral to the reserve. This will give the Board direct control over the funds instead of withdrawals being restricted by the `Tap` contract.

- The presale phase can be set to open at a specific date or when opening it manually. Reaching the presale goal opens trading with the Fundraising MarketMaker.

- When the presale is in `GOAL_REACHED` state it can be closed by anyone making sure funds cannot be stuck waiting for an administrator to close it.
- Anyone can contribute to the presale via `Controller` (`CONTRIBUTE_ROLE`).
- Refunds must be claimed directly from the presale contract.
- Shareholder tokens are minted when processing the contribution. Tokens are not immediately available to contributors but vesting based on `vestingCliffPeriod` and `vestingCompletePeriod` configured for the contract. `vestingCliffPeriod` can only start after the presale period ends. Since the presale goal can be reached before the end of the presale period tokens will not start vesting immediately after the presale succeeded.
- Depending on the initial configuration of the presale contract, Shareholder tokens may start vesting right after the presale period, even though the presale failed, giving contributors stake in the DAO.
- Investors should claim a refund and abandon the DAO in case the presale fails. Trading with the MarketMaker opens only of it he presale succeeds. Tokens should not vest to an extent that would allow majority voting power in the Shareholder application before investors had enough time to refund their tokens (Board and Shareholder might vote on presale contract update).
- Contract upgraded must be performed via the DAO/ `AragonApp` update mechanisms. Upgrades must be proposed by `BOARD` and approved by `SHARE`, therefore, the presale contract cannot be upgraded through the DAO/ `AragonApp` upgrade mechanisms as this requires Shareholder approval and Shareholder tokens are not vested yet.
- Settings cannot be updated after initialization, **Beneficiary** cannot be updated.
- Unauthenticated functionality is protected by a reentrancy guard.
- There is an incentive to front-run transactions or act based on the fact that a presale appears to be successful soon (e.g. 40% goal is achieved - a Board member then decides to buy majority stake to gain control of the fundraising DAO)
- Board members can participate in the presale.
- Depending on the amount of Shareholder token available, single individuals might become a majority Stakeholder at a fixed rate from the presale as there is no limit

for individual buyers. Investors might want to consider that before or even after they invest someone might be able to buy close to all available tokens at minimal risk of losing funds.

Fundraising

Following the successful presale phase, the actual BondingCurveMarketMaker based fundraising starts. It allows new investors to buy Shareholder token and therefore stake in the DAO for whitelisted collateral. The price is regulated by a BancorFormula. Buy or sell orders must be placed and claimed after a batch of orders has been executed. Shareholder token can be sold for collateral at any point in time. They are transferable and there is no limit on how many tokens one account can hold.

- **Any Ethereum Account** can interact with the `AragonFundraisingController` to invest in the fundraising campaign (`CREATE_BUY_ORDER` , `CREATE_SELL_ORDER`). *Investors* get Shareholder tokens in return that give them a stake in the Voting application for **Shareholders**.
- *Investors* turn into **Shareholders**. They can use their stake of Shareholder token to participate in Shareholder Votes.
- **Shareholders** can transfer their Shareholder token. The amount of tokens per Shareholder is not limited.
- **Shareholder** tokens are minted/burned and assigned/revoked from `Presale` (contribute/refund) and the `BatchedBancorMarkedMaker` (buy/sell).
- **Shareholders** can not create votes themselves. They rely on at least one Board member to create a vote on the Shareholder voting application.
- The **Shareholder** voting application owns the management role of all permissions in the system. This means that any permission change requires a Board member to propose this change to **Shareholders** which in turn have to approve the change by casting their vote. The permission manager for a specific role can revoke or grant permissions to entities in the system.
- The **Shareholder** voting application owns critical DAO kernel roles like `APP_MANAGER_ROLE` or `CREATE_PERMISSION_ROLE` . This means that any assignment of new permissions or DAO application management (e.g. contract upgrades) requires a Board member to propose this change to **Shareholders** which in turn have to approve the change by casting their vote.

- Quorum and Support settings for the **Board** and **Shareholder** voting applications are critical to the security of the system.
- A **Board member** can also be a **Shareholder** in the DAO. Members of both user groups might also collude to create votings in the **Shareholder** voting application that are subsequently passed by majority vote.
- The **FundraisingMultisigTemplate** specifies the initial list of whitelisted collateral (`ANT` , `DAI`). Additional collateral can be added by **Board members** creating a vote on the **Shareholder** voting application to approve the addition of a collateral token.
- Actions or changes to the system proposed by **Board members** might be blocked by passive or inactive **Shareholders**.
- Adjusting quorum and support to address failing **Shareholder** participation in votings might not be possible when the majority of **Shareholders** are passive or inactive.
- **Shareholders** can execute EVMScript (depending on the executor e.g. call out to other contracts w/o value transfer) on behalf of their *TokenManager*.
- **Board members** require **Shareholder** approval to update the tap amount for specific collateral.
- **Board members** can propose to call other contracts via `Agents.safeExecute()` pending **Shareholder** approval.
- Every **Board member** can create a new voting on the board voting application.
- **Board members** can vote on changing their voting applications quorum and support.
- **Board members** can vote on adding and removing new Board members.
- **Board members** are bound to a membership token. Every Board member has the same voting power.
- **Board members** can vote on spending funds from the board vault (beneficiary in the system) via the board finance app.
- **Board members** can vote on opening the presale. This can even be done before the actual open date that was specified when deploying the DAO was hit.

- **Any Ethereum Account** can initiate the withdrawal of an amount limited by the `Tap` contract from the reserve to the beneficiary. In most cases, this will be an account acting in control or favor of **Board members**.
- **Board members** can propose an update to fees, the beneficiary account, collateral tokens, tap rate, the token tap to **Shareholders** for approval.
- Collaterals stored in and taken from the applications reserve are not recorded in a finance application. Only `Tap` can directly interact with the reserves Vault functionality.
- **Board members** can execute EVMScript (depending on the executor e.g. call out to other contracts w/o value transfer) on behalf of their *TokenManager*.
- **Board members** can withdraw collateral tokens from the Board vault and invest in getting more stake in the **Shareholder** voting application to change system settings in their favor.

General

- Compromise of individual accounts (Beneficiary, Board members, major Stakeholders) or contracts can put the system at risk.
- The early stages of fundraising may be susceptible to manipulations as it is easier to get majority voting rights.
- Setup and permission of the fundraising application with the DAO is critical. Misconfiguration can easily expose critical functionality to potential attackers.
- The Beneficiary can potentially block presale funds or funds from sell orders if it reverts when receiving `ETH` or tokens (e.g. fees). This is possible if the collateral is either `ETH` or an `ERC-20` compatible token with callbacks (e.g. `ERC-777` or `ERC-223`).
- MarketMaking - is susceptible to price manipulations (Pump & Dump).
- A general issue with creating votings that perform actions if they pass is that the executed actions need to be reviewed by all the voters to make sure that e.g. a vote asking to execute a benign action actually executes that benign action and not a critical one instead (requires means to review evmscript provided with the vote action).
- Implications of using a democracy based system on the fundraising application should be researched further. E.g. the majority can decide critical aspects of the

system: steal collaterals and Board tokens; destroy the system; blocked decisions; parties may abstain from voting or generally not interested in voting.

- Using multiple bonding curves for a single token can be problematic. Currently, the Shareholder token is issued using multiple different bonding curves (one per each collateral token). It is yet to be verified whether this is safe and working correctly. The first assumption is that if someone buys Shareholder tokens for some collateral token (say, DAI), price in DAI will go up. It looks like price in any other collateral (e.g. ETH) will go down (because in other tokens total supply will still grow, but the reserve balance will remain the same) which might be problematic. We suggest to further investigate the potential security implications of that design.

- The idea behind implementing Bonding Curves in a batched way is mostly based on fighting against front-running. While getting rid of front-running in its pure way, the batching algorithm introduces multiple price manipulation techniques that are very similar to front-running and sometimes even more dangerous.

    - If some whale buys a huge amount of tokens, it's profitable for anyone to buy tokens also in this batch because there is a very high chance that the price will go up after this batch and the next batch will be with a much higher price. Because of doing so, the price will go even higher. It's bad for the whale because they'll buy for a slightly higher price than they intended to. The thing is that a whale will not immediately sell the tokens back (because of high costs). But all other traders can sell everything back and make a profit.

6 Issues

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 Collaterals are not guaranteed to be returned after a batch is cancelled Major ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising#162

Description

When traders open buy orders, they also transfer collateral tokens to the market maker contract. If the current batch is going to be cancelled, there is a chance that these collateral tokens will not be returned to the traders.

Examples

If a current `collateralsToBeClaimed` value is zero on a batch initialization and in this new batch only buy orders are submitted, `collateralsToBeClaimed` value will still stay zero.

At the same time if in `Tap` contract `tapped` amount was bigger than `_maximumWithdrawal()` on batch initialisation, `_maximumWithdrawal()` will most likely increase when the traders transfer new collateral tokens with the buy orders. And a beneficiary will be able to withdraw part of these tokens. Because of that, there might be not enough tokens to withdraw by the traders if the batch is cancelled.

It's partially mitigated by having `floor` value in `Tap` contract, but if there are more collateral tokens in the batch than `floor`, the issue is still valid.

Recommendation

Ensure that `tapped` is not bigger than `_maximumWithdrawal()`

6.2 Fees can be changed during the batch Major ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`0941f53` by storing current fee in meta batch.

Description

Shareholders can vote to change the fees. For buy orders, fees are withdrawn immediately when order is submitted and the only risk is frontrunning by the shareholder's voting contract.

For sell orders, fees are withdrawn when a trader claims an order and withdraws funds in `_claimSellOrder` function:

code/apps/batched-bancor-market-maker/contracts/BatchedBancorMarketMaker.sol:L790-L792

```
if (fee > 0) {
    reserve.transfer(_collateral, beneficiary, fee);
}
```

Fees can be changed between opening order and claiming this order which makes the fees unpredictable.

Recommendation

Fees for an order should not be updated during its lifetime.

6.3 Bancor formula should not be updated during the batch Major

✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`a8c2e21` by storing a ref to the Formula with the meta batch.

Description

Shareholders can vote to change the bancor formula contract. That can make a price in the current batch unpredictable.

code/apps/batched-bancor-market-maker/contracts/BatchedBancorMarketMaker.sol:L212-L216

```
function updateFormula(IBancorFormula _formula) external auth(UPDATE_FORMULA_ROLE) {
    require(isContract(_formula), ERROR_CONTRACT_IS_EOA);

    _updateFormula(_formula);
}
```

Recommendation

Bancor formula update should be executed in the next batch or with a timelock that is greater than batch duration.

6.4 Maximum slippage shouldn't be updated for the current batch

Major ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`aa4f03e` by storing slippage with the batch.

Description

When anyone submits a new order, the batch price is updated and it's checked whether the price slippage is acceptable. The problem is that the maximum slippage can be updated during the batch and traders cannot be sure that price is limited as they initially expected.

code/apps/batched-bancor-market-maker/contracts/BatchedBancorMarketMaker.sol:L487-L489

```
function _slippageIsValid(Batch storage _batch, address _collateral) internal view re
    uint256 staticPricePPM = _staticPricePPM(_batch.supply, _batch.balance, _batch.re
    uint256 maximumSlippage = collaterals[_collateral].slippage;
```

Additionally, if a maximum slippage is updated to a lower value, some of the orders that should lower the current slippage will also revert.

Recommendation

Save a slippage value on batch initialization and use it during the current batch.

6.5 AragonFundraisingController - an untapped address in toReset can block attempts of opening Trading after presale Major

✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@ `9451147` by checking if token is tapped. Gas consumption is increased due to external call to Tap to check if token is actually tapped. The number of tokens to be reset is capped.

Description

AragonFundraisingController can be initialized with a list of token addresses `_toReset` that are to be reset when trading opens after the presale. These addresses are supposed to be addresses of tapped tokens. However, the list needs to be known when initializing the contract but the tapped tokens are added after initialization when calling `addCollateralToken` (and tapped with `_rate>0`). This can lead to an inconsistency that blocks `openTrading`.

code/apps/aragon-fundraising/contracts/AragonFundraisingController.sol:L99-L102

```
for (uint256 i = 0; i < _toReset.length; i++) {
    require(_tokenIsContractOrETH(_toReset[i]), ERROR_INVALID_TOKENS);
    toReset.push(_toReset[i]);
}
```

In case a token address makes it into the list of `toReset` tokens that is not tapped it will be impossible to `openTrading` as `tap.resetTappedToken(toReset[i]);` throws for untapped tokens. According to the permission setup in `FundraisingMultisigTemplate` only Controller can call `Marketmaker.open`

code/apps/aragon-fundraising/contracts/AragonFundraisingController.sol:L163-L169

```
function openTrading() external auth(OPEN_TRADING_ROLE) {
    for (uint256 i = 0; i < toReset.length; i++) {
        tap.resetTappedToken(toReset[i]);
    }

    marketMaker.open();
}
```

Recommendation

Instead of initializing the Controller with a list of tapped tokens to be reset when trading opens, add a flag to `addCollateralToken` to indicate that the token should be reset when calling `openTrading`, making sure only tapped tokens are added to this list. This also allows adding tapped tokens that are to be reset at a later point in time.

6.6 Tap payments inconsistency Major ✓ Fixed

Resolution
Fixed with AragonBlack/fundraising#162

Description

Every time project managers want to withdraw tapped funds, the maximum amount of withdrawable funds is calculated in `tap._maximumWithdrawal` function. The method ensures that project managers can only withdraw unlocked funds (balance exceeding the collaterals `minimum` comprised of the collaterals configured `floor` including the minimum tokens to `hold`) even though their allowance might be higher.

1. if there are **no** unlocked funds available, the maximum withdrawal is zero (`balance <= minimum`).
2. if there are unlocked funds available (`balance > minimum`) and the allowance (`tapped`) would result in a `balance >= minimum`, the maximum withdrawal amount is the calculated allowance `tapped`.
3. if there are unlocked funds available (`balance > minimum`) and the allowance (`tapped`) would result in a `balance < minimum`, the maximum withdrawal amount `tapped` is capped to `balance - minimum` to ensure that the remaining collateral `balance` is at least at the `minimum` and not below.

This means that in the case of (3) if there are not enough funds to withdraw `tapped` (`time*tap_rate`) amount of tokens, it gets truncated and only a part of tapped tokens gets withdrawn.

code/apps/tap/contracts/Tap.sol:L239-L255

```
function _maximumWithdrawal(address _token) internal view returns (uint256) {
    uint256 toBeClaimed = controller.collateralsToBeClaimed(_token);
    uint256 floor = floors[_token];
    uint256 minimum = toBeClaimed.add(floor);
    uint256 balance = _token == ETH ? address(reserve).balance : ERC20(_token).static
    uint256 tapped = (_currentBatchId().sub(lastWithdrawals[_token])).mul(rates[_toke

    if (minimum >= balance) {
        return 0;
    }

    if (balance >= tapped.add(minimum)) {
        return tapped;
    }

    return balance.sub(minimum);
}
```

The problem is that the remaining tokens (`tapped - capped_tapped`) cannot be claimed afterward and `tapped` value is reset to zero.

Remediation

In case the maximum withdrawal amount gets capped, the information about the remaining tokens that the project team should have been able to withdraw should be kept to allow them to withdraw the tokens at a later point in time when there are enough funds for it.

6.7 [New] Tapped collaterals can be bought by traders Medium

Won't Fix

Resolution
This behaviour is intentional and if there is not a lot of funds in the pool, shareholders have a priority to buy tokens even if these tokens can already be withdrawn by the beneficiary. It is done in order to protect shareholders in case if

the project is dying and running out of funds. The downside of this behaviour is that it creates an additional incentive for the beneficiary to withdraw tapped tokens as soon and as often as possible which creates a race condition.

Description

When a trader submits a sell order, `_openSellOrder()` function checks that there are enough tokens in `reserve` by calling `_poolBalanceIsSufficient` function

code/apps/batched-bancor-market-maker/contracts/BatchedBancorMarketMaker.sol:L483-L485

```
function _poolBalanceIsSufficient(address _collateral) internal view returns (bool) {
    return controller.balanceOf(address(reserve), _collateral) >= collateralsToBeClai
}
```

the problem is that because `collateralsToBeClaimed[_collateral]` has increased, `controller.balanceOf(address(reserve), _collateral)` could also increase. It happens so because `controller.balanceOf()` function subtracts tapped amount from the reserve's balance.

code/apps/aragon-fundraising/contracts/AragonFundraisingController.sol:L358-L366

```
function balanceOf(address _who, address _token) public view isInitialized returns (u
    uint256 balance = _token == ETH ? _who.balance : ERC20(_token).staticBalanceOf(_w

    if (_who == address(reserve)) {
        return balance.sub(tap.getMaximumWithdrawal(_token));
    } else {
        return balance;
    }
}
```

And `tap.getMaximumWithdrawal(_token)` could decrease because it depends on `collateralsToBeClaimed[_collateral]`

apps/tap/contracts/Tap.sol:L231-L264

```
function _tappedAmount(address _token) internal view returns (uint256) {
    uint256 toBeKept = controller.collateralsToBeClaimed(_token).add(floors[_token]);
    uint256 balance = _token == ETH ? address(reserve).balance : ERC20(_token).static
    uint256 flow = (_currentBatchId().sub(lastTappedAmountUpdates[_token])).mul(rates
    uint256 tappedAmount = tappedAmounts[_token].add(flow);
    /**
     * whatever happens enough collateral should be
     * kept in the reserve pool to guarantee that
     * its balance is kept above the floor once
     * all pending sell orders are claimed
    */

    /**
     * the reserve's balance is already below the balance to be kept
     * the tapped amount should be reset to zero
    */
    if (balance <= toBeKept) {
        return 0;
    }

    /**
     * the reserve's balance minus the upcoming tap flow would be below the balance to
     * the flow should be reduced to balance - toBeKept
    */
    if (balance <= toBeKept.add(tappedAmount)) {
        return balance.sub(toBeKept);
    }

    /**
     * the reserve's balance minus the upcoming flow is above the balance to be kept
     * the flow can be added to the tapped amount
    */
    return tappedAmount;
}
```

That means that the amount that beneficiary can withdraw has just decreased, which should not be possible.

Recommendation

Ensure that `tappedAmount` cannot be decreased once updated.

6.8 Presale - `contributionToken` double cast and invalid comparison Medium ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`61f5803`.

Description

The Presale can be configured to accept `ETH` or a valid `ERC20` token. This token is stored as an `ERC20` contract type in the state variable `contributionToken`. It is then directly compared to constant `ETH` which is `address(0x0)` in various locations. Additionally, the `_transfer` function double casts the `token` to `ERC20` if the `contributionToken` is passed as an argument.

Examples

- `contribute` - invalid comparison of contract type against `address(0x00)`. Even though this is accepted in solidity `<0.5.0` it is going to raise a compiler error with newer versions (`>=0.5.0`).

code/apps/presale/contracts/Presale.sol:L163-L170

```
function contribute(address _contributor, uint256 _value) external payable nonReentra
    require(state() == State.Funding, ERROR_INVALID_STATE);

    if (contributionToken == ETH) {
        require(msg.value == _value, ERROR_INVALID_CONTRIBUTE_VALUE);
    } else {
        require(msg.value == 0,      ERROR_INVALID_CONTRIBUTE_VALUE);
    }
```

- `_transfer` - double cast `token` to `ERC20` if it is the contribution token.

code/apps/presale/contracts/Presale.sol:L344-L344

```
require(ERC20(_token).safeTransfer(_to, _amount), ERROR_TOKEN_TRANSFER_REVERTED);
```

Recommendation

`contributionToken` can either be `ETH` or a valid `ERC20` contract address. It is therefore recommended to store the token as an address type instead of the more precise

contract type to resolve the double cast and the invalid contract type to address comparison or cast the `ERC20` type to `address()` before comparison.

6.9 Fees are not returned for buy orders if a batch is canceled

Medium Won't Fix

Resolution

This issue has been addressed with the following statement:

> The only situation where a batch can be cancelled is when a collateral is un-whitelisted. This is obviously a very critical operation that we introduced just in case the collateral happened to be malicious token. Handling the ability to return fees in case a batch order is cancelled would thus add a lot of computation overhead for: a. a very unlikely situation b. where the fees would anyhow be returned in a malicious token. c. given a small amount [it's a fee and not the main amount]. We figured out that it was a bad decision to add gas overhead to all orders just to prevent this situation.

Description

Every trader pays fees on each buy order and transfers it directly to the `beneficiary`.

code/apps/batched-bancor-market-maker/contracts/BatchedBancorMarketMaker.sol:L706-L713

```
uint256 fee = _value.mul(buyFeePct).div(PCT_BASE);
uint256 value = _value.sub(fee);

// collect fee and collateral
if (fee > 0) {
    _transfer(_buyer, beneficiary, _collateral, fee);
}
_transfer(_buyer, address(reserve), _collateral, value);
```

If the batch is canceled, fees are not returned to the traders because there is no access to the beneficiary account.

Additionally, fees are returned to traders for all the sell orders if the batch is canceled.

Recommendation

Consider transferring fees to a beneficiary only after the batch is over.

6.10 Tap - Controller should not be updateable Medium ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@ `f6054443` by removing update functionality.

Description

Similar to the issue 6.11, `Tap` allows updating the `Controller` contract it is using. The permission is currently not assigned in the `FundraisingMultisigTemplate` but might be used in custom deployments.

code/apps/tap/contracts/Tap.sol:L117-L125

```
/**
 * @notice Update controller to `_controller`
 * @param _controller The address of the new controller contract
*/
function updateController(IAragonFundraisingController _controller) external auth(UPD
    require(isContract(_controller), ERROR_CONTRACT_IS_EOA);

    _updateController(_controller);
}
```

Recommendation

To avoid inconsistencies, we suggest to remove this functionality and provide a guideline on how to safely upgrade components of the system.

6.11 Tap - reserve can be updated in Tap but not in MarketMaker or Controller Medium ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@ `987720b1` by removing update functionality.

Description

The address of the pool/reserve contract can be updated in `Tap` if someone owns the `UPDATE_RESERVE_ROLE` permission. The permission is currently not assigned in the template.

The reserve is being referenced by multiple Contracts. `Tap` interacts with it to transfer funds to the beneficiary, `Controller` adds new protected tokens, and `MarketMaker` transfers funds when someone sells their Shareholder token.

Updating reserve only in `Tap` is inconsistent with the system as the other contracts are still referencing the old reserve unless they are updated via the Aragon Application update mechanisms.

code/apps/tap/contracts/Tap.sol:L127-L135

```
/**
 * @notice Update reserve to `_reserve`
 * @param _reserve The address of the new reserve [pool] contract
*/
function updateReserve(Vault _reserve) external auth(UPDATE_RESERVE_ROLE) {
    require(isContract(_reserve), ERROR_CONTRACT_IS_EOA);

    _updateReserve(_reserve);
}
```

Recommendation

Remove the possibility to update reserve in `Tap` to keep the system consistent. Provide information about update mechanisms in case the reserve needs to be updated for all components.

6.12 Presale can be opened earlier than initially assigned date

Medium ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`0726e29`.

Description

There are 2 ways how presale opening date can be assigned. Either it's defined on initialization or the presale will start when `open()` function is executed.

code/apps/presale/contracts/Presale.sol:L144-L146

```
if (_openDate != 0) {
    _setOpenDate(_openDate);
}
```

The problem is that even if `openDate` is assigned to some non-zero date, it can still be opened earlier by calling `open()` function.

code/apps/presale/contracts/Presale.sol:L152-L156

```
function open() external auth(OPEN_ROLE) {
    require(state() == State.Pending, ERROR_INVALID_STATE);

    _open();
}
```

Recommendation

Require that `openDate` is not set (`0`) when someone manually calls the `open()` function.

6.13 Presale - should not allow zero value contributions Minor ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`6a6e222`.

Description

The Presale accepts zero value contributions emitting a contribution event if none of the Aragon components (TokenManager, MinimeToken) raises an exception.

code/apps/presale/contracts/Presale.sol:L163-L173

```
function contribute(address _contributor, uint256 _value) external payable nonReentra
    require(state() == State.Funding, ERROR_INVALID_STATE);

    if (contributionToken == ETH) {
        require(msg.value == _value, ERROR_INVALID_CONTRIBUTE_VALUE);
    } else {
        require(msg.value == 0,      ERROR_INVALID_CONTRIBUTE_VALUE);
    }

    _contribute(_contributor, _value);
}
```

Recommendation

Reject zero value `ETH` or `ERC20` contributions.

6.14 Compiler Warnings - Function state mutability can be restricted to view Minor ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`cfd677a`.

Description

The following methods are not state-changing and can, therefore, be restricted to `view`.

```
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:485:5: Warning: Function state mutabil
    function _daoCache() internal returns (Kernel dao) {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:491:5: Warning: Function state mutabil
    function _boardAppsCache() internal returns (TokenManager boardTM, Voting boardVoting, Vault vaul
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:500:5: Warning: Function state mutabil
    function _shareAppsCache() internal returns (TokenManager shareTM, Voting shareVoting) {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:507:5: Warning: Function state mutabil
    function _fundraisingAppsCache() internal returns (Agent reserve, Presale presale, MarketMaker ma
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:541:5: Warning: Function state mutabil
    function _vaultCache() internal returns (Vault vault) {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:547:5: Warning: Function state mutabil
    function _shareTMCache() internal returns (TokenManager shareTM) {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:553:5: Warning: Function state mutabil
    function _reserveCache() internal returns (Agent reserve) {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:559:5: Warning: Function state mutabil
    function _presaleCache() internal returns (Presale presale) {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:565:5: Warning: Function state mutabil
    function _controllerCache() internal returns (Controller controller) {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:577:5: Warning: Function state mutabil
    function _ensureBoardAppsCache() internal {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:588:5: Warning: Function state mutabil
    function _ensureShareAppsCache() internal {
    ^ (Relevant source part starts here and spans across multiple lines).
,/templates/multisig/contracts/FundraisingMultisigTemplate.sol:597:5: Warning: Function state mutabil
    function _ensureFundraisingAppsCache() internal {
    ^ (Relevant source part starts here and spans across multiple lines).
```

Recommendation

Restrict function state mutability of the listed methods to `view`.

6.15 FundraisingMultisigTemplate - should use `BaseTemplate._createPermissionForTemplate()` to assign permissions to itself Minor ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`dd153e0`.

Description

The template temporarily assigns permissions to itself to be able to configure parts of the system. This can either be done by calling `acl.createPermission(address(this), app, role, manager)` or by using a distinct method provided with the DAO-Templates BaseTemplate `_createPermissionForTemplate`.

We suggest that in order to make it clear that permissions are assigned to the template and make it easier to audit that permissions are either revoked or transferred before the DAO is transferred to the new user, the method provided and used with the default Aragon DAO-Templates should be used.

- use `createPermission` if permissions are assigned to an entity other than the template contract.
- use `_createPermissionForTemplate` when creating permissions for the template contract.

code/templates/multisig/contracts/FundraisingMultisigTemplate.sol:L333-L334

```
// create and grant ADD_PROTECTED_TOKEN_ROLE to this template
acl.createPermission(this, controller, controller.ADD_COLLATERAL_TOKEN_ROLE(), this);
```

Sidenote: pass `address(this)` instead of the contract instance to `createPermission`.

Recommendation

Use `BaseTemplate._createPermissionForTemplate` to assign permissions to the template.

6.16 FundraisingMultisigTemplate - misleading comments Minor

✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`3b4700a` and AragonBlack/fundraising@`40c465fc`.

Description

The comment mentions `ADD_PROTECTED_TOKEN_ROLE` but permissions for `ADD_COLLATERAL_TOKEN_ROLE` are created.

code/templates/multisig/contracts/FundraisingMultisigTemplate.sol:L333-L334

```
// create and grant ADD_PROTECTED_TOKEN_ROLE to this template
acl.createPermission(this, controller, controller.ADD_COLLATERAL_TOKEN_ROLE(), this);
```

code/templates/multisig/contracts/FundraisingMultisigTemplate.sol:L355-L356

```
// transfer ADD_PROTECTED_TOKEN_ROLE
_transferPermissionFromTemplate(acl, controller, shareVoting, controller.ADD_COLLATER
```

Recommendation

`ADD_PROTECTED_TOKEN_ROLE` in the comment should be `ADD_COLLATERAL_TOKEN_ROLE`.

6.17 FundraisingMultisigTemplate - unnecessary cast to address

Minor ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`0e00269`.

Description

The addresses of DAI (argument `address _dai`) and AND (argument `address _ant`) are unnecessarily cast to `address`.

code/templates/multisig/contracts/FundraisingMultisigTemplate.sol:L58-L76

```
constructor(
    DAOFactory              _daoFactory,
    ENS                     _ens,
    MiniMeTokenFactory      _miniMeFactory,
    IFIFSResolvingRegistrar _aragonID,
    address                 _dai,
    address                 _ant
)
    BaseTemplate(_daoFactory, _ens, _miniMeFactory, _aragonID)
    public
{
    _ensureAragonIdIsValid(_aragonID);
    _ensureMiniMeFactoryIsValid(_miniMeFactory);
    _ensureTokenIsContractOrETH(_dai);
    _ensureTokenIsContractOrETH(_ant);

    collaterals.push(address(_dai));
    collaterals.push(address(_ant));
}
```

Recommendation

Both arguments are already of type `address`, therefore remove the explicit cast to `address()` when pushing to the `collaterals` array.

6.18 FundraisingMultisigTemplate - unused import `ERC20` Minor

✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`73481d1`.

Description

The interface `ERC20` is imported but never used.

code/templates/multisig/contracts/FundraisingMultisigTemplate.sol:L4-L4

```
import "@aragon/os/contracts/lib/token/ERC20.sol";
```

Recommendation

Remove the unused import.

6.19 FundraisingMultisigTemplate - DAI/ANT token address cannot be zero Minor ✓ Fixed

Resolution

Fixed with AragonBlack/fundraising@`da561ce`.

Description

The fundraising template is configured with the `DAI` and `ANT` token address upon deployment and checks if the provided addresses are valid. The check performed is `_ensureTokenIsContractOrETH()` which allows the `address(0)` (constant for `ETH`) for the token contracts. However, `address(0)` is not a valid option for either `DAI` or `ANT` and the contract expects a valid token address to be provided as the deployment of a new DAO will have unexpected results (collateral `ETH` is added instead of an ERC20 token) or fail (`DAI` == `ANT` == `0x0`).

code/templates/multisig/contracts/FundraisingMultisigTemplate.sol:L71-L72

```
_ensureTokenIsContractOrETH(_dai);
_ensureTokenIsContractOrETH(_ant);
```

code/templates/multisig/contracts/FundraisingMultisigTemplate.sol:L572-L575

```
 function _ensureTokenIsContractOrETH(address _token) internal view returns (bool) {
    require(isContract(_token) || _token == ETH, ERROR_BAD_SETTINGS);
}
```

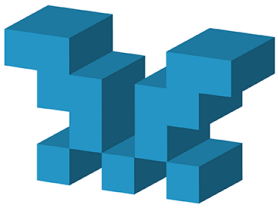
Recommendation

Use `isContract()` instead of `_ensureTokenIsContractOrETH()` and optionally require that `collateral[0] != collateral[1]` as an additional check to prevent that the fundraising template is being deployed with an invalid configuration.

7 Tool-Based Analysis

Several tools were used to perform an automated analysis of the reviewed contracts. These issues were reviewed by the audit team, and relevant issues are listed in the Issue Details section.

7.1 MythX



MythX is a security analysis API for Ethereum smart contracts. It performs multiple types of analysis, including fuzzing and symbolic execution, to detect many common vulnerability types. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on MythX can be found at mythx.io.

7.2 Ethlint

Ethlint is an open-source project for linting Solidity code. Only security-related issues were reviewed by the audit team.

ETHLINT
Analyse Solidity for Style and Security

Below is the raw output of the Ethlint vulnerability scan:

```
$ solium --version
Solium version 1.2.5

$ solium -d
apps/aragon-fundraising/contracts/AragonFundraisingController.sol
  370:5    error    Only use indent of 4 spaces.    indentation

templates/multisig/contracts/FundraisingMultisigTemplate.sol
  573:5    error    Only use indent of 4 spaces.    indentation
```

7.3 Surya

Surya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Below is a complete list of functions with their visibility and modifiers (please use horizontal scroll to view all columns):

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
AragonFundraisingController	Implementation	EtherTokenConstant, IsContract, IAragonFundraisingController, AragonApp		
└	initialize	External ❗	🛑	onlyInit
└	updateBeneficiary	External ❗	🛑	auth
└	updateFees	External ❗	🛑	auth
└	openPresale	External ❗	🛑	auth
└	closePresale	External ❗	🛑	isInitialized
└	contribute	External ❗	💵	auth
└	refund	External ❗	🛑	isInitialized
└	openTrading	External ❗	🛑	auth
└	openBuyOrder	External ❗	💵	auth
└	openSellOrder	External ❗	🛑	auth
└	claimBuyOrder	External ❗	🛑	isInitialized
└	claimSellOrder	External ❗	🛑	isInitialized
└	addCollateralToken	External ❗	🛑	auth
└	reAddCollateralToken	External ❗	🛑	auth
└	removeCollateralToken	External ❗	🛑	auth

Contract	Type	Bases		
└	updateCollateralToken	External ❗	🛑	auth
└	updateMaximumTapRateIncreasePct	External ❗	🛑	auth
└	updateMaximumTapFloorDecreasePct	External ❗	🛑	auth
└	addTokenTap	External ❗	🛑	auth
└	updateTokenTap	External ❗	🛑	auth
└	withdraw	External ❗	🛑	auth
└	token	Public ❗		isInitialized
└	contributionToken	Public ❗		isInitialized
└	getMaximumWithdrawal	Public ❗		isInitialized
└	collateralsToBeClaimed	Public ❗		isInitialized
└	balanceOf	Public ❗		isInitialized
└	_tokenIsContractOrETH	Internal 🔒		
BancorFormula	Implementation	IBancorFormula, Utils		
└	<Constructor>	Public ❗	🛑	
└	calculatePurchaseReturn	Public ❗		NO ❗
└	calculateSaleReturn	Public ❗		NO ❗

Contract	Type	Bases		
└	calculateCrossConnectorReturn	Public ❗		NO ❗
└	power	Internal 🔒		
└	generalLog	Internal 🔒		
└	floorLog2	Internal 🔒		
└	findPositionInMaxExpArray	Internal 🔒		
└	generalExp	Internal 🔒		
└	optimalLog	Internal 🔒		
└	optimalExp	Internal 🔒		
BatchedBancorMarketMaker	Implementation	EtherTokenConstant, IsContract, AragonApp		
└	initialize	External ❗	🛑	onlyInit
└	open	External ❗	🛑	auth
└	updateFormula	External ❗	🛑	auth
└	updateBeneficiary	External ❗	🛑	auth
└	updateFees	External ❗	🛑	auth
└	addCollateralToken	External ❗	🛑	auth
└	removeCollateralToken	External ❗	🛑	auth
└	updateCollateralToken	External ❗	🛑	auth
└	openBuyOrder	External ❗	💵	auth

Contract	Type	Bases		
└	openSellOrder	External ❗	🛑	auth
└	claimBuyOrder	External ❗	🛑	nonReentrant isInitialized
└	claimSellOrder	External ❗	🛑	nonReentrant isInitialized
└	claimCancelledBuyOrder	External ❗	🛑	nonReentrant isInitialized
└	claimCancelledSellOrder	External ❗	🛑	nonReentrant isInitialized
└	getCurrentBatchId	Public ❗		isInitialized
└	getCollateralToken	Public ❗		isInitialized
└	getBatch	Public ❗		isInitialized
└	getStaticPricePPM	Public ❗		isInitialized
└	_staticPricePPM	Internal 🔒		
└	_currentBatchId	Internal 🔒		
└	_beneficiaryIsValid	Internal 🔒		
└	_feeIsValid	Internal 🔒		
└	_reserveRatioIsValid	Internal 🔒		
└	_tokenManagerSettingIsValid	Internal 🔒		

Contract	Type	Bases		
└	_collateralValueIsValid	Internal 🔒		
└	_bondAmountIsValid	Internal 🔒		
└	_collateralIsWhitelisted	Internal 🔒		
└	_batchIsOver	Internal 🔒		
└	_batchIsCancelled	Internal 🔒		
└	_userIsBuyer	Internal 🔒		
└	_userIsSeller	Internal 🔒		
└	_poolBalanceIsSufficient	Internal 🔒		
└	_slippageIsValid	Internal 🔒		
└	_buySlippageIsValid	Internal 🔒		
└	_sellSlippageIsValid	Internal 🔒		
└	_currentBatch	Internal 🔒	🛑	
└	_open	Internal 🔒	🛑	
└	_updateBeneficiary	Internal 🔒	🛑	
└	_updateFormula	Internal 🔒	🛑	
└	_updateFees	Internal 🔒	🛑	
└	_cancelCurrentBatch	Internal 🔒	🛑	
└	_addCollateral	Internal 🔒	🛑	

Contract	Type	Bases		
	Token			
└	_removeCollateralToken	Internal 🔒	🛑	
└	_updateCollateralToken	Internal 🔒	🛑	
└	_openBuyOrder	Internal 🔒	🛑	
└	_openSellOrder	Internal 🔒	🛑	
└	_claimBuyOrder	Internal 🔒	🛑	
└	_claimSellOrder	Internal 🔒	🛑	
└	_claimCancelledBuyOrder	Internal 🔒	🛑	
└	_claimCancelledSellOrder	Internal 🔒	🛑	
└	_updatePricing	Internal 🔒	🛑	
└	_transfer	Internal 🔒	🛑	
Presale	Implementation	EtherTokenConstant, IsContract, AragonApp		
└	initialize	External ❗	🛑	onlyInit
└	open	External ❗	🛑	auth
└	contribute	External ❗	💵	nonReentrant auth
└	refund	External ❗	🛑	nonReentrant isInitialized

Contract	Type	Bases		
└	close	External ❗	🛑	nonReentrant isInitialized
└	contributionToTokens	Public ❗		isInitialized
└	state	Public ❗		isInitialized
└	_timeSinceOpen	Internal 🔒		
└	_setOpenDate	Internal 🔒	🛑	
└	_setVestingDatesWhenOpenDateIsKnown	Internal 🔒	🛑	
└	_open	Internal 🔒	🛑	
└	_contribute	Internal 🔒	🛑	
└	_refund	Internal 🔒	🛑	
└	_close	Internal 🔒	🛑	
└	_transfer	Internal 🔒	🛑	
Tap	Implementation	TimeHelpers, EtherTokenConstant, IsContract, AragonApp		
└	initialize	External ❗	🛑	onlyInit
└	updateController	External ❗	🛑	auth
└	updateReserve	External ❗	🛑	auth
└	updateBeneficiary	External ❗	🛑	auth
└	updateMaximumTapRateInc	External ❗	🛑	auth

Contract	Type	Bases		
	reasePct			
└	updateMaximumTapFloorDecreasePct	External ❗	🛑	auth
└	addTappedToken	External ❗	🛑	auth
└	removeTappedToken	External ❗	🛑	auth
└	updateTappedToken	External ❗	🛑	auth
└	resetTappedToken	External ❗	🛑	auth
└	withdraw	External ❗	🛑	auth
└	getMaximumWithdrawal	Public ❗		isInitialized
└	_currentBatchId	Internal 🔒		
└	_maximumWithdrawal	Internal 🔒		
└	_beneficiaryIsValid	Internal 🔒		
└	_maximumTapFloorDecreasePctIsValid	Internal 🔒		
└	_tokenIsContractOrETH	Internal 🔒		
└	_tokenIsTapped	Internal 🔒		
└	_tapRateIsValid	Internal 🔒		

Contract	Type	Bases		
└	_tapUpdateIsValid	Internal 🔒		
└	_tapRateUpdateIsValid	Internal 🔒		
└	_tapFloorUpdateIsValid	Internal 🔒		
└	_updateController	Internal 🔒	🛑	
└	_updateReserve	Internal 🔒	🛑	
└	_updateBeneficiary	Internal 🔒	🛑	
└	_updateMaximumTapRateIncreasePct	Internal 🔒	🛑	
└	_updateMaximumTapFloorDecreasePct	Internal 🔒	🛑	
└	_addTappedToken	Internal 🔒	🛑	
└	_removeTappedToken	Internal 🔒	🛑	
└	_updateTappedToken	Internal 🔒	🛑	
└	_resetTappedToken	Internal 🔒	🛑	
└	_withdraw	Internal 🔒	🛑	
Fundraising MultisigTemplate	Implementation	EtherTokenConstant, BaseTemplate		

Contract	Type	Bases		
└	<Constructor>	Public ❗	🛑	BaseTemplate
└	prepareInstance	External ❗	🛑	NO ❗
└	installShareApps	External ❗	🛑	NO ❗
└	installFundraisingApps	External ❗	🛑	NO ❗
└	finalizeInstance	External ❗	🛑	NO ❗
└	_installBoardApps	Internal 🔒	🛑	
└	_installShareApps	Internal 🔒	🛑	
└	_installFundraisingApps	Internal 🔒	🛑	
└	_proxifyFundraisingApps	Internal 🔒	🛑	
└	_initializePresale	Internal 🔒	🛑	
└	_initializeMarketMaker	Internal 🔒	🛑	
└	_initializeTap	Internal 🔒	🛑	
└	_initializeController	Internal 🔒	🛑	
└	_setupCollaterals	Internal 🔒	🛑	
└	_setupBoardPermissions	Internal 🔒	🛑	
└	_setupSharePe	Internal 🔒	🛑	

Contract	Type	Bases		
	rmissions			
└	_setupFundraisingPermissions	Internal 🔒	🛑	
└	_cacheDao	Internal 🔒	🛑	
└	_cacheBoardApps	Internal 🔒	🛑	
└	_cacheShareApps	Internal 🔒	🛑	
└	_cacheFundraisingApps	Internal 🔒	🛑	
└	_daoCache	Internal 🔒	🛑	
└	_boardAppsCache	Internal 🔒	🛑	
└	_shareAppsCache	Internal 🔒	🛑	
└	_fundraisingAppsCache	Internal 🔒	🛑	
└	_clearCache	Internal 🔒	🛑	
└	_vaultCache	Internal 🔒	🛑	
└	_shareTMCache	Internal 🔒	🛑	
└	_reserveCache	Internal 🔒	🛑	
└	_presaleCache	Internal 🔒	🛑	
└	_controllerCache	Internal 🔒	🛑	
└	_ensureTokenIsContractOrETH	Internal 🔒		

Contract	Type	Bases		
└	_ensureBoardAppsCache	Internal 🔒	🛑	
└	_ensureShareAppsCache	Internal 🔒	🛑	
└	_ensureFundraisingAppsCache	Internal 🔒	🛑	
└	_registerApp	Internal 🔒	🛑	

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

7.4 Test Coverage Measurement

Testing is implemented using Truffle an all provided test cases pass. However, the Presale contract fails to generate coverage statistics.

MarketMaker

```
@ablack/fundraising-batched-bancor-market-maker:   161 passing (14m)
@ablack/fundraising-batched-bancor-market-maker: -------------------------------|----------|---------
@ablack/fundraising-batched-bancor-market-maker: File                           |  % Stmts | % Branch
@ablack/fundraising-batched-bancor-market-maker: -------------------------------|----------|---------
@ablack/fundraising-batched-bancor-market-maker:  contracts/                    |    99.54 |    81.36
@ablack/fundraising-batched-bancor-market-maker:   BatchedBancorMarketMaker.sol |    99.54 |    81.36
@ablack/fundraising-batched-bancor-market-maker: -------------------------------|----------|---------
@ablack/fundraising-batched-bancor-market-maker: All files                      |    99.54 |    81.36
@ablack/fundraising-batched-bancor-market-maker: -------------------------------|----------|---------
```

Controller

```
@ablack/fundraising-aragon-fundraising:   62 passing (9m)
@ablack/fundraising-aragon-fundraising: ----------------------------------|----------|----------|----
@ablack/fundraising-aragon-fundraising: File                              |  % Stmts | % Branch |  %
@ablack/fundraising-aragon-fundraising: ----------------------------------|----------|----------|----
@ablack/fundraising-aragon-fundraising:  contracts/                       |    97.96 |    94.44 |
@ablack/fundraising-aragon-fundraising:   AragonFundraisingController.sol |    97.96 |    94.44 |
@ablack/fundraising-aragon-fundraising: ----------------------------------|----------|----------|----
@ablack/fundraising-aragon-fundraising: All files                         |    97.96 |    94.44 |
@ablack/fundraising-aragon-fundraising: ----------------------------------|----------|----------|----
```

Tap

```
@ablack/fundraising-tap:   49 passing (3m)
@ablack/fundraising-tap: ------------|----------|----------|----------|----------|----------------|
@ablack/fundraising-tap: File        |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
@ablack/fundraising-tap: ------------|----------|----------|----------|----------|----------------|
@ablack/fundraising-tap:  contracts/ |    99.02 |    94.83 |      100 |    99.06 |                |
@ablack/fundraising-tap:   Tap.sol   |    99.02 |    94.83 |      100 |    99.06 |            313 |
@ablack/fundraising-tap: ------------|----------|----------|----------|----------|----------------|
@ablack/fundraising-tap: All files   |    99.02 |    94.83 |      100 |    99.06 |                |
```

Presale

```
lerna ERR! npm run test:coverage exited 1 in '@ablack/fundraising-presale'
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! @ablack/fundraising@1.0.0 coverage:presale: `lerna run --scope=@ablack/fundraising-presale -
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the @ablack/fundraising@1.0.0 coverage:presale script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.
```

Appendix 1 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in

any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

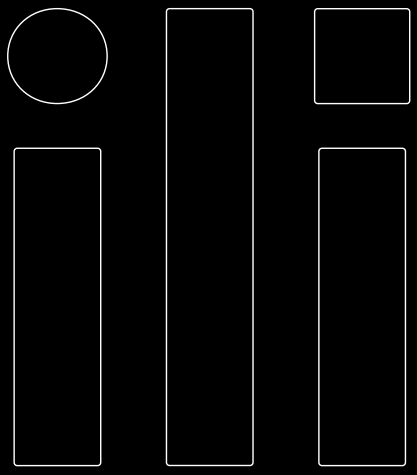
LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.

Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

CONTACT US



AUDITS

FUZZING

SCRIBBLE

BLOG

TOOLS

RESEARCH

ABOUT

CONTACT

CAREERS

PRIVACY POLICY

Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.