Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: TFY Liquid Staking Token
Website: thirdfy.com
Platform: Base Sepolia Network
Language: Solidity
Date: July 22nd, 2025

Table of contents

Email: audit@EtherAuthority.io

THIS IS A SECURITY AUDIT REPORT DOCUMENT WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contacted by the Thirdfy team to perform a security audit of the o33 Protocol smart contract's code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on July 22nd, 2025.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The o33 Protocol Contracts handle multiple contracts, and all contracts have different functions.
    - **o33:** The o33 contract is a liquid staking wrapper for xTFY, enabling auto-compounding of rewards, delegated voting, and bribe claiming. It integrates with external vote and reward modules, supports gasless meta-transactions, and includes security measures like epoch locks and whitelisted aggregators.
    - **xTFY:** The xTFY is a yield-bearing ERC20 token representing staked TFY with slashing and vesting mechanics. It enables emissions conversion, vesting-based exit strategies, rebasing via a VoteModule, and transfer restrictions to ensure governance control. The contract also supports pause control, migration, and rescue functionalities via the ACCESS_HUB.
    - **VoterV4:** The VoterV4 is the core governance contract of the TFY protocol that manages voting, emissions distribution, and gauge control. It supports legacy, concentrated liquidity (CL), and Ichi Vault gauges with UUPS upgradability, delegation, and per-period vote accounting. It integrates with the xTFY staking token and VoteModule for modular governance.
    - **AccessHub:** AccessHub is the central governance and access control contract for the TFY Protocol, using UUPS upgradeability and OpenZeppelin's role-based permissions. It coordinates core modules like Voter, xTFY, Minter, o33, and VoteModule, and enables secure execution,

upgrades, parameter management, and emergency actions through a timelock-controlled system.

- **FeeCollector:** FeeCollector is a fee aggregation and distribution contract for Algebra-based pools within the TFY Protocol. It collects protocol fees from pools, splits them between the treasury and gauge-linked fee distributors, and supports withdrawals from the AlgebraCommunityVault under role-based permissions. It provides granular event logging and safeguards for treasury and voter-controlled operations.
- **IchiBribeDistributor:** IchiBribeDistributor manages the deposit and distribution of bribes for a specific IchiVaultGauge based on user voting weights. Bribes are deposited for upcoming voting periods and later claimed by voters proportionally to their vote share. The contract ensures only the authorized VoterV4 can submit vote weights and enforces token whitelisting for bribes.
- **IchiVaultGauge:** IchiVaultGauge manages time-weighted reward distribution to users based on their participation in paired Ichi Vaults during each reward period (weekly). It allows a designated recorder to submit user "share-seconds" data off-chain and supports both voter-notified and externally deposited rewards. Whitelisted tokens can be used as rewards, and users can claim them per completed period.
- **Minter:** Minter handles the emission schedule for the TFY token, minting weekly rewards with a configurable growth/decay multiplier and enforcing a max supply cap. It interacts with a Voter contract to distribute emissions and can trigger a rebase in the xTFY contract. Governance can adjust the emissions multiplier with a 25% per-epoch deviation limit.
- **PositionOracle:** PositionOracle feeds time-in-range data of Uniswap V3 NFT positions to the Voter contract for use in gauges. It supports normal and fallback modes for data submission, with role-based access for an operator and emergency admin. The contract ensures batch-safe updates, enabling accurate reward distribution even in subgraph or data feed failures.
- **RevenueToRebaseManager:** RevenueToRebaseManager automates the weekly distribution of protocol revenue by burning a portion of TFY tokens and rebasing the rest to stakers. It supports community governance to override distribution ratios per epoch, uses UUPS upgradeability, and

enforces security via access control, reentrancy protection, and emergency pause mechanisms.

- **Thirdfy:** Thirdfy is a mintable, burnable ERC20 token with permit support, designed for the TFY protocol's emissions system. It restricts minting access to a designated minter contract, which typically handles weekly emissions based on governance decisions.
- **ThirdfyTimelock:** ThirdfyTimelock is a governance contract extending OpenZeppelin's TimelockController, used to manage delayed execution of proposals within the TFY protocol. It ensures secure and transparent upgrades or parameter changes by enforcing a minimum delay between proposal approval and execution.
- **VoteModule:** VoteModule is the TFY Protocol's core staking contract, enabling xTFY deposits for voting power and dual reward streams. It securely distributes both protocol emissions (rebase rewards) and external revenue rewards, with cooldown protection, delegation support, and robust access control via AccessHub.
- **ClGaugeFactory:** The ClGaugeFactory is a governance-controlled factory contract for deploying and managing Concentrated Liquidity (CL) gauge contracts within the TFY Protocol. It handles role assignments (voter, nfpManager, accessHub) and maintains a registry of created gauges.
- **IchiBribeDistributorFactory:** The IchiBribeDistributorFactory is a UUPS upgradeable factory contract used to deploy IchiBribeDistributor instances for distributing bribes to gauges. It includes access control via AccessHub, supports governance via VoterV4, and tracks the initial implementation address.
- **IchiVaultGaugeFactory:** The IchiVaultGaugeFactory is a UUPS upgradeable factory contract for deploying IchiVaultGauge instances linked to Ichi Vault pairs. It supports secure role-based access via AccessHub and VoterV4, and ensures controlled gauge creation with event logging and upgrade flexibility.

- This audit scope has included 16 smart contract files, 18 interface files, and 4 libraries files.
- The o33 Token contracts inherit the Initializable, UUPSUpgradeable, ERC20, IERC20, Pausable, Math, EnumerableSet, ERC4626, SafeERC20,

ReentrancyGuard, ReentrancyGuardUpgradeable, TimelockController ,ERC20Burnable, ERC20Permit, Ownable, AccessControlEnumerableUpgradeable standard smart contracts from the OpenZeppelin library.

- These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for o33 Token Smart Contracts**
Platform	**Base Sepolia Network**
Language	**Solidity**
File 1	o33.sol
File 1 MD5 hash	DAF54C8F51BE891EBF9CF38FABD75092
File 2	xTFY.sol
File 2 MD5 hash	3308E3B86EE2CAF4A4C27257DA65A31B
File 3	VoterV4.sol
File 3 MD5 hash	1018D3920C0D2AD09B3EF6DC44C28A01
File 4	AccessHub.sol
File 4 MD5 hash	61CE0333E6840CA58AA0D91977975B04
Updated File 4 MD5 hash	BCB5BE0F967E8CC3C350FAB9B284E5D8
File 5	FeeCollector.sol
File 5 MD5 hash	CE78073E427CB77301F1369FF4F8DE44
File 6	IchiBribeDistributor.sol
File 6 MD5 hash	B890A22058217EF23372B3C9FFA4F8A1
File 7	IchiVaultGauge.sol
File 7 MD5 hash	44EE8CE6F9BFF6C03405A527C2110512
File 8	Minter.sol

File 8 MD5 hash	D8CBD5595D732E4601E0D3B6AC2D8AA0
Updated File 8 MD5 hash	DC773CA7C0F4420F5A04B55B6F638580
File 9	PositionOracle.sol
File 9 MD5 hash	E37C22DDFA583EF3CABE3F63FD96D035
File 10	RevenueToRebaseManager.sol
File 10 MD5 hash	B3EA4C080F115A027AC2CDDDA16EFF92
File 11	Thirdfy.sol
File 11 MD5 hash	2A47D658BD2E25CA2A5BD43E41D39B6A
File 12	ThirdfyTimelock.sol
File 12 MD5 hash	E12440DE058B29A0CDA591779D9CE314
File 13	VoteModule.sol
File 13 MD5 hash	ACE9AC889075680456FC8F226894EDF6
File 14	ClGaugeFactory.sol
File 14 MD5 hash	5CE0D77F39BECBE953849F2537E43E08
File 15	IchiBribeDistributorFactory.sol
File 15 MD5 hash	D2B9C569078F340B42E5F04FAA5DA131
File 16	IchiVaultGaugeFactory.sol
File 16 MD5 hash	30818913707082F7ACB58AEB914CF031
Audit Date	July 22nd, 2025
Revised Audit Date	August 7th, 2025

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1: o33.sol **Tokenomics:** • **Name:** TFY Liquid Staking Token • **Symbol:** o33 **o33 Smart Contract – Key Features:** • **ERC4626 Vault**: Wraps xTFY as a yield-bearing token o33. • **Autocompounding**: Converts TFY emissions → xTFY → deposits into VoteModule. • **Voting Integration**: Submits votes via a Voter each epoch for protocol optimization. • **Rebase Handling**: Claims and compounds TFY rebases into staked xTFY. • **Incentive Claiming**: Collects rewards via FeeDistributors. • **Token Swapping**: Swaps non-TFY rewards to TFY using whitelisted aggregators. • **Meta-Transactions**: Supports gasless user interactions via relayer signatures. • **Access Control**: operator executes logic, accessHub governs admin functions. • **Epoch Locking**: Disables deposits/withdrawals near epoch flips to prevent exploits. • **Rescue Mechanism**: Safely extracts non-core tokens without affecting staking state. • **Whitelisting**: Controlled access for aggregators and relayers.	**YES, This is valid.**
File 2: xTFY.sol **Tokenomics:** • **Name:** xTFY	**YES, This is valid.**

• **Symbol:** xTFY • **BASIS:** Denotes the denominator for basis point calculations (10,000 = 100%). • **SLASHING_PENALTY:** Represents a 50% penalty (5000 / 10,000) applied during slashing events. • **MIN_VEST:** Minimum vesting duration set to 14 days. • **MAX_VEST:** Maximum vesting duration set to 180 days. **xTFY Smart Contract – Key Features:** • **Staking Token**: Represents staked TFY with rebasing and vesting logic. • **Minting**: TFY → xTFY via convertEmissionsToken(). • **Vesting System**: Supports vest creation and linear vest exits with penalties. • **Instant Exit**: Exit with 50% slashing penalty (exit()). • **Rebase Engine**: Emits rewards to VoteModule each epoch or via emergency. • **Transfer Restrictions**: Enforced via whitelists (exempt, exemptTo). • **Governance Controls**: Timelocked control over operator, pausing, exemptions, and token rescue. • **Pause/Unpause**: Emergency control for system safety.	

File 3: VoterV4.sol
VoterV4 Contract:
• **Gauge Management**:
  ○ Supports Legacy, CL, and IchiVault gauge types.
  ○ Maps pools ↔ gauges.
• **Voting System**:
  ○ Epoch-based vote casting with power delegation.
  ○ Vote resets, pokes, and revoting supported.
• **Emissions Distribution**:
  ○ Distributes TFY and xTFY to active gauges.
	YES, This is valid.

○ Separate logic for IchiVault vs CL/legacy gauges. • **Bribe Support**: ○ Integrates with IchiVault Bribe Distributors. • **Admin Functions**: ○ Set factories, fee collectors, launcher plugin. ○ Set global xTFY:TFY emission split ratio. • **Upgradeable via UUPS**, ○ with controlled access via AccessHub and governor.	
File 4: AccessHub.sol **AccessHub – Key Features:** • **Role-Based Governance** — Uses AccessControl with timelock and operator roles. • **UUPS Upgradeable** — Secure and upgradeable via DEFAULT_ADMIN_ROLE. • **Centralized Module Control** — Manages Voter, xTFY, Minter, o33, VoteModule. • **Secure Execution** — Timelock can execute arbitrary calls on whitelisted targets. • **Voter Management** — Set governor, whitelist tokens, reset inactive voters. • **xTFY Ops** — Pause/unpause, redeem, migrate operator, rescue tokens. • **VoteModule Tuning** — Adjust cooldowns, rebase stream duration. • **Quick Reinit** — Easily rewire all dependencies post-deployment.	**YES, This is valid.**
File 5: FeeCollector.sol **FeeCollector – Key Features:** • **Treasury Management** ○ Configurable treasury address and treasuryFees (in basis points).	**YES, This is valid.**

○ Safely collects treasury shares before fee distribution. • **Protocol Fee Collection** ○ Collects fees from Algebra pools via collectProtocolFees. ○ Handles dead/no gauge gracefully by routing fees to the treasury. • **Voter Integration** ○ Read gauge info from IVoter. ○ Only voters can update the fee distributor or trigger vault withdrawals. • **Algebra Integration** ○ Collects via AlgebraPool.collectProtocolFees. ○ Withdraws tokens from AlgebraCommunityVault (single & batch). • **Robust Debugging** ○ Emits detailed Debug* events at every step for transparency. ○ Captures success/failure in transfers and role checks. • **Role-Based Access** ○ onlyTreasury and onlyVoter modifiers. ○ Checks Algebra factory role (FACTORY_WITHDRAWER_ROLE) for vault access. • **Modular Token Handling** ○ Uses SafeERC20 for secure transfers. ○ Transfers remaining fees to feeDist after treasury cut.	
File 6: IchiBribeDistributor.sol **<u>Tokenomics:</u>** • **DURATION:** Defines the length of a voting or bribe epoch, set to 7 days (1 week). **<u>IchiBribeDistributor – Key Features :</u>** • **Bribe Deposits**	**YES, This is valid.**

○ Accepts bribes for the next voting period. ○ Only accepts whitelisted tokens via VoterV4. ○ Tracked by totalBribeForPeriod. • **Vote-Based Rewards** ○ Users earn bribes based on vote weight per period. ○ earned() calculates unclaimed rewards. ○ Claims via claimBribes(tokens, periods). • **Secure Distribution** ○ Tracks: ■ totalWeightForPeriod ■ userWeightForPeriod ■ claimedBribes ■ Prevents double-claiming or over-claiming • **Epoch Control** ○ Periods are 1 week (7 days). ○ Rewards can only be claimed for past periods. • **VoterV4 Integration** ○ Only VoterV4 can call _depositVoteWeight(). ○ Updates user + total weights for upcoming epochs. • **Security** ○ Uses ReentrancyGuard and SafeERC20. ○ Reverts on: ■ Zero deposit ■ Invalid period ■ Nothing to claim ■ Mismatched input arrays • **Transparent & Modular** ○ All states are exposed via public mappings/interfaces. ○ Clean separation of deposit, claim, and vote logic.	
File 7: IchiVaultGauge.sol **<u>IchiVaultGauge – Key Features:</u>** • **Reward Distribution Based on Share-Seconds**	**YES, This is valid.**

- Uses time-weighted vault shares ("share-seconds") to calculate user rewards per epoch.
- Reward logic recorded off-chain via recordShareSeconds() by a shareRecorder.

- **Epoch-Based Rewarding**
  - Rewards are distributed per period (1 week).
  - Period data must be recorded after the period ends.
- **Vault & LP Rewards:**
  - Users can:
    - Earn rewards based on IchiVault staking.
    - Claim rewards via claimRewardsForPeriod.
  - Third parties can:
    - Deposit external LP rewards for the current period using depositExternalLPReward().
- **Permissions & Access Control:**
  - shareRecorder: Authorized to record user share-seconds.
  - voter & accessHub: Can whitelist reward tokens, update recorder, etc.
  - Rewards can only be distributed in whitelisted tokens.
- **Tracking & Claiming**
  - Tracks:
    - totalRewardByPeriod
    - userShareSecondsByPeriod
    - claimedRewards
  - earnedForVaultShares() returns claimable rewards for user/token/period.
- **Dynamic Whitelisting**
  - Admin can add/remove whitelisted reward tokens using:
    - whitelistReward()
    - removeRewardWhitelist()
  - Only whitelisted tokens can be deposited or notified.

• **Safe Transfers** ○ Uses _safeTransfer and _safeTransferFrom with explicit contract code checks and low-level call protection.	

File 8: Minter.sol
<u>Tokenomics:</u>
• **BASIS:** Used as the denominator for percentage math.
• **MAX_DEVIATION:** Caps changes to 25% per epoch.
• **INITIAL_SUPPLY:** Sets the initial token supply to 500 million TFY.
• **MAX_SUPPLY:** Sets the maximum cap of TFY tokens to 1.5 billion.
 	
<u>Minter Contract – Key Features:</u>
• **One-time Initialization**
  ○ Sets TFY token, xTFY, voter, emissions, and mints initial supply.
• **Start Emissions**
  ○ Begins weekly emissions from epoch 0 and sets timing variables.
• **Weekly Emissions Update**
  ○ Mints emit each new epoch, notify the voter, and trigger xTFY.rebase().
• **Dynamic Emission Control**
  ○ Allows governance to adjust emissions multiplier (max ±25% per epoch).
• **Emissions Cap Enforcement**
  ○ Ensures total TFY supply does not exceed 1.5B tokens.
• **Epoch & Period Tracking**
  ○ Calculates current period and epoch using block.timestamp / 1 weeks.
	YES, This is valid.

• **Access Control**
  ○ kickoff and startEmissions: only operator.
  ○ updateEmissionsMultiplier: only accessHub.
• **Auto-Rebase Trigger**
  ○ Calls xTFY.rebase() after emissions each epoch.	

File 9: PositionOracle.sol
<u>PositionOracle – Key Features:</u>
• **Submit LP position data** to Voter contract.
• **Batch data submission** for multiple pools and epochs.
• **Fallback mode** for emergency use (simulates time-in-range).
• **Adjustable fallback factor** (default 80% of full range).
• **Role-based access**:
  ○ operator: submits data.
  ○ voter: sets roles.
  ○ emergencyAdmin: handles fallback mode.
• **Tracks last update timestamp** for monitoring.
• **Emits events** for transparency and auditability.
	YES, This is valid.

File 10: RevenueToRebaseManager.sol **<u>Tokenomics:</u>** • **COLLECTION_INTERVAL:** Sets a 7-day minimum gap between fee/revenue collections. • **VOTING_DURATION:** Duration of voting window per proposal (4 days). • **MIN_VOTES_REQUIRED:** Minimum 10,000 xTFY needed for proposal to pass. • **MIN_PROPOSAL_THRESHOLD:** Requires at least 1,000 xTFY to submit a proposal. • **PROPOSAL_REPLACEMENT_THRESHOLD:** Needs 5,000 xTFY to replace an active proposal.	**YES, This is valid.**

RevenueToRebaseManager – Key Features:
• **Automated TFY Revenue Processing**
  ○ Collects and distributes TFY revenue weekly:
  → **Burn (deflation)** + **Rebase (staker rewards)**.
• **Default 50/50 Burn/Rebase Split**
  ○ With support for community-governed overrides.
• **Governance Proposals**
  ○ xTFY holders can propose and vote on custom distribution ratios per epoch.
• **Rebase Execution**
  ○ Sends rewards to VoteModule to distribute as external revenue.
• **Burn Execution**
  ○ Burns TFY by transferring to 0x...dEaD.
• **Role-Controlled Access**
  ○ operator: triggers revenue execution.
  ○ accessHub: governance authority.
• **Weekly Collection Interval**
  ○ Enforces a 7-day delay between distributions.
• **Emergency Pause**
  ○ Halts revenue execution in critical situations.
• **Analytics Tracking**
  ○ Tracks burned, rebased, and collected amounts per period and in total.
• **UUPS Upgradeable**
  ○ Supports secure future upgrades with _authorizeUpgrade.	

File 11: Thirdfy.sol **Thirdfy (TFY Token) – Key Features:** • **Mintable ERC20 Token** ○ TFY token with mint() restricted to a designated minter (e.g., Minter contract).	**YES, This is valid.**

• **Burnable**
  ○ Supports burn() and burnFrom() via ERC20Burnable.
• **EIP-2612 Permit Support**
  ○ Gasless approvals via ERC20Permit (signed approvals).
• **Access-Controlled Minting**
  ○ Only the minter address can call mint().
• **Token Metadata**
  ○ Name: TFY
  ○ Symbol: TFY
  ○ Decimals: 18 (default)	

File 12: ThirdfyTimelock.sol
<u>ThirdfyTimelock – Key Features:</u>
• **Time-locked Governance Execution**
  ○ Delays sensitive actions to allow community review (via minDelay).
• **Access Roles**
  ○ proposers: Can queue proposals.
  ○ executors: Can execute approved proposals.
  ○ admin: Initial administrator with setup control.
• **Inherits OpenZeppelin's Timelock Controller**
  ○ Secure, battle-tested governance time-lock implementation.
• **Used with Governor Contracts**
  ○ Typically paired with on-chain voting for decentralized governance.
	YES, This is valid.

File 13: VoteModule.sol **<u>Tokenomics:</u>** • duration: Rebases are streamed over a 30-minute period once initiated. • cooldown: A 12-hour lock period before a user can claim rebase rewards.	**YES, This is valid.**

- PRECISION: Standard decimal precision set to 1e18 for fixed-point calculations.

VoteModule – Key Features:

- **Core Role:**
  - Main staking and reward distribution contract for the TFY Protocol.
- **Staking Mechanics**
  - Users stake xTFY to gain voting power and earn rewards.
  - Tracks balances, total supply, and delegation.

Dual Reward Streams

1. **Rebase Rewards** (from xTFY emissions):
   - Streamed over time via notifyRewardAmount().
2. **External Revenue Rewards** (from protocol revenue):
   - Sent via notifyExternalRevenue() by RevenueToRebaseManager.

Time-Locked Mechanics

- **30-minute streaming duration** (duration) for emissions.
- **12-hour cooldown** before withdrawal or restaking (cooldown).
- unlockTime prevents reward gaming during rebase events.

Security & Safety

- **Immutable staking contract**—user funds can't be seized via governance.
- **ReentrancyGuard** protection on all state-changing logic.
- **Cooldown exemption** managed via AccessHub.
- **Safe initialization** of new external reward users.
- **External rewards can be toggled off** for safety.

Governance Integration ● Automatically calls voter.poke() on deposit/withdraw. ● Supports delegation and admin setting for UX. **Accurate Accounting** ● Per-user tracking for: ○ rewardPerTokenStored. ○ externalRewardPerTokenStored. ● **Claim both rewards** using getReward().	
File 14: ClGaugeFactory.sol **ClGaugeFactory – Key Features:** **Purpose:** ● Factory for creating Concentrated Liquidity (CL) Gauges used in the TFY Protocol. ● Manages gauge metadata and access control. **Core State Variables** ● **voter:** Governance contract allowed to control gauge creation/settings. ● **nfpManager:** (To be used) likely for managing NFT-based LP positions (e.g. Uniswap V3). ● **accessHub:** Access control contract for broader protocol permissions. ● **lastGauge:** Tracks the most recently created gauge. ● **gauges:** Array of all deployed CL gauge addresses.	**YES, This is valid.**
File 15: IchiBribeDistributorFactory.sol **Key Features:** ● **Factory** for deploying IchiBribeDistributor contracts.	**YES, This is valid.**

• **Access-controlled** via AccessHub and Voter. • **Upgradeable** via UUPS pattern (only AccessHub can upgrade). • **Tracks implementation** (first deployed distributor). • **Emits event** on each distributor deployment. • **Deployment safety checks** for zero addresses. • **Admin functions** for updating voter, accessHub, and implementation.	
File 16: IchiVaultGaugeFactory.sol **Key Features:** • **Factory** for deploying IchiVaultGauge contracts. • **Access-controlled** via AccessHub and VoterV4. • **Upgradeable** using **UUPS pattern** (authorized by AccessHub). • **Gauge creation** with paired Ichi Vaults and snapshot recorder. • **Tracks lastGauge** deployed for indexing or automation. • **Events** emitted on gauge creation and admin changes. • **Safety checks** for zero address and access control.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**. Also, these contracts contain owner control, which does not make them fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 1 high, 0 medium, 7 low, and 1 very low-level issues.

We confirm that all issues are fixed in the revised smart contracts code.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack a check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks an event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis - o33.sol

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	Not Detected
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	No
Can Pause Trade?	No
Pause Transfer?	Partial (Transfers aren't explicitly pausable, but deposit() is gated by whileNotLocked)
Max Tax?	No
Is it Anti-whale?	No
Is Anti-bot?	No
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	No
Is it a Proxy?	No
Can Take Ownership?	Yes (Ownership isn't defined via Ownable, but Operator can be transferred via transferOperator() by accessHub)
Hidden Owner?	No
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Code Quality

This audit scope has included 16 smart contract files, 18 interface files, and 4 library files. Smart contracts contain Libraries, Smart contracts, inheritance, and Interfaces. This is a compact and well-written smart contract.

The libraries in o33 Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address, and its properties/methods can be reused many times by other contracts in the o33 Protocol.

The Thirdfy team has provided scenarios and unit test scripts, which have helped to determine the integrity of the code in an automated way.

Code parts are **well** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given an o33 Protocol smart contract code in the form of a file. The MD5 hash of that code is mentioned in the table above.

As mentioned above, code parts are **well-**commented. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as the complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry-standard open-source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

o33.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	whileNotLocked	modifier	Passed	No Issue
3	onlyOperator	modifier	Passed	No Issue
4	onlyAccessHub	modifier	Passed	No Issue
5	submitVotes	external	access only Operator	No Issue
6	compound	external	access only Operator	No Issue
7	claimRebase	external	access only Operator	No Issue
8	claimIncentives	external	access only Operator	No Issue
9	swapIncentiveViaAggregator	external	access only Operator	No Issue
10	rescue	external	access only AccessHub	No Issue
11	unlock	external	access only Operator	No Issue
12	transferOperator	external	access only AccessHub	No Issue
13	whitelistAggregator	external	access only AccessHub	No Issue
14	whitelistRelayer	external	access only AccessHub	No Issue
15	executeMetaTransaction	write	Passed	No Issue
16	totalAssets	read	Passed	No Issue
17	ratio	read	Passed	No Issue
18	getPeriod	read	Passed	No Issue
19	isUnlocked	read	Passed	No Issue
20	isCooldownActive	read	Passed	No Issue
21	_deposit	internal	whileNotLocked	No Issue
22	_withdraw	internal	Passed	No Issue
23	_tryGetAssetDecimals	read	Passed	No Issue
24	decimals	read	Passed	No Issue
25	asset	read	Passed	No Issue
26	totalAssets	read	Passed	No Issue
27	convertToShares	read	Passed	No Issue
28	convertToAssets	read	Passed	No Issue
29	maxDeposit	read	Passed	No Issue
30	maxMint	read	Passed	No Issue
31	maxWithdraw	read	Passed	No Issue

32	maxRedeem	read	Passed	No Issue
33	previewDeposit	read	Passed	No Issue
34	previewMint	read	Passed	No Issue
35	previewWithdraw	read	Passed	No Issue
36	previewRedeem	read	Passed	No Issue
37	deposit	write	Passed	No Issue
38	mint	write	Passed	No Issue
39	withdraw	write	Passed	No Issue
40	redeem	write	Passed	No Issue
41	_convertToShares	internal	Passed	No Issue
42	_convertToAssets	internal	Passed	No Issue
43	_deposit	internal	Passed	No Issue
44	_withdraw	internal	Passed	No Issue
45	_decimalsOffset	internal	Passed	No Issue
46	nonReentrant	modifier	Passed	No Issue
47	_nonReentrantBefore	write	Passed	No Issue
48	_nonReentrantAfter	write	Passed	No Issue
49	_reentrancyGuardEntered	internal	Passed	No Issue

xTFY.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyGovernance	modifier	Passed	No Issue
3	pause	external	access only Governance	No Issue
4	unpause	external	access only Governance	No Issue
5	_update	internal	Passed	No Issue
6	_isExempted	internal	Passed	No Issue
7	convertEmissionsToken	external	whenNotPaused	No Issue
8	rebase	external	whenNotPaused	No Issue
9	emergencyRebase	external	access only Governance	No Issue
10	exit	external	Passed	No Issue
11	createVest	external	Passed	No Issue
12	exitVest	external	Passed	No Issue
13	operatorRedeem	external	access only Governance	No Issue
14	rescueTrappedTokens	external	access only Governance	No Issue
15	migrateOperator	external	access only Governance	No Issue
16	setExemption	external	access only Governance	No Issue

17	setExemptionTo	external	access only Governance	No Issue
18	getBalanceResiding	read	Passed	No Issue
19	usersTotalVests	read	Passed	No Issue
20	getVestInfo	read	Passed	No Issue
21	isExempt	external	Passed	No Issue
22	tfy	external	Passed	No Issue
23	name	read	Passed	No Issue
24	symbol	read	Passed	No Issue
25	decimals	read	Passed	No Issue
26	totalSupply	read	Passed	No Issue
27	balanceOf	read	Passed	No Issue
28	transfer	write	Passed	No Issue
29	allowance	read	Passed	No Issue
30	approve	write	Passed	No Issue
31	transferFrom	write	Passed	No Issue
32	_transfer	internal	Passed	No Issue
33	_update	internal	Passed	No Issue
34	_mint	internal	Passed	No Issue
35	_burn	internal	Passed	No Issue
36	_approve	internal	Passed	No Issue
37	_approve	internal	Passed	No Issue
38	_spendAllowance	internal	Passed	No Issue
39	whenNotPaused	modifier	Passed	No Issue
40	whenPaused	modifier	Passed	No Issue
41	paused	read	Passed	No Issue
42	_requireNotPaused	internal	Passed	No Issue
43	_requirePaused	internal	Passed	No Issue
44	_pause	internal	Passed	No Issue
45	_unpause	internal	Passed	No Issue

VoterV4.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyGovernance	modifier	Passed	No Issue
3	initialize	external	initializer	No Issue
4	_authorizeUpgrade	internal	access only Governance	No Issue
5	setFeeCollector	external	Passed	No Issue
6	setIchiVaultGaugeFactory	external	access only Governance	No Issue
7	setIchiBribeDistributorFactory	external	access only Governance	No Issue
8	setGlobalRatio	external	access only Governance	No Issue

9	setLauncherPlugin	external	access only Governance	No Issue
10	reset	external	Passed	No Issue
11	_reset	internal	Passed	No Issue
12	poke	external	Passed	No Issue
13	vote	external	Passed	No Issue
14	_vote	internal	Passed	No Issue
15	_distribute	internal	Passed	No Issue
16	getVotes	external	Passed	No Issue
17	setGovernor	external	access only Governance	No Issue
18	whitelist	write	Passed	No Issue
19	revokeWhitelist	write	Passed	No Issue
20	killGauge	write	access only Governance	No Issue
21	reviveGauge	write	access only Governance	No Issue
22	stuckEmissionsRecovery	external	access only Governance	No Issue
23	whitelistGaugeRewards	external	access only Governance	No Issue
24	removeGaugeRewardWhitelist	external	access only Governance	No Issue
25	removeFeeDistributorReward	external	Passed	No Issue
26	setMainTickSpacing	external	Passed	No Issue
27	getPeriod	read	Passed	No Issue
28	createCLGauge	external	access only Governance	No Issue
29	_createGaugeForCLPool	internal	Passed	No Issue
30	claimClGaugeRewards	external	Passed	Fixed
31	claimIncentives	external	Passed	No Issue
32	claimRewards	external	Passed	Fixed
33	notifyRewardAmount	external	Passed	No Issue
34	distribute	write	nonReentrant	No Issue
35	distributeForPeriod	write	nonReentrant	No Issue
36	distributeAll	external	Passed	No Issue
37	batchDistributeByIndex	external	Passed	No Issue
38	getAllGauges	external	Passed	No Issue
39	getAllFeeDistributors	external	Passed	No Issue
40	isGauge	external	Passed	No Issue
41	isFeeDistributor	external	Passed	No Issue
42	_claimablePerPeriod	internal	Passed	No Issue
43	withdrawFromCommunityVault	external	access only Governance	No Issue
44	withdrawMultipleFromCommunityVault	external	access only Governance	No Issue
45	recordPositionsTimeInRange	external	Passed	No Issue

46	setOracleApproval	external	access only Governance	No Issue
47	setPositionOracle	external	access only Governance	No Issue
48	setOracleOperator	external	Passed	No Issue
49	createIchiVaultGauge	external	access only Governance	No Issue
50	setShareRecorder	external	access only Governance	No Issue
51	initializer	modifier	Passed	No Issue
52	reinitializer	modifier	Passed	No Issue
53	onlyInitializing	modifier	Passed	No Issue
54	_checkInitializing	internal	Passed	No Issue
55	_disableInitializers	internal	Passed	No Issue
56	_getInitializedVersion	internal	Passed	No Issue
57	_isInitializing	internal	Passed	No Issue
58	_initializableStorageSlot	internal	Passed	No Issue
59	_getInitializableStorage	write	Passed	No Issue
60	_getReentrancyGuardStorage	write	Passed	No Issue
61	__ReentrancyGuard_init	internal	access only Initializing	No Issue
62	__ReentrancyGuard_init_unchained	internal	access only Initializing	No Issue
63	nonReentrant	modifier	Passed	No Issue
64	_nonReentrantBefore	write	Passed	No Issue
65	_nonReentrantAfter	write	Passed	No Issue
66	_reentrancyGuardEntered	internal	Passed	No Issue
67	onlyProxy	modifier	Passed	No Issue
68	notDelegated	modifier	Passed	No Issue
69	__UUPSUpgradeable_init	internal	access only Initializing	No Issue
70	__UUPSUpgradeable_init_unchained	internal	access only Initializing	No Issue
71	proxiableUUID	external	Passed	No Issue
72	upgradeToAndCall	write	access only Proxy	No Issue
73	_checkProxy	internal	Passed	No Issue
74	_checkNotDelegated	internal	Passed	No Issue
75	_authorizeUpgrade	internal	Passed	No Issue
76	_upgradeToAndCallUUPS	write	Passed	No Issue

AccessHub.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	timelocked	modifier	Passed	No Issue
3	initialize	external	initializer	No Issue

4	_authorizeUpgrade	internal	access by default admin	No Issue
5	reinit	external	timelocked	No Issue
6	initializeVoter	external	timelocked	No Issue
7	execute	external	timelocked	No Issue
8	setNewTimelock	external	timelocked	Fixed
9	setAuthorizedTarget	external	timelocked	No Issue
10	isAuthorizedTarget	external	Passed	No Issue
11	setNewGovernorInVoter	external	access by protocol operator	No Issue
12	governanceWhitelist	external	access by protocol operator	No Issue
13	kickInactive	external	access by default admin	No Issue
14	setXTFY	external	timelocked	No Issue
15	setO33	external	timelocked	No Issue
16	transferWhitelistInxTFY	external	access by protocol operator	No Issue
17	togglexTFYGovernance	external	access by protocol operator	No Issue
18	operatorRedeemxTFY	external	access by protocol operator	No Issue
19	migrateOperator	external	access by protocol operator	No Issue
20	rescueTrappedTokens	external	access by protocol operator	No Issue
21	transferOperatorIno33	external	access by protocol operator	No Issue
22	setEmissionsMultiplierInMinter	external	access by protocol operator	No Issue
23	setCooldownExemption	external	timelocked	Fixed
24	setNewRebaseStreamingDuration	external	timelocked	No Issue
25	setNewVoteModuleCooldown	external	timelocked	No Issue
26	_setAuthorizedTarget	internal	Passed	No Issue
27	onlyProxy	modifier	Passed	No Issue
28	notDelegated	modifier	Passed	No Issue
29	__UUPSUpgradeable_init	internal	access only Initializing	No Issue
30	__UUPSUpgradeable_init_unchained	internal	access only Initializing	No Issue
31	proxiableUUID	external	Passed	No Issue
32	upgradeToAndCall	write	access only Proxy	No Issue
33	_checkProxy	internal	Passed	No Issue
34	_checkNotDelegated	internal	Passed	No Issue
35	_authorizeUpgrade	internal	Passed	No Issue
36	_upgradeToAndCallUUPS	write	Passed	No Issue

37	_getAccessControlEnumerable Storage	write	Passed	No Issue
38	__AccessControlEnumerable_in it	internal	access only Initializing	No Issue
39	__AccessControlEnumerable_in it_unchained	internal	access only Initializing	No Issue
40	supportsInterface	read	Passed	No Issue
41	getRoleMember	read	Passed	No Issue
42	getRoleMemberCount	read	Passed	No Issue
43	getRoleMembers	read	Passed	No Issue
44	_grantRole	internal	Passed	No Issue
45	_revokeRole	internal	Passed	No Issue
46	initializer	modifier	Passed	No Issue
47	reinitializer	modifier	Passed	No Issue
48	onlyInitializing	modifier	Passed	No Issue
49	_checkInitializing	internal	Passed	No Issue
50	_disableInitializers	internal	Passed	No Issue
51	_getInitializedVersion	internal	Passed	No Issue
52	_isInitializing	internal	Passed	No Issue
53	_initializableStorageSlot	internal	Passed	No Issue
54	_getInitializableStorage	write	Passed	No Issue
55	onlyProxy	modifier	Passed	No Issue
56	notDelegated	modifier	Passed	No Issue
57	__UUPSUpgradeable_init	internal	access only Initializing	No Issue
58	__UUPSUpgradeable_init_unch ained	internal	access only Initializing	No Issue
59	proxiableUUID	external	Passed	No Issue
60	upgradeToAndCall	write	access only Proxy	No Issue
61	_checkProxy	internal	Passed	No Issue
62	_checkNotDelegated	internal	Passed	No Issue
63	_authorizeUpgrade	internal	Passed	No Issue
64	_upgradeToAndCallUUPS	write	Passed	No Issue

FeeCollector.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyTreasury	modifier	Passed	No Issue
3	onlyVoter	modifier	Passed	No Issue
4	setTreasury	external	access only Treasury	No Issue
5	setTreasuryFees	external	access only Treasury	No Issue
6	setFeeDistributor	external	access only Voter	No Issue
7	safeTransferWithLogging	internal	Passed	No Issue
8	collectProtocolFees	external	Passed	No Issue
9	hasWithdrawerRole	read	Passed	No Issue

10	withdrawFromCommunityVault	external	access only Voter	No Issue
11	withdrawMultipleFromCommunityVault	external	access only Voter	No Issue

IchiBribeDistributor.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	depositBribe	external	Passed	No Issue
3	claimBribes	external	nonReentrant	No Issue
4	earned	read	Passed	No Issue
5	_depositVoteWeight	external	Passed	No Issue
6	getCurrentPeriod	read	Passed	No Issue
7	nonReentrant	modifier	Passed	No Issue
8	_nonReentrantBefore	write	Passed	No Issue
9	_nonReentrantAfter	write	Passed	No Issue
10	_reentrancyGuardEntered	internal	Passed	No Issue

IchiVaultGauge.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	ReentrancyGuard	external	Passed	No Issue
3	earnedForVaultShares	read	Passed	No Issue
4	claimRewardsForPeriod	external	nonReentrant	No Issue
5	setShareRecorder	external	Passed	No Issue
6	notifyVaultRewardAmount	external	Passed	No Issue
7	whitelistReward	external	Passed	No Issue
8	removeRewardWhitelist	external	Passed	No Issue
9	depositExternalLPReward	external	Passed	No Issue
10	rewardsList	external	Passed	No Issue
11	rewardsListLength	external	Passed	No Issue
12	isWhitelisted	read	Passed	No Issue
13	getCurrentPeriod	read	Passed	No Issue
14	_safeTransfer	internal	Passed	No Issue
15	_safeTransferFrom	internal	Passed	No Issue
16	nonReentrant	modifier	Passed	No Issue
17	_nonReentrantBefore	write	Passed	No Issue
18	_nonReentrantAfter	write	Passed	No Issue
19	_reentrancyGuardEntered	internal	Passed	No Issue

Minter.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyGovernance	modifier	Passed	No Issue
3	kickoff	external	Passed	No Issue
4	updatePeriod	external	Passed	No Issue
5	startEmissions	external	Passed	Fixed
6	updateEmissionsMultiplier	external	access only Governance	No Issue
7	calculateWeeklyEmissions	read	Passed	No Issue
8	getPeriod	read	Passed	No Issue
9	getEpoch	read	Passed	No Issue

PositionOracle.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyVoter	modifier	Passed	No Issue
3	onlyOperator	modifier	Passed	No Issue
4	onlyOperator	modifier	Passed	No Issue
5	setOperator	external	access only Governance	No Issue
6	setEmergencyAdmin	external	access only Governance	No Issue
7	setFallbackMode	external	access only Emergency Admin	No Issue
8	setFallbackFactor	external	access only Emergency Admin	No Issue
9	submitPositionData	external	access only Operator	No Issue
10	_submitPositionData	internal	Passed	No Issue
11	batchSubmitPositionData	external	access only Operator	No Issue
12	onlyOwner	modifier	Passed	No Issue
13	owner	read	Passed	No Issue
14	_checkOwner	internal	Passed	No Issue
15	renounceOwnership	write	access only Owner	No Issue
16	transferOwnership	write	access only Owner	No Issue
17	_transferOwnership	internal	Passed	No Issue

Thirdfy.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue

2	mint	write	Passed	No Issue
3	name	read	Passed	No Issue
4	symbol	read	Passed	No Issue
5	decimals	read	Passed	No Issue
6	totalSupply	read	Passed	No Issue
7	balanceOf	read	Passed	No Issue
8	transfer	write	Passed	No Issue
9	allowance	read	Passed	No Issue
10	approve	write	Passed	No Issue
11	transferFrom	write	Passed	No Issue
12	_transfer	internal	Passed	No Issue
13	_update	internal	Passed	No Issue
14	_mint	internal	Passed	No Issue
15	_burn	internal	Passed	No Issue
16	_approve	internal	Passed	No Issue
17	_approve	internal	Passed	No Issue
18	_spendAllowance	internal	Passed	No Issue
19	burn	write	Passed	No Issue
20	burnFrom	write	Passed	No Issue
21	permit	write	Passed	No Issue
22	nonces	read	Passed	No Issue
23	DOMAIN_SEPARATOR	external	Passed	No Issue

ThirdfyTimelock

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyRoleOrOpenRole	modifier	Passed	No Issue
3	receive	external	Passed	No Issue
4	supportsInterface	read	Passed	No Issue
5	isOperation	read	Passed	No Issue
6	isOperationPending	read	Passed	No Issue
7	isOperationReady	read	Passed	No Issue
8	isOperationDone	read	Passed	No Issue
9	getTimestamp	read	Passed	No Issue
10	getOperationState	read	Passed	No Issue
11	getMinDelay	read	Passed	No Issue
12	hashOperation	write	Passed	No Issue
13	hashOperationBatch	write	Passed	No Issue
14	schedule	write	access by Proposer role	No Issue
15	scheduleBatch	write	access by Proposer role	No Issue
16	_schedule	write	Passed	No Issue

17	cancel	write	access by Canceller role	No Issue
18	execute	write	access by executor role	No Issue
19	executeBatch	write	access by executor role	No Issue
20	_execute	internal	Passed	No Issue
21	_beforeCall	read	Passed	No Issue
22	_afterCall	write	Passed	No Issue
23	updateDelay	external	Passed	No Issue
24	_encodeStateBitmap	internal	Passed	No Issue

VoteModule.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyAccessHub	modifier	Passed	No Issue
3	onlyRevenueManager	modifier	Passed	No Issue
4	updateExternalReward	modifier	Passed	No Issue
5	initialize	external	initializer	No Issue
6	updateReward	modifier	Passed	No Issue
7	depositAll	external	Passed	No Issue
8	deposit	write	update Reward	No Issue
9	withdrawAll	external	Passed	No Issue
10	withdraw	write	update Reward	No Issue
11	notifyRewardAmount	external	update Reward	No Issue
12	setCooldownExemption	external	access by AccessHub	No Issue
13	setRevenueManager	external	access by AccessHub	No Issue
14	notifyExternalRevenue	external	access by Revenue Manager	No Issue
15	setNewDuration	external	access by AccessHub	No Issue
16	setNewCooldown	external	access by AccessHub	No Issue
17	delegate	external	Passed	No Issue
18	setAdmin	external	Passed	No Issue
19	lastTimeRewardApplicable	read	Passed	No Issue
20	earned	read	Passed	No Issue
21	getReward	external	update Reward	No Issue
22	_claim	internal	Passed	No Issue
23	rewardPerToken	read	Passed	No Issue
24	left	read	Passed	No Issue
25	isDelegateFor	external	Passed	No Issue
26	isAdminFor	external	Passed	No Issue
27	getXTFY	external	Passed	No Issue

28	lastTimeExternalRewardApplicable	read	Passed	No Issue
29	externalRewardPerToken	read	Passed	No Issue
30	earnedExternalRevenue	read	Passed	No Issue
31	_claimExternalRewards	internal	Passed	No Issue
32	externalLeft	read	Passed	No Issue
33	emergencyDisableExternalRewards	external	access by AccessHub	No Issue
34	areExternalRewardsEnabled	external	Passed	No Issue
35	nonReentrant	modifier	Passed	No Issue
36	_nonReentrantBefore	write	Passed	No Issue
37	_nonReentrantAfter	write	Passed	No Issue
38	_reentrancyGuardEntered	internal	Passed	No Issue
39	initializer	modifier	Passed	No Issue
40	reinitializer	modifier	Passed	No Issue
41	onlyInitializing	modifier	Passed	No Issue
42	_checkInitializing	internal	Passed	No Issue
43	_disableInitializers	internal	Passed	No Issue
44	_getInitializedVersion	internal	Passed	No Issue
45	_isInitializing	internal	Passed	No Issue
46	_initializableStorageSlot	internal	Passed	No Issue
47	_getInitializableStorage	write	Passed	No Issue

ClGaugeFactory.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	setVoter	external	Passed	No Issue
3	setNFPManager	external	Passed	No Issue
4	setAccessHub	external	Passed	No Issue
5	createGauge	external	Passed	No Issue
6	gaugesLength	external	Passed	No Issue

IchiBribeDistributorFactory.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyAccessHub	modifier	Passed	No Issue
3	initialize	external	initializer	No Issue
4	setAccessHub	external	access by AccessHub	No Issue
5	setVoter	external	access by AccessHub	No Issue
6	createDistributor	external	Passed	No Issue

7	setImplementation	external	access by AccessHub	No Issue
8	_authorizeUpgrade	internal	access by AccessHub	No Issue
9	initializer	modifier	Passed	No Issue
10	reinitializer	modifier	Passed	No Issue
11	onlyInitializing	modifier	Passed	No Issue
12	_checkInitializing	internal	Passed	No Issue
13	_disableInitializers	internal	Passed	No Issue
14	_getInitializedVersion	internal	Passed	No Issue
15	_isInitializing	internal	Passed	No Issue
16	_initializableStorageSlot	internal	Passed	No Issue
17	_getInitializableStorage	write	Passed	No Issue
18	onlyProxy	modifier	Passed	No Issue
19	notDelegated	modifier	Passed	No Issue
20	__UUPSUpgradeable_init	internal	access only Initializing	No Issue
21	__UUPSUpgradeable_init_unchained	internal	access only Initializing	No Issue
22	proxiableUUID	external	Passed	No Issue
23	upgradeToAndCall	write	access only Proxy	No Issue
24	_checkProxy	internal	Passed	No Issue
25	_checkNotDelegated	internal	Passed	No Issue
26	_authorizeUpgrade	internal	Passed	No Issue
27	_upgradeToAndCallUUPS	write	Passed	No Issue

IchiVaultGaugeFactory.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyVoterOrAccessHub	modifier	Passed	No Issue
3	onlyAccessHub	modifier	Passed	No Issue
4	initialize	external	initializer	No Issue
5	setVoter	external	access by AccessHub	No Issue
6	setAccessHub	external	access by AccessHub	No Issue
7	createVaultGauge	external	access only Voter Or AccessHub	No Issue
8	_authorizeUpgrade	internal	access by AccessHub	No Issue
9	initializer	modifier	Passed	No Issue
10	reinitializer	modifier	Passed	No Issue
11	onlyInitializing	modifier	Passed	No Issue
12	_checkInitializing	internal	Passed	No Issue
13	_disableInitializers	internal	Passed	No Issue

14	_getInitializedVersion	internal	Passed	No Issue
15	_isInitializing	internal	Passed	No Issue
16	_initializableStorageSlot	internal	Passed	No Issue
17	_getInitializableStorage	write	Passed	No Issue
18	onlyProxy	modifier	Passed	No Issue
19	notDelegated	modifier	Passed	No Issue
20	__UUPSUpgradeable_init	internal	access only Initializing	No Issue
21	__UUPSUpgradeable_init_unchained	internal	access only Initializing	No Issue
22	proxiableUUID	external	Passed	No Issue
23	upgradeToAndCall	write	access only Proxy	No Issue
24	_checkProxy	internal	Passed	No Issue
25	_checkNotDelegated	internal	Passed	No Issue
26	_authorizeUpgrade	internal	Passed	No Issue
27	_upgradeToAndCallUUPS	write	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss, etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc., code snippets, which can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

(1) Timelock Can Be Set to Zero Address (Bricking Governance): **AccessHub.sol**

```
/// @notice Set new timelock address
function setNewTimelock(address _timelock) external timelocked {
    require(timelock != _timelock, SAME_ADDRESS());
    timelock = _timelock;
}
```

The setNewTimelock function allows the current timelock to set the timelock address to address(0). If this happens, all timelocked functions become permanently inaccessible, bricking the contract's governance and upgradeability.

Resolution: Add a check:

require(_timelock != address(0), "Zero address not allowed");

to prevent setting the timelock to the zero address.

Status: Fixed

Medium

No medium severity vulnerabilities were found.

Low

(1) No AccessHub Ownership Transfer Event: **AccessHub.sol**

```
/// @notice Set new timelock address
function setNewTimelock(address _timelock) external timelocked {
    require(timelock != _timelock, SAME_ADDRESS());
    timelock = _timelock;
}
```

When the timelock is changed, there is no event emitted to signal the change.

Resolution: Emit an event when the timelock is updated.

Status: Fixed

(2) Silent Failure in claimClGaugeRewards: **RewardClaimers.sol**

```
            try IGauge(_gauges[i]).getRewardForPosition(
                _nfpTokenIds[i][j],
                _tokens[i]
            ) {
                // Success using our interface
            } catch {
                try IGaugeV3(_gauges[i]).getRewardForOwner(
                    _nfpTokenIds[i][j],
                    _tokens[i]
                ) {
                    // Success using legacy interface
                } catch {
                    // Neither interface worked, silently continue
                    // This handles the case where the gauge might not support NFT positions
                }
            }
        }
```

The function uses nested try/catch blocks and silently continues if both interfaces fail. This may make it difficult for users to know if their reward claim was successful or not.

Resolution: Emit an event or return a status to indicate which claims failed.

Status: Fixed

(3) No Input Validation for Nested Arrays: **RewardClaimers.sol**

```
/// @dev function for claiming CL rewards with multiple ownership/access checks
function claimClGaugeRewards(
    address nfpManager,
    address[] calldata _gauges,
    address[][] calldata _tokens,
    uint256[][] calldata _nfpTokenIds
) external {
    for (uint256 i; i < _gauges.length; ++i) {
        for (uint256 j; j < _nfpTokenIds[i].length; ++j) {
            require(
                msg.sender ==
                    INonfungiblePositionManager(nfpManager).ownerOf(
                        _nfpTokenIds[i][j]
                    ) ||
                    msg.sender ==
                    INonfungiblePositionManager(nfpManager).getApproved(
                        _nfpTokenIds[i][j]
                    ) ||
                    INonfungiblePositionManager(nfpManager)
                        .isApprovedForAll(
                            INonfungiblePositionManager(nfpManager).ownerOf(
                                _nfpTokenIds[i][j]
                            ),
                            msg.sender
                        )
            );
```

The function claimClGaugeRewards assumes that _gauges, _tokens, and _nfpTokenIds are all of the same length and that each _tokens[i] and _nfpTokenIds[i] are valid. If the arrays are mismatched, this could cause out-of-bounds errors.

Resolution: Array length check require.

Status: Fixed

(4) No Array Length Check in setCooldownExemption: **AccessHub.sol**

```
/// @notice Set cooldown exemption for addresses
function setCooldownExemption(
    address[] calldata _candidates,
    bool[] calldata _exempt
) external timelocked {
    for (uint256 i; i < _candidates.length; ++i) {
        voteModule.setCooldownExemption(_candidates[i], _exempt[i]);
    }
}
```

The function loops over _candidates and _exempt arrays but does not check that their lengths match, which could cause out-of-bounds errors.

Resolution: require(_candidates.length == _exempt.length, "LENGTH_MISMATCH");

Status: Fixed

(5) No Error Handling for External Calls in claimRewards: **RewardClaimers.sol**

```
/// @dev for claiming a batch of legacy gauge rewards
function claimRewards(
    address[] calldata _gauges,
    address[][] calldata _tokens
) external {
    for (uint256 i; i < _gauges.length; ++i) {
        IGauge(_gauges[i]).getReward(msg.sender, _tokens[i]);
    }
}
```

The function claimRewards calls IGauge(_gauges[i]).getReward(msg.sender, _tokens[i]); in a loop without try/catch or error handling. If any call fails, the entire transaction will revert, potentially preventing users from claiming rewards from other gauges.

Resolution: Wrap the call in a try/catch block to allow the function to continue if one gauge call fails and Emit an event or return a status to indicate which claims failed.

Status: Fixed

(6) No Input Validation for Array Lengths in: **RewardClaimers.sol**

```
/// @dev for claiming a batch of legacy gauge rewards
function claimRewards(
    address[] calldata _gauges,
    address[][] calldata _tokens
) external {
    for (uint256 i; i < _gauges.length; ++i) {
        IGauge(_gauges[i]).getReward(msg.sender, _tokens[i]);
    }
}
```

The function claimRewards assumes that _gauges and _tokens arrays are of the same length but does not check this, which could lead to out-of-bounds errors.

Resolution: require(_gauges.length == _tokens.length, "Array length mismatch");

Status: Fixed

(7) Lack of Event Emission: **Minter.sol**

```
/// @inheritdoc IMinter
function startEmissions() external {
    /// @dev ensure only the operator can start the emissions
    require(msg.sender == operator, IVoter.NOT_AUTHORIZED(msg.sender));
    /// @dev ensure epoch 0 has not started yet
    require(firstPeriod == 0, STARTED());
    /// @dev set the active period to the current
    activePeriod = getPeriod();
    /// @dev set the last update as the last period so emissions can be updated once if needed
    lastMultiplierUpdate = activePeriod - 1;
    /// @dev set the first period to the active period
    firstPeriod = activePeriod;
    /// @dev mints the epoch 0 emissions for manual distribution
    tfy.mint(operator, weeklyEmissions);
}
```

startEmissions does not emit an event for starting emissions. This can make off-chain tracking and auditing more difficult.

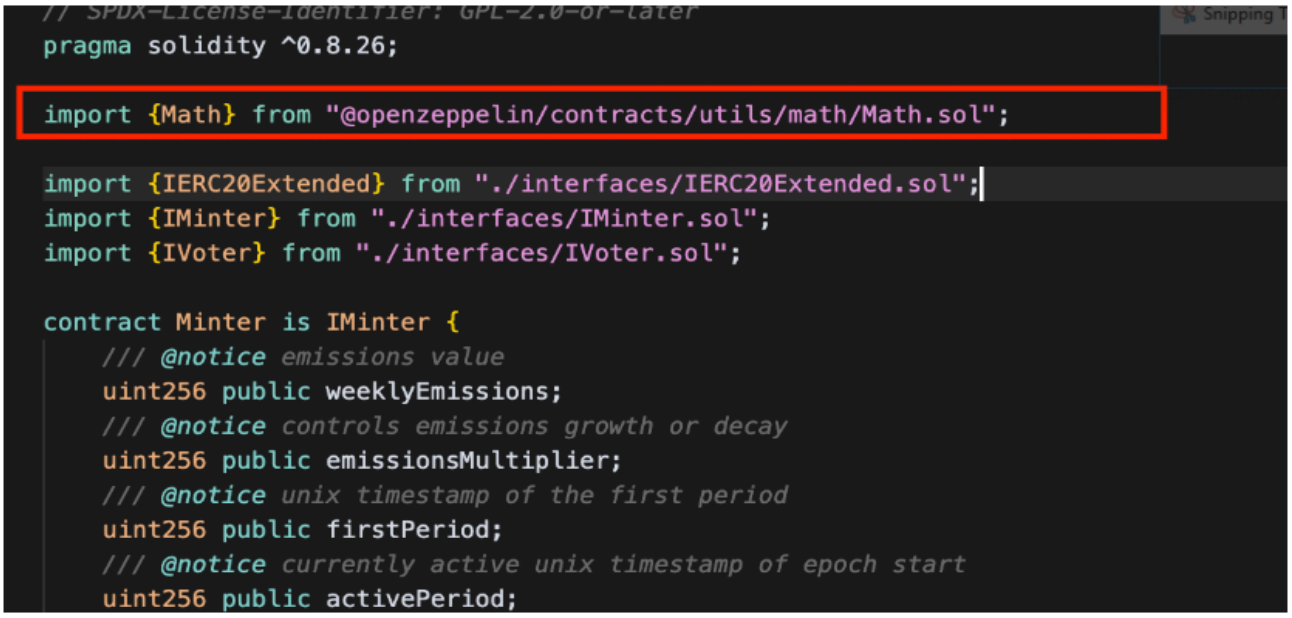
Resolution: Emit events for all significant state changes, especially those that affect emissions or contract configuration.

Status: Fixed

Very Low / Informational / Best practices:

(1) Unnecessary import statement:

Minter.sol



```
// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity ^0.8.26;

import {Math} from "@openzeppelin/contracts/utils/math/Math.sol";

import {IERC20Extended} from "./interfaces/IERC20Extended.sol";
import {IMinter} from "./interfaces/IMinter.sol";
import {IVoter} from "./interfaces/IVoter.sol";

contract Minter is IMinter {
    /// @notice emissions value
    uint256 public weeklyEmissions;
    /// @notice controls emissions growth or decay
    uint256 public emissionsMultiplier;
    /// @notice unix timestamp of the first period
    uint256 public firstPeriod;
    /// @notice currently active unix timestamp of epoch start
    uint256 public activePeriod;
```

The import statement import {Math} from "@openzeppelin/contracts/utils/math/Math.sol"; is present in Minter.sol, but based on a search of the file, the Math library is not actually used anywhere in the contract.

Resolution: can be removed to clean up the code.

Status: Fixed

Centralization

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet private key were compromised, then it would create trouble. The following are Admin functions:

o33.sol

- submitVotes: Casts votes on specified pools with given weights via the Voter contract by the Operator.
- compound: Converts any TFY balance to xTFY and stakes it into the VoteModule to auto-compound rewards by the Operator.

- claimRebase: Claims TFY rebases, converts them to xTFY, and deposits into the VoteModule for compounding by the Operator.
- claimIncentives: Claims incentive rewards from FeeDistributors for voted gauges by the Operator.
- swapIncentiveViaAggregator: Swaps reward tokens (non-TFY) to TFY using a whitelisted aggregator and enforces slippage check by the Operator.
- rescue: Allows emergency token rescue by AccessHub while ensuring xTFY balance/integrity remains unchanged by the AccessHub.
- unlock: Unlocks the current epoch for user deposits and withdrawals if cooldown is complete by the Operator.
- transferOperator: Updates the operator address via the AccessHub contract.
- whitelistAggregator: Adds or removes a token swap aggregator from the whitelist via the AccessHub contract..
- whitelistRelayer: Adds or removes a relayer from the meta-transaction whitelist via the AccessHub contract.

xTFY.sol

- pause: Pauses all state-changing functions by the governance.
- unpause: Unpauses contract operations by the governance.
- rebase: Called by Minter to distribute pendingRebase to VoteModule once per epoch.
- emergencyRebase: Allows governance to manually trigger a rebase, skipping epoch check.
- operatorRedeem: Burns `xTFY` from operator and returns TFY to it by the governance.
- rescueTrappedTokens: Allows governance to recover non-TFY tokens by the governance.
- migrateOperator: Updates the `operator` address by the governance.
- setExemption: Sets exemption-from-transfer rules for senders by the governance.
- setExemptionTo: Sets exemption-to-transfer rules for receivers by the governance.

AccessHub.sol

- _authorizeUpgrade: Restricts upgrades to `DEFAULT_ADMIN_ROLE`.
- reinit: Updates protocol contracts and re-authorizes them by timelock.

- initializeVoter: Initializes the `Voter` contract with required addresses by timelock.
- execute: Executes arbitrary function call on a whitelisted contract by timelock.
- setNewTimelock: Updates the timelock address by timelock.
- setAuthorizedTarget: Adds or removes a contract from the execute whitelist by timelock.
- setNewGovernorInVoter: Updates governor address in the `Voter` contract by the protocol operator-only.
- governanceWhitelist: Whitelists or removes governance tokens by the protocol operator-only.
- kickInactive: Resets inactive voters with no upcoming votes by the admin-only.
- setXTFY: Updates `xTFY` contract address and re-authorizes by timelock.
- setO33: Updates `o33` contract address and re-authorizes by timelock.
- transferWhitelistInxTFY: Updates transfer exemption list in `xTFY` by the protocol operator-only.
- togglexTFYGovernance: Pauses/unpauses `xTFY` contract by the protocol operator-only.
- operatorRedeemxTFY: Redeems `xTFY` and sends TFY to operator by the protocol operator-only.
- migrateOperator: Transfers operator role in `xTFY by the protocol operator-only.
- rescueTrappedTokens: Recovers stuck tokens from `xTFY` by the protocol operator-only.
- transferOperatorIno33: Transfers operator role in `o33` contract by the protocol operator-only.
- setEmissionsMultiplierInMinter: Updates emissions multiplier in `Minter` by the protocol operator-only.
- setCooldownExemption: Adds/removes cooldown exemptions by timelock.
- setNewRebaseStreamingDuration: Updates the rebase streaming duration by timelock.
- setNewVoteModuleCooldown: Updates cooldown period for vote module by timelock.

FeeCollector.sol

- setTreasury: Updates the treasury address; only callable by current treasury.
- setTreasuryFees: Sets the treasury fee percentage by only treasury.

- setFeeDistributor: Sets the fee distributor address; only callable by the voter.
- withdrawFromCommunityVault: Withdraws a specific token from the AlgebraCommunityVault; optionally sends to fee distributor.
- withdrawMultipleFromCommunityVault: Withdraws multiple tokens from the vault in a batch; optionally sends to fee distributor.

IchiVaultGauge.sol

- setShareRecorder: Allows the governor or accessHub to update the address that can submit share-seconds.
- whitelistReward: Authorizes a token to be used as a reward by adding it to the whitelist.
- removeRewardWhitelist: Removes a previously whitelisted reward token.

Minter.sol

- updateEmissionsMultiplier: Governance-controlled function to adjust emissions multiplier within a 25% bound.

PositionOracle.sol

- setOperator: Voter-only function to assign a new oracle operator address.
- setEmergencyAdmin: Voter-only function to assign a new emergency admin.
- setFallbackMode: Emergency admin can enable or disable fallback mode.
- setFallbackFactor: Emergency admin can set the fallback time-in-range percentage (max 100).
- submitPositionData: Operator-only function to submit time-in-range and liquidity data to the voter for a single pool.
- batchSubmitPositionData: Operator-only function to submit position data across multiple pools in a single transaction.

RevenueToRebaseManager.sol

- _authorizeUpgrade: Authorizes UUPS upgrade; restricted to governance (`AccessHub`).
- setOperator: Governance can update the operator responsible for triggering weekly revenue execution.

- governanceCancelProposal: Governance can cancel a proposal in case of malicious activity or emergency.
- emergencyPause: Pauses contract operations in emergency situations.
- emergencyUnpause: Resumes contract operations after an emergency pause.
- emergencyRecoverTokens: Allows governance to recover ERC20 tokens during an emergency pause.

Thirdfy.sol

- mint: Only the minter address can mint new tokens.

VoteModule.sol

- setCooldownExemption: Sets cooldown exemption for a user by AccessHub-Only.
- setRevenueManager: Sets the address allowed to stream external rewards by AccessHub-Only.
- notifyExternalRevenue: Streams external rewards from `RevenueManager` Only callable by authorized RevenueToRebaseManager.
- setNewDuration: Updates reward distribution duration by AccessHub-Only.
- setNewCooldown: Updates the cooldown period by AccessHub-Only.
- emergencyDisableExternalRewards: Emergency function to disable external rewards for safety Only callable by AccessHub governance.

:

VoterV4.sol

- setFeeCollector: Sets the fee collector address by the contract owner or accessHub or governor.
- setIchiVaultGaugeFactory: Updates the Ichi Vault Gauge Factory address via governance.
- setIchiBribeDistributorFactory: Updates the Ichi Bribe Distributor Factory address via governance.
- setGlobalRatio: Sets the global xTFY emissions ratio; Can only be called by governance.
- setLauncherPlugin: Updates the address of the launcher plugin via governance.
- setGovernor: Updates the `governor` address and emits an event by current governors.
- whitelist: Marks a token as whitelisted; callable by deployer or governance.

- revokeWhitelist: Removes a token from the whitelist by the contract owner or accessHub or governor.
- killGauge: Deactivates a gauge, distributes remaining emissions to `governor`, and marks it as killed by the governor.
- reviveGauge: Reactivates a previously killed gauge and updates its distribution state by the governor.
- stuckEmissionsRecovery: Allows governance to recover unclaimed emissions from a dead gauge for a specific period.
- whitelistGaugeRewards: Whitelists a reward token for a specific gauge based on its type by the governor.
- removeGaugeRewardWhitelist: Removes a whitelisted reward token from a gauge by the governor.
- createCLGauge: Creates a new CL gauge for an Algebra pool, verifies token whitelist and feeCollector by the governor.
- notifyRewardAmount: Called by `Minter` to fund this contract with TFY and mark reward distribution for the current period.
- withdrawFromCommunityVault: Withdraws specified amount of a token from the AlgebraCommunityVault, optionally routing to the fee distributor by the governor.
- withdrawMultipleFromCommunityVault: Withdraws multiple token amounts from the vault in a single transaction by the governor.
- recordPositionsTimeInRange: Allow calls from governor, approved oracles, or the position oracle can record NFT LP position time-in-range data for a gauge.
- setOracleApproval: Grants or revokes permission for an address to record LP position data by the governor.
- setPositionOracle: Set the position oracle address by the governor.
- setOracleOperator: Sets the operator address in the PositionOracle by the contract owner or accessHub or governor.
- createIchiVaultGauge: Creates a new IchiVaultGauge instance for a pair of Ichi vaults by the governor.
- setShareRecorder: Sets the share recorder address for a specific IchiVaultGauge by the governor.

ClGaugeFactory.sol

- setVoter: Sets the `voter` address; callable only once or by current voter/accessHub.
- setNFPManager: Sets the `nfpManager` address; restricted to `voter` or `accessHub`.
- setAccessHub: Sets the `accessHub`; can be initialized once or updated by `voter` or current `accessHub`.

IchiBribeDistributorFactory.sol

- setAccessHub: Updates the `AccessHub` address; callable only by the current `AccessHub`.
- setVoter: Sets the Voter contract address; callable only by `AccessHub`.
- createDistributor: Deploys a new `IchiBribeDistributor` contract and emits `DistributorCreated`; callable by `AccessHub` or `Voter`.
- setImplementation: Updates the tracked implementation address; callable only by `AccessHub`.
- _authorizeUpgrade: UUPS upgrade hook that restricts upgrades to only `AccessHub`.

IchiVaultGaugeFactory.sol

- setVoter: Updates the authorized `VoterV4` address; callable only by `AccessHub`.
- setAccessHub: Updates the `AccessHub` address; callable only by current `AccessHub`.
- createVaultGauge: Deploys a new `IchiVaultGauge` with the specified vault pair and initial share recorder; callable by `Voter` or `AccessHub`.
- Deploys a new `IchiVaultGauge` with the specified vault pair and initial share recorder; callable by `Voter` or `AccessHub`.
- _authorizeUpgrade: Authorize contract upgrades - only AccessHub can authorize upgrades.

Conclusion

We were given a contract code in the form of a file. And we have used all possible tests based on the given objects as files. We observed 1 high, 7 low, and 1 very low / Informational issues in the smart contracts. We confirm that all issues are fixed in the revised smart contracts code. **So, the smart contracts are ready for the mainnet deployment.**

Since possible test cases can be unlimited for such a smart contract protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of the functionality of the software under review. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract by the best industry practices at the date of this report, about: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best to conduct the analysis and produce this report, it is important to note that you should not rely on this report alone. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

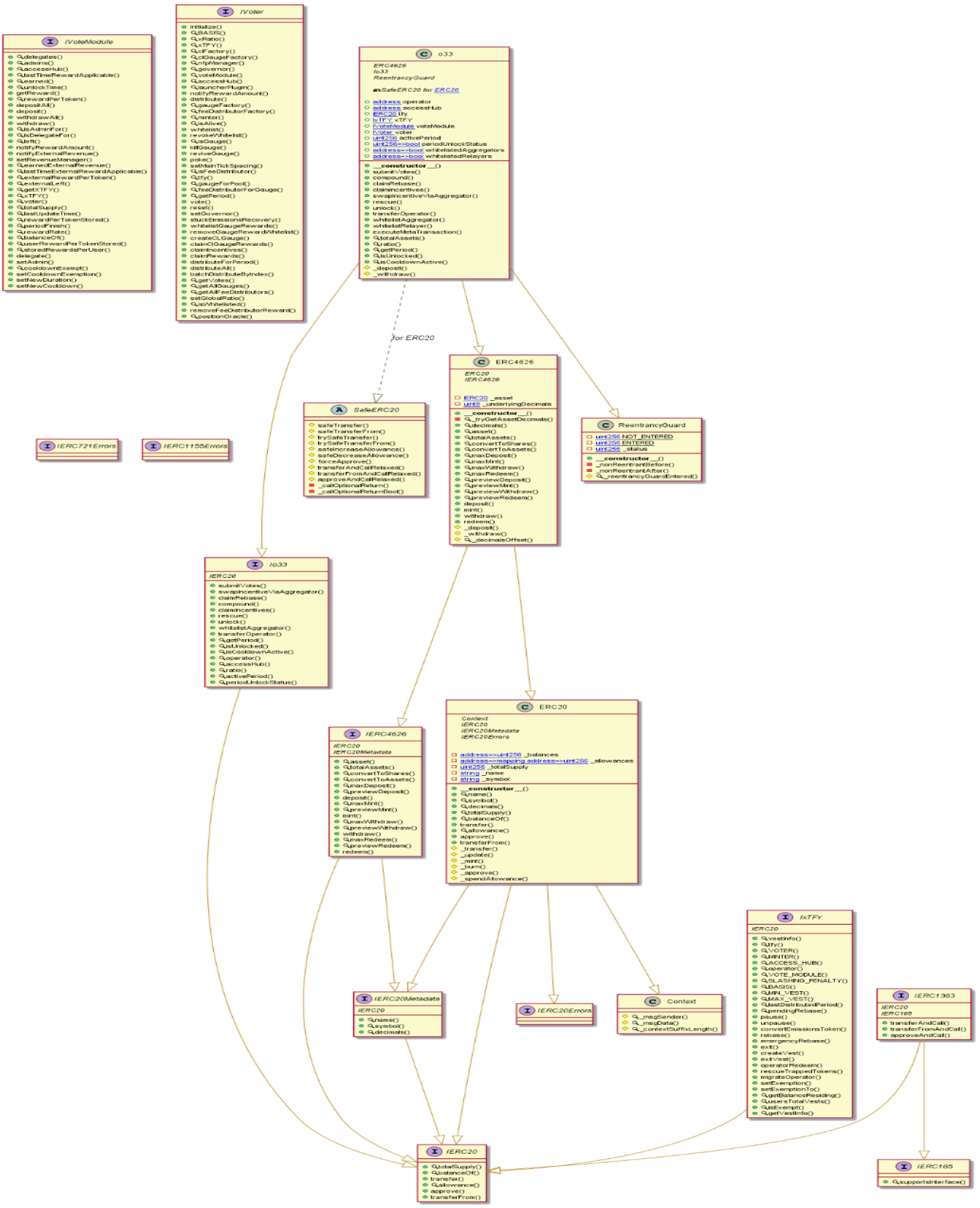
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

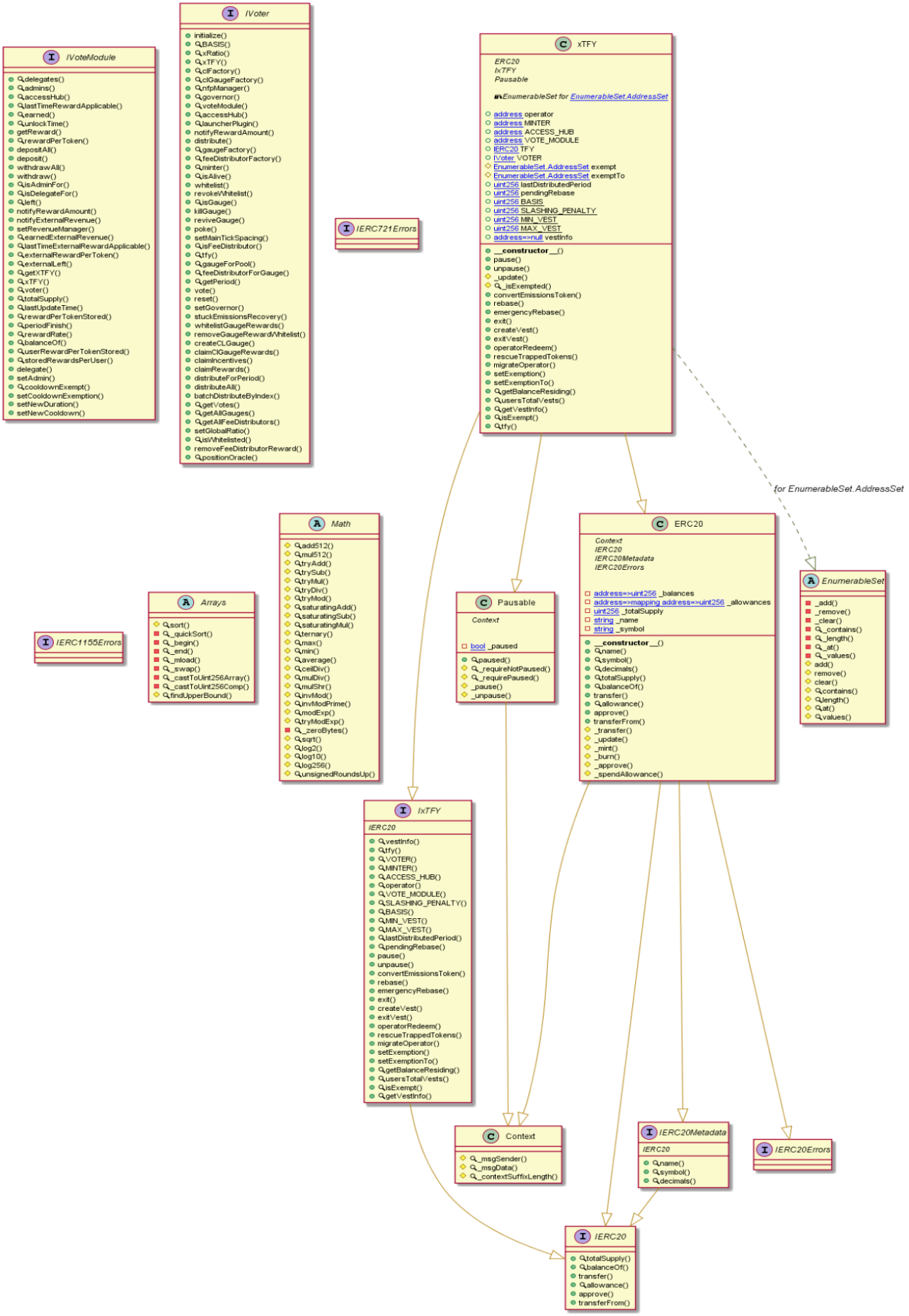
Appendix

Code Flow Diagram - TFY Liquid Staking (o33) Protocol

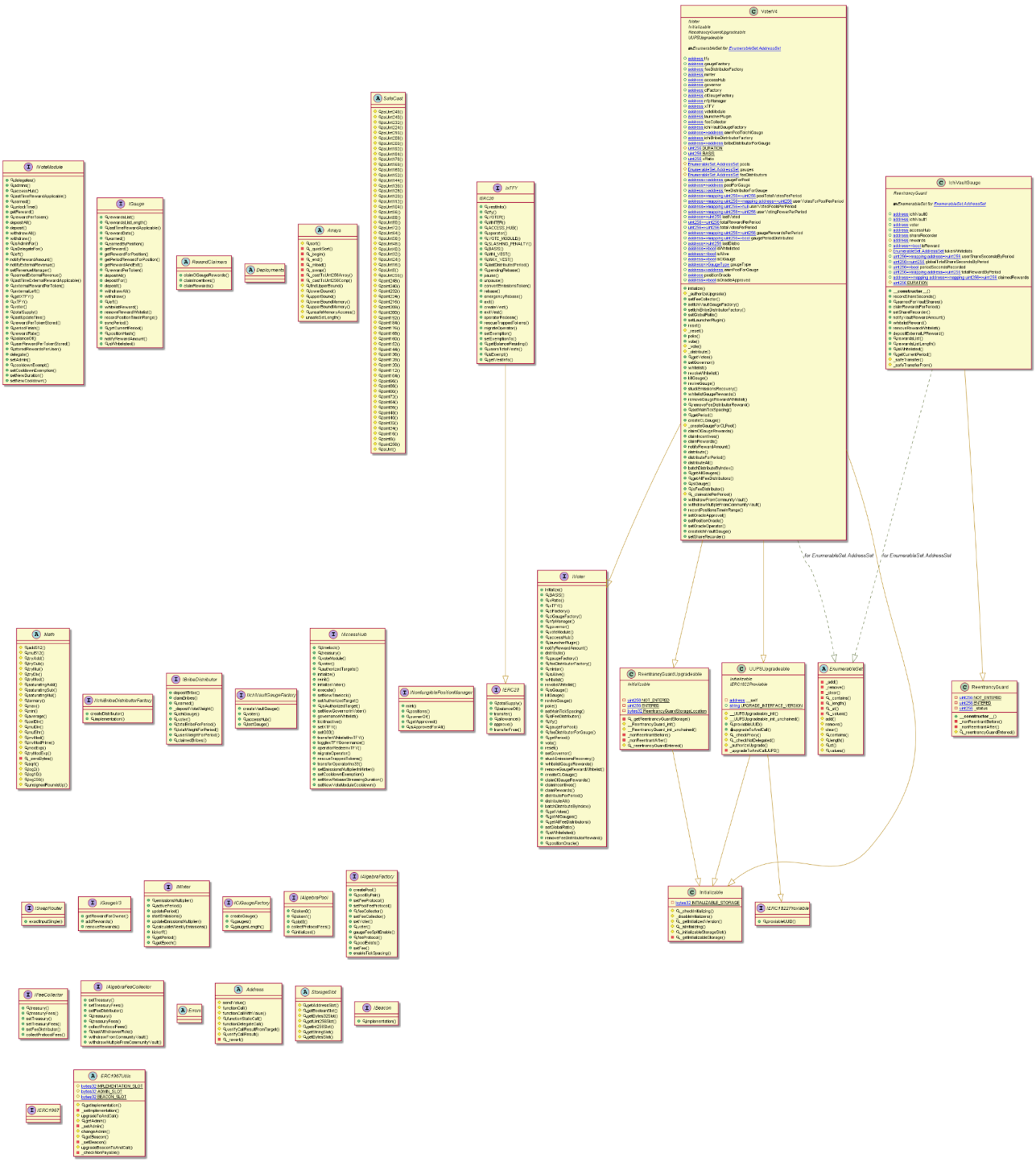
o33 Diagram



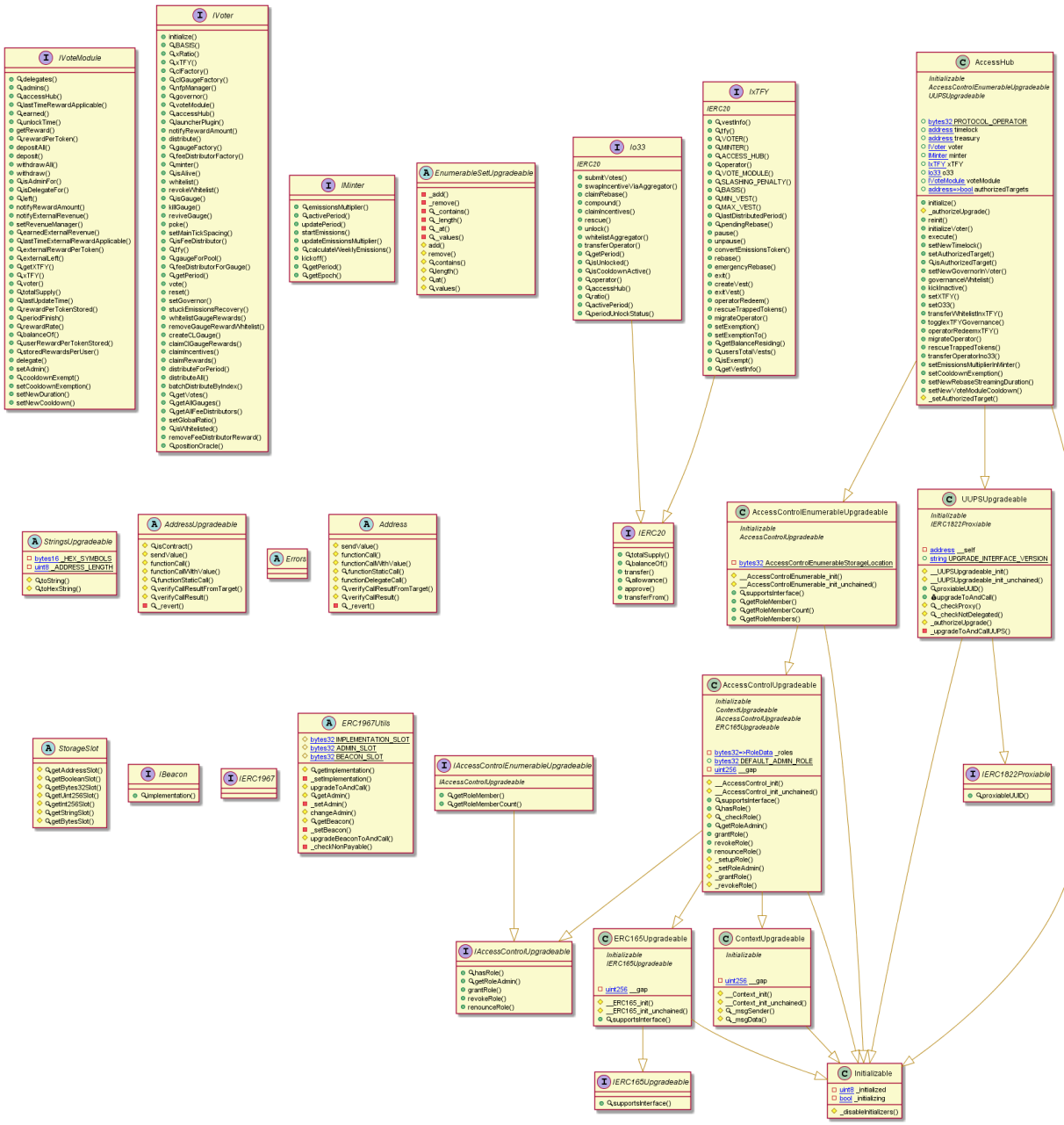
xTFY Diagram



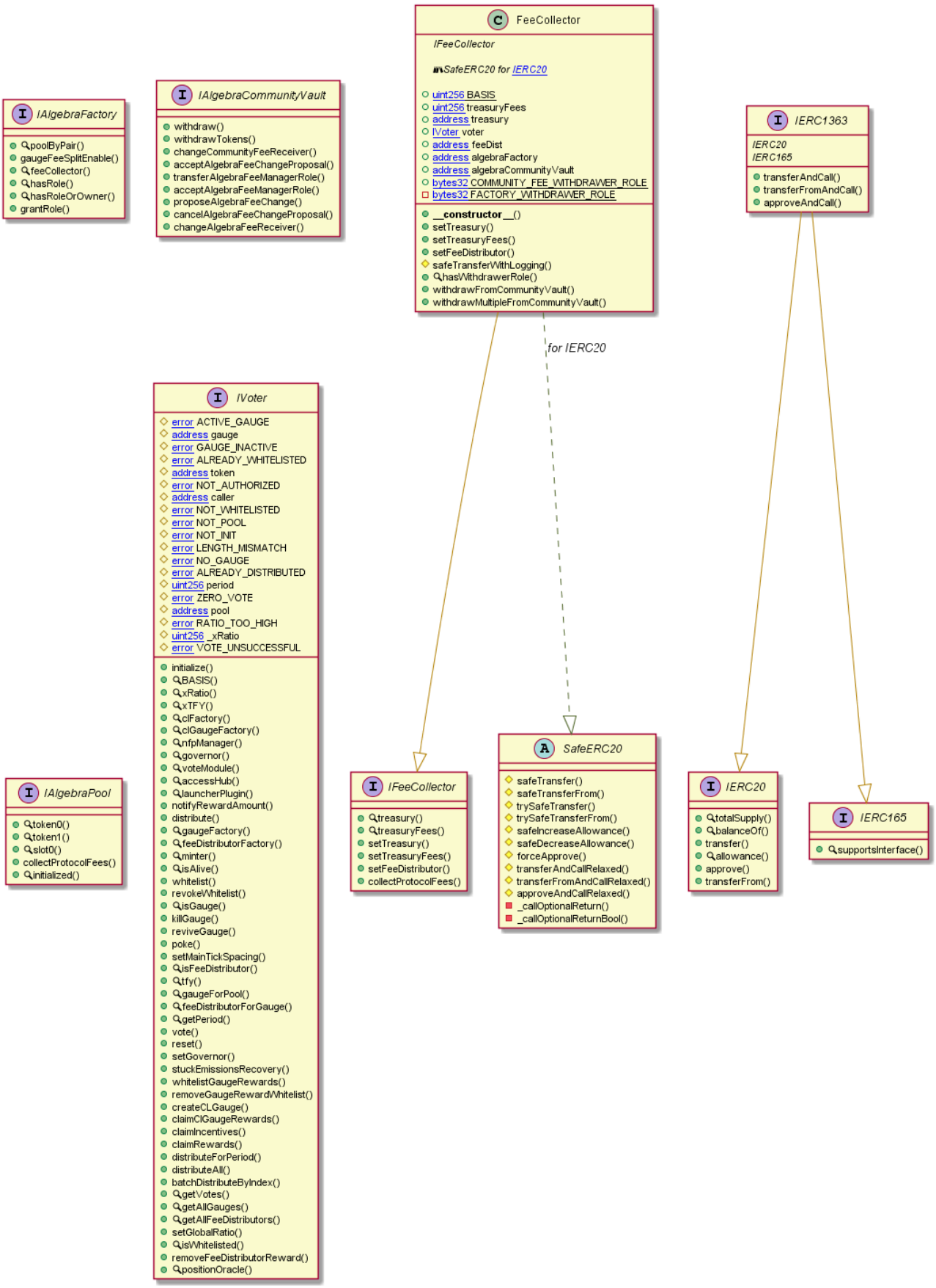
VoterV4 Diagram



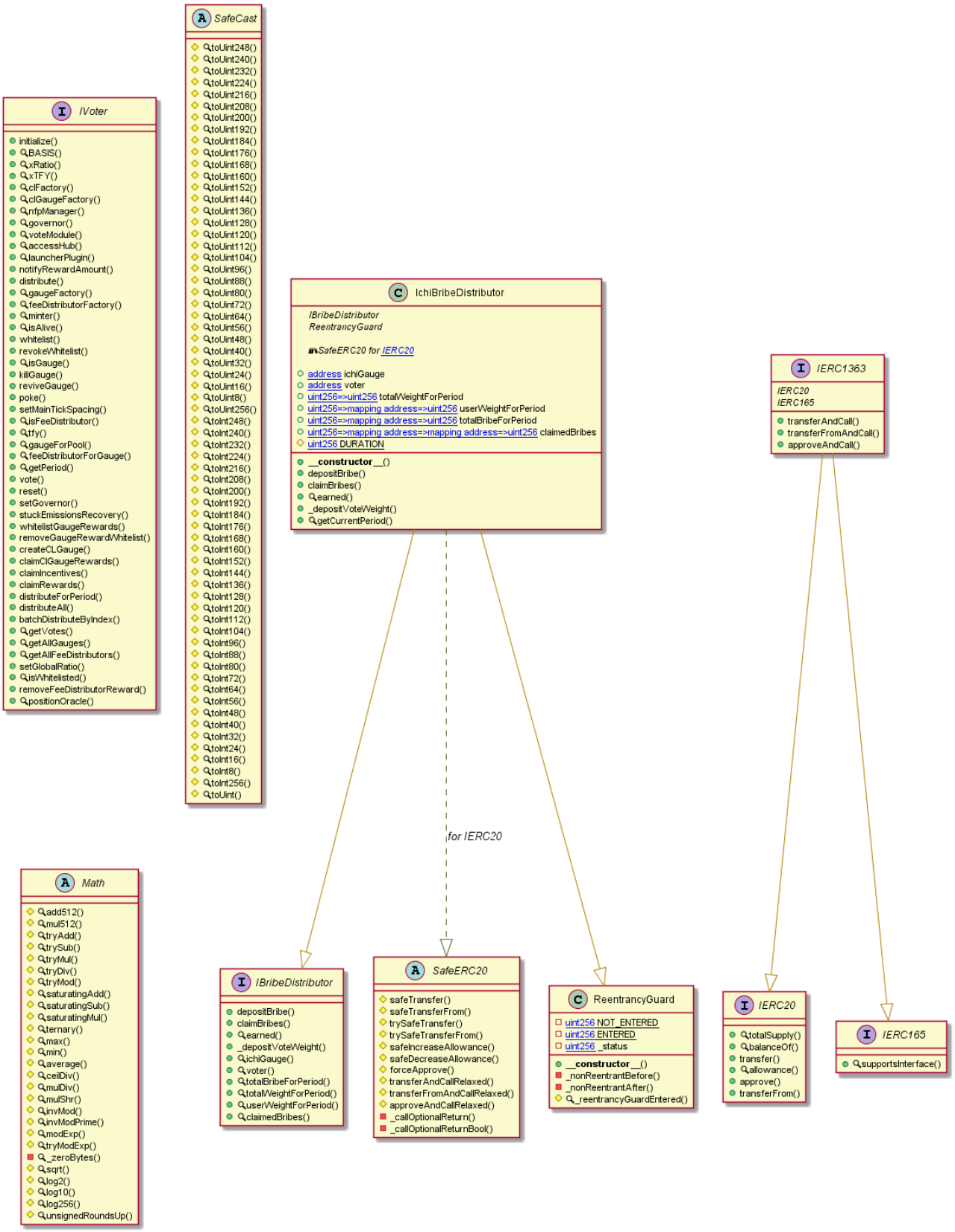
AccessHub Diagram



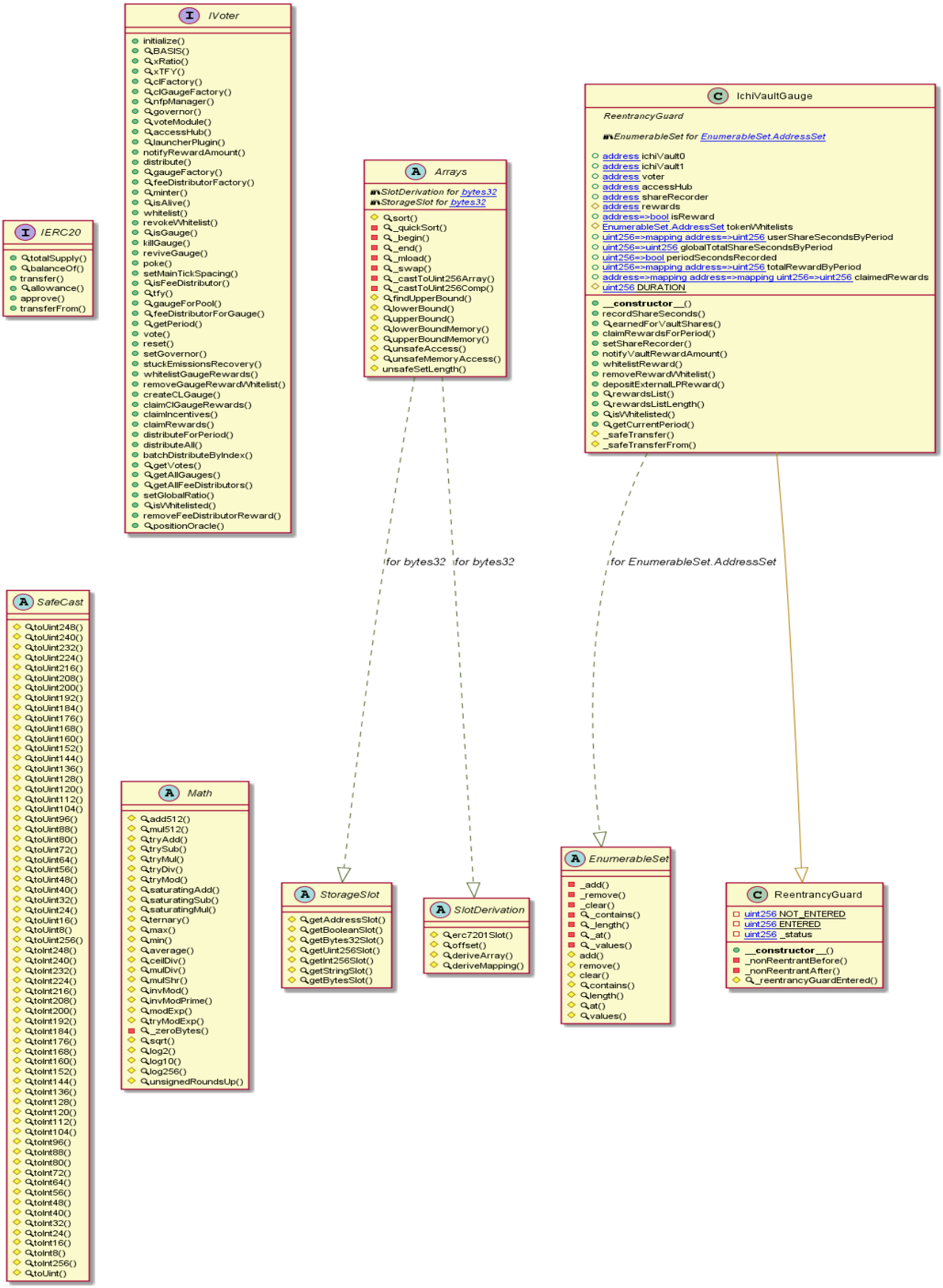
FeeCollector Diagram



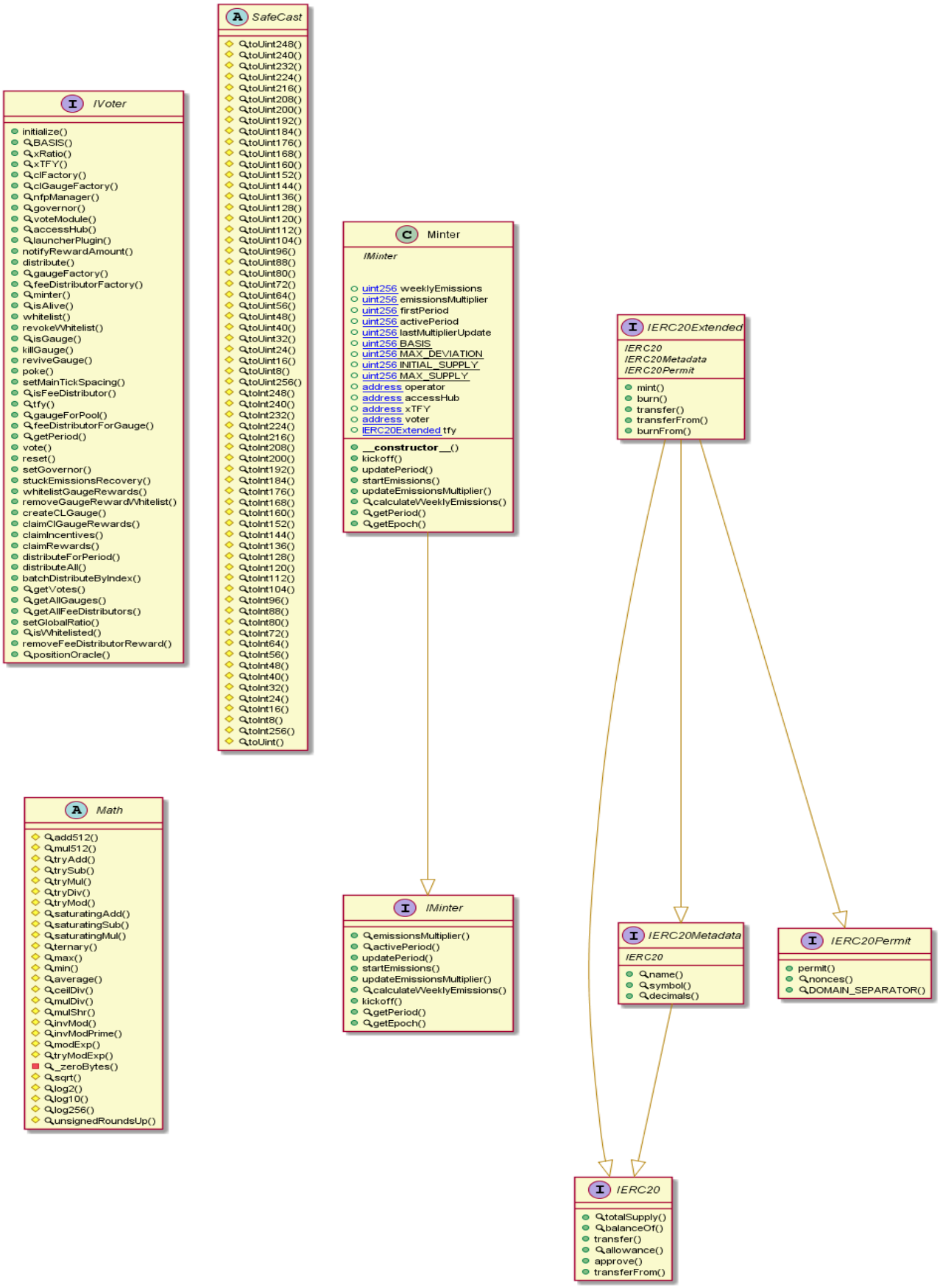
IchiBribeDistributor Diagram



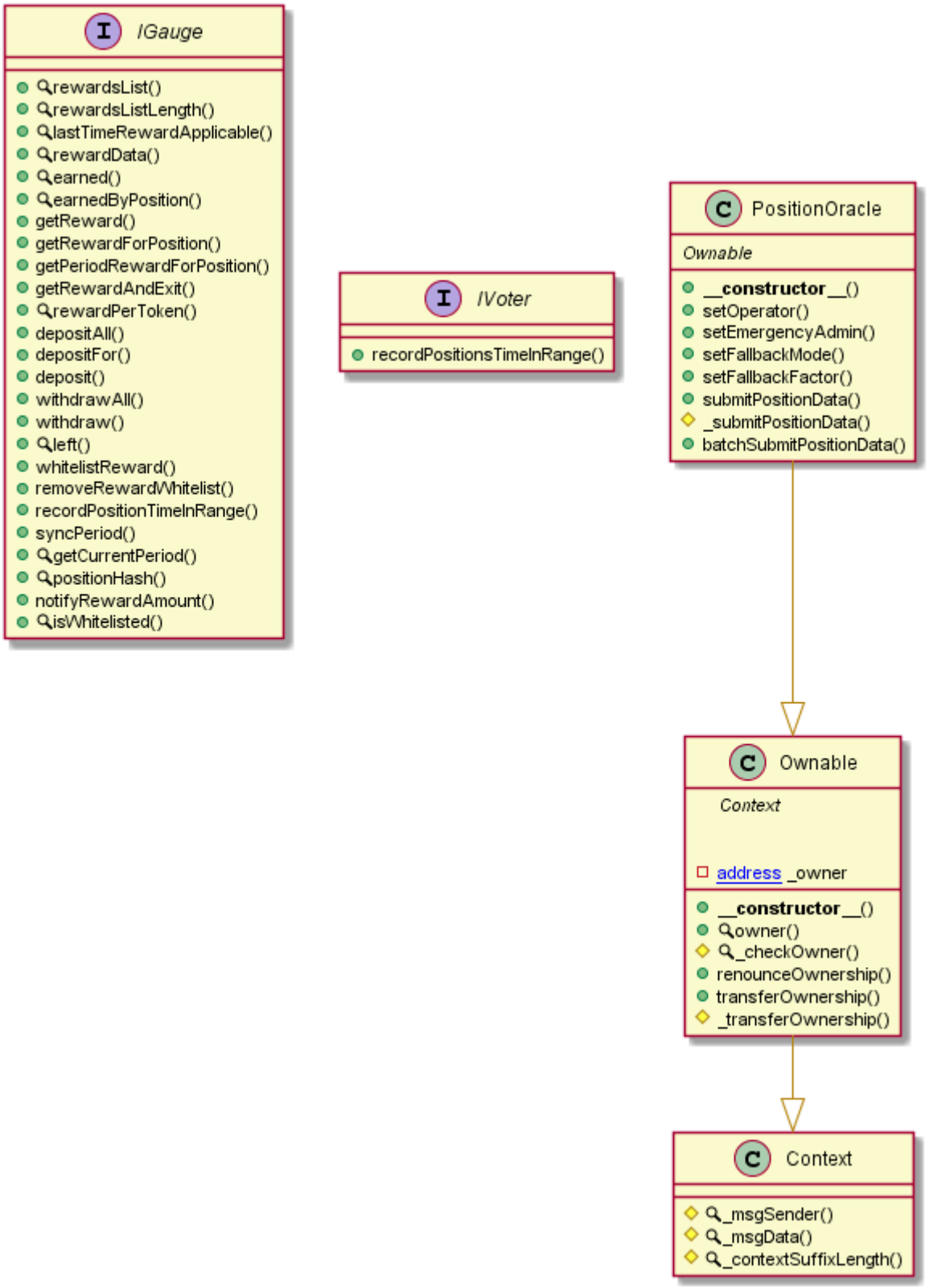
IchiVaultGauge Diagram



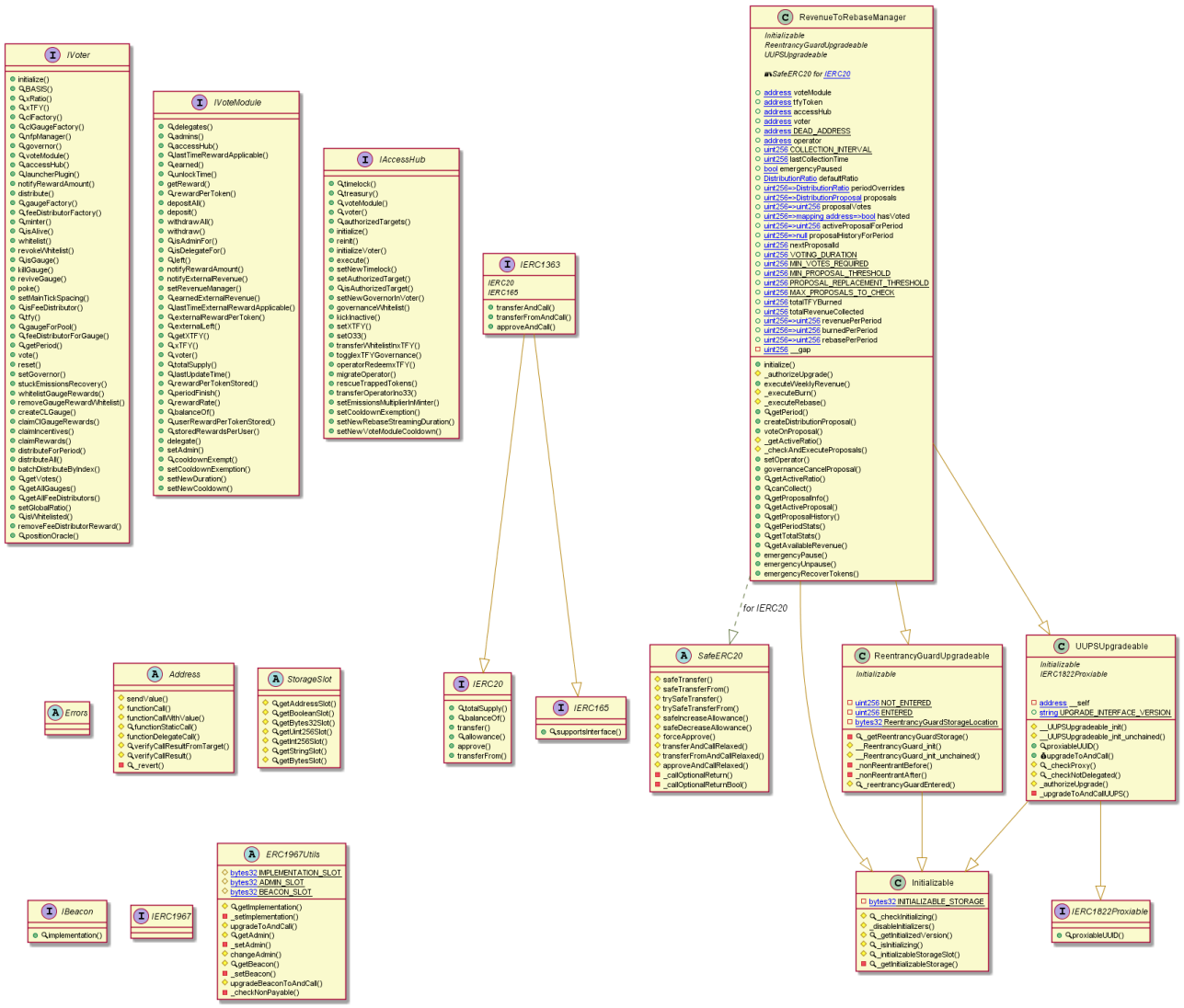
Minter Diagram



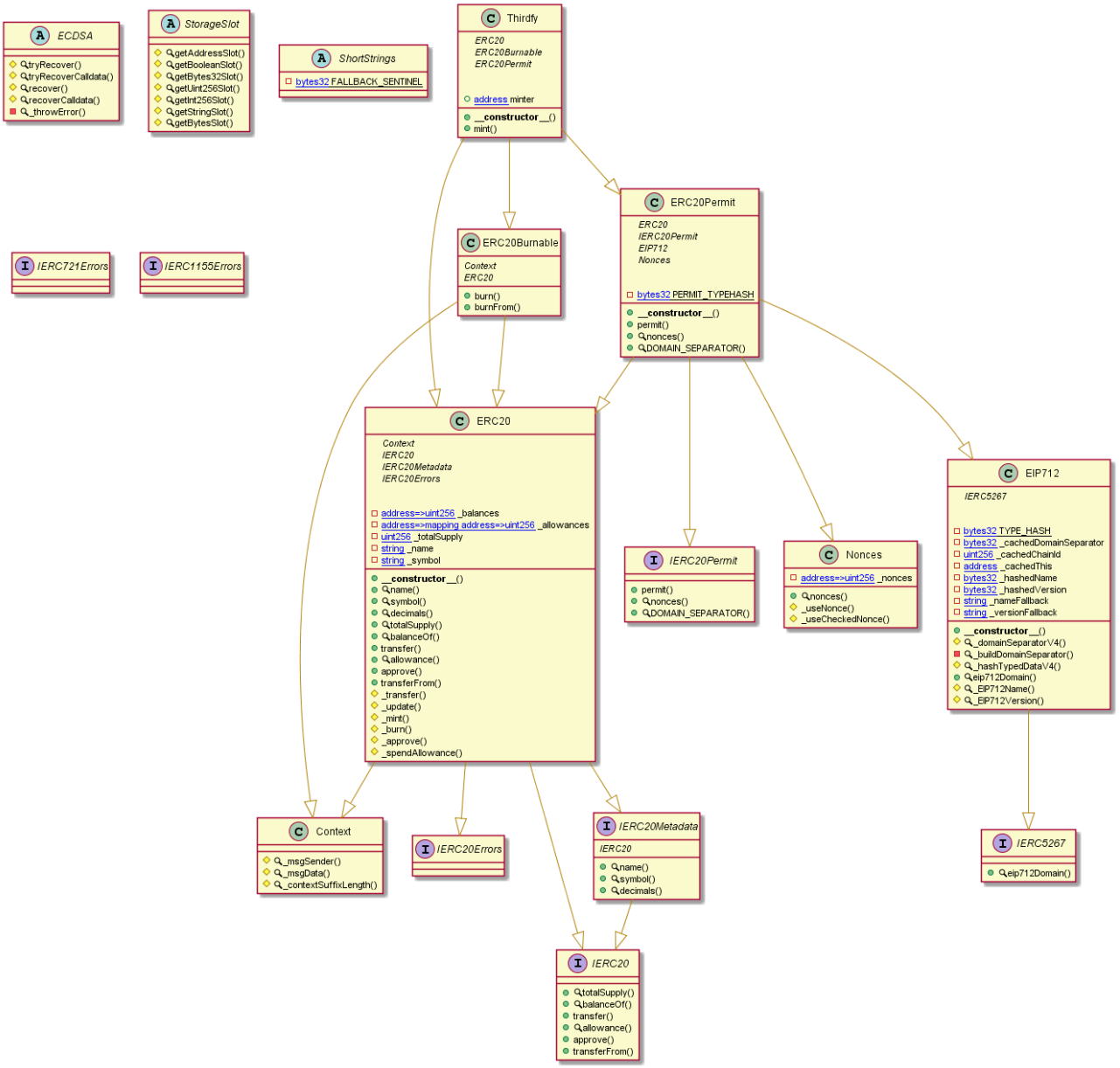
PositionOracle Diagram



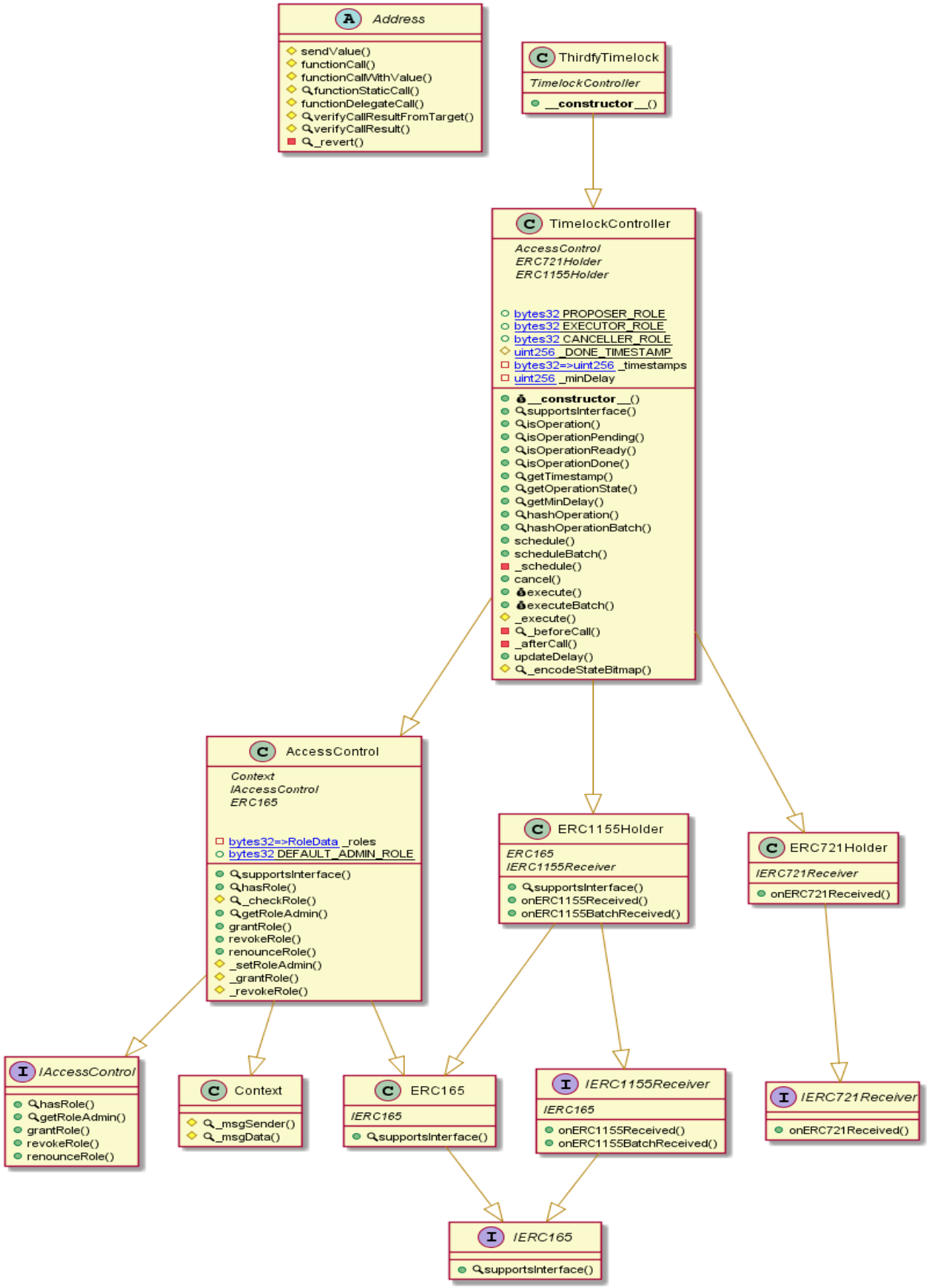
RevenueToRebaseManager Diagram



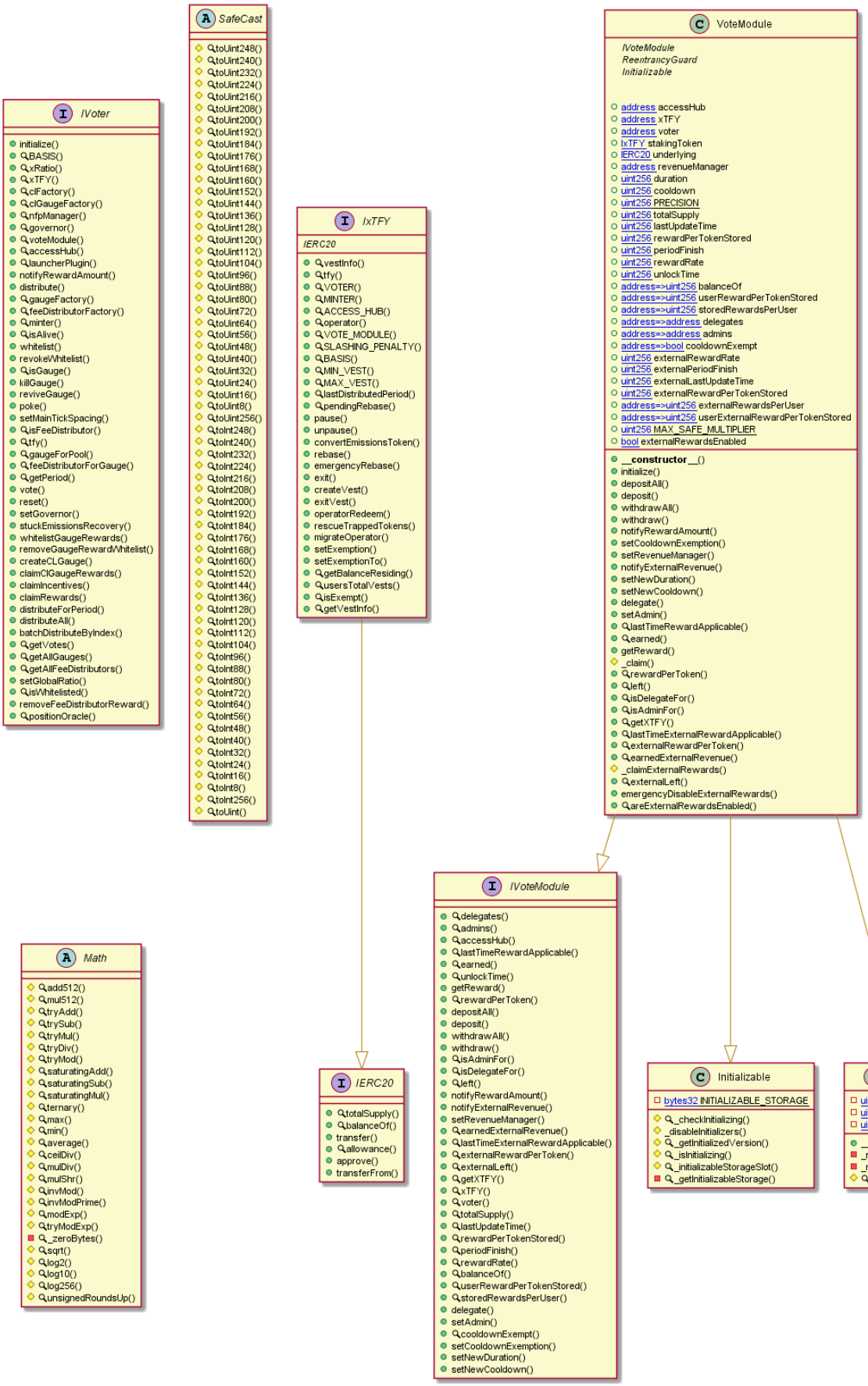
Thirdfy Diagram



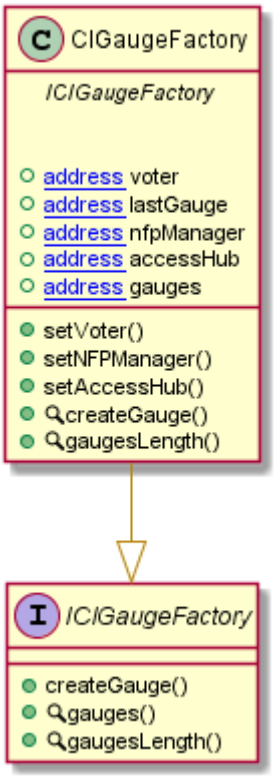
ThirdfyTimelock Diagram



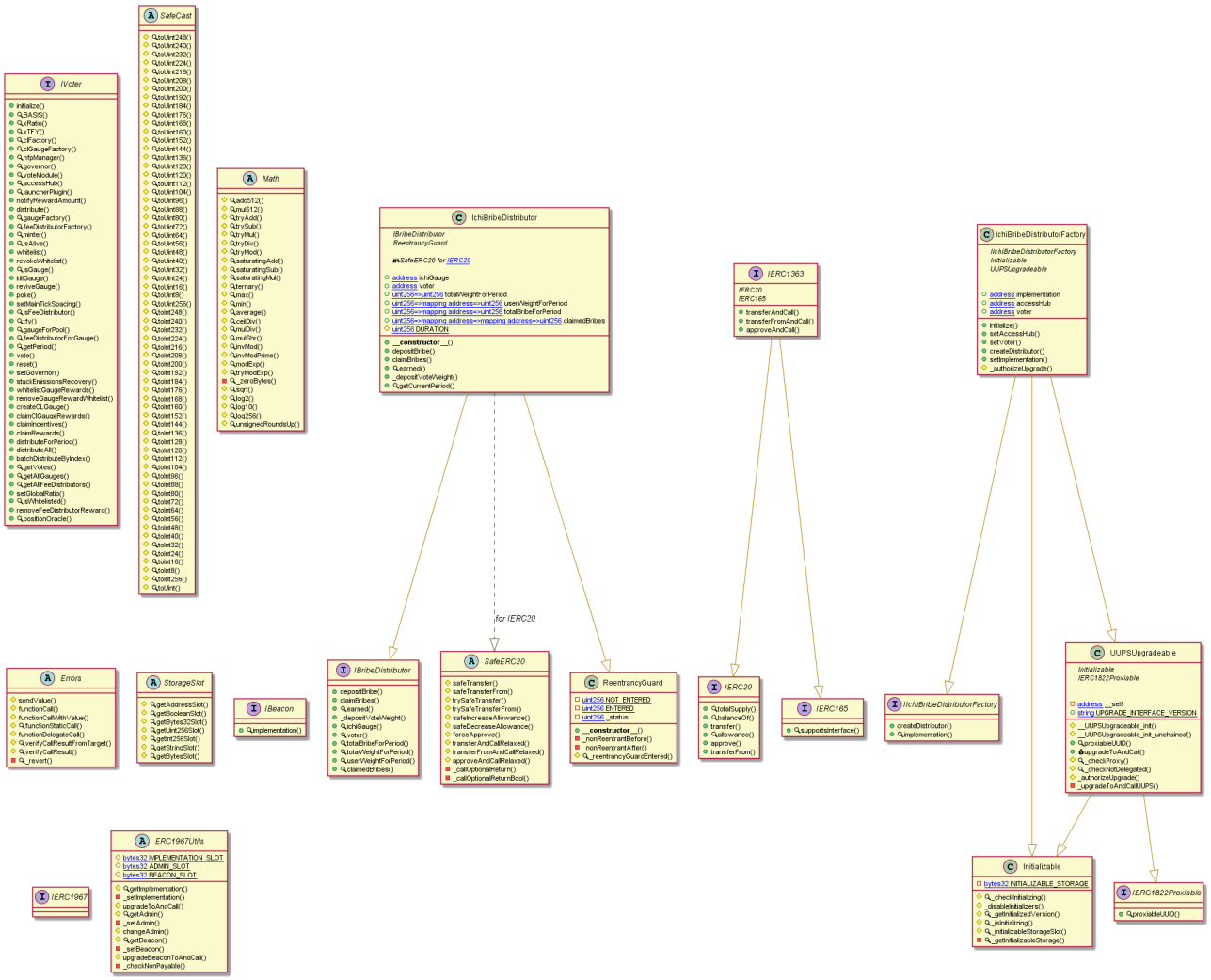
VoteModule Diagram



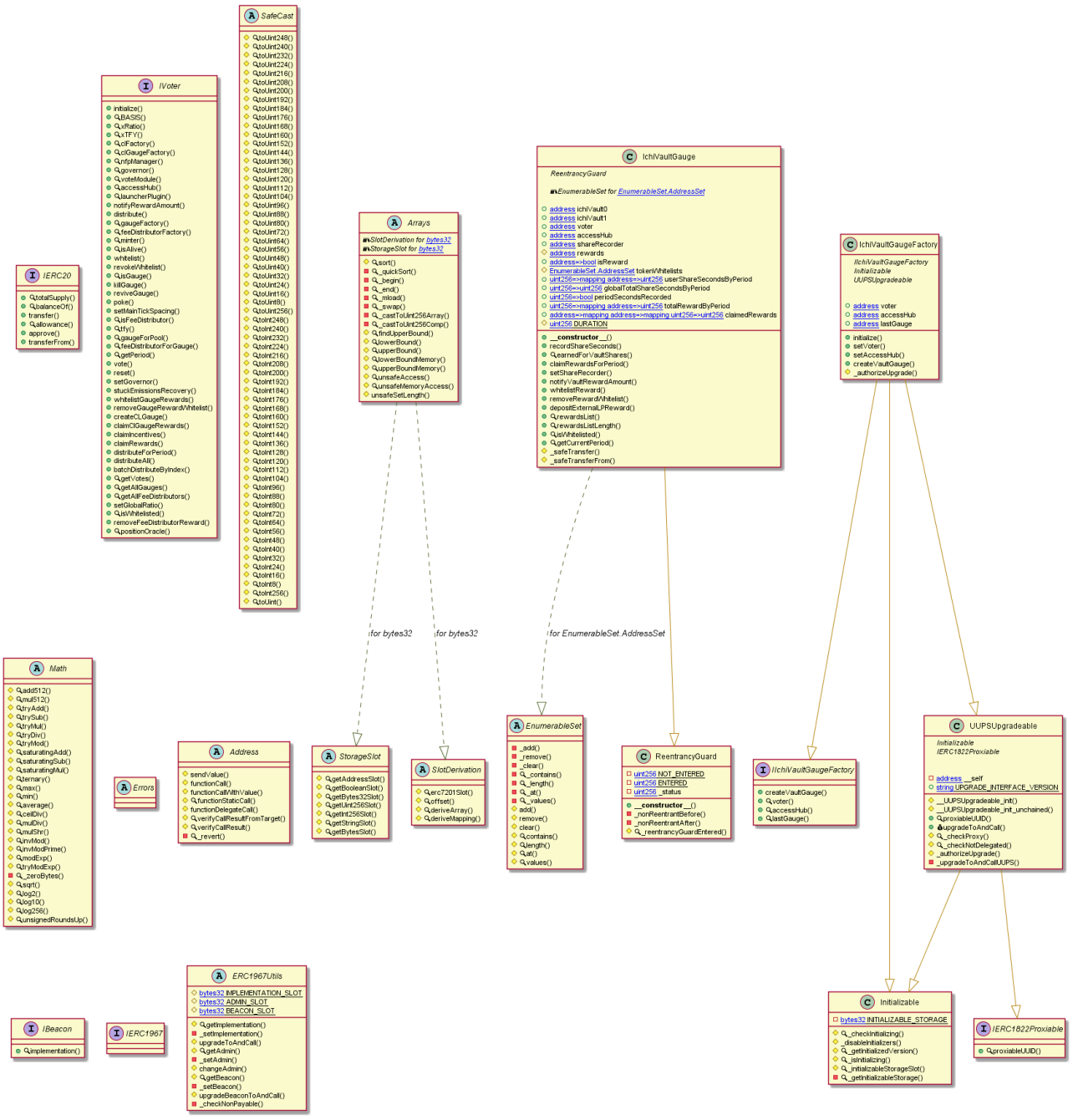
ClGaugeFactory Diagram



IchiBribeDistributorFactory Diagram



IchiVaultGaugeFactory Diagram



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We analyzed the project together. Below are the results.

Slither Log >> o33.sol

```
INFO:Detectors:
o33.whitelistRelayer(address,bool)._status (o33.sol#2180) shadows:
	- ReentrancyGuard._status (o33.sol#1895) (state variable)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
o33.transferOperator(address)._newOperator (o33.sol#2158) lacks a zero-check on :
		- operator = _newOperator (o33.sol#2162)
o33.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8).user (o33.sol#2189) lacks a zero-check on :
		- (success,returnData) = address(this).call(abi.encodePacked(functionSignature,user)) (o33.sol#2208-2210)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in o33._deposit(address,address,uint256,uint256) (o33.sol#2262-2275):
	External calls:
	- voteModule.deposit(assets) (o33.sol#2271)
	State variables written after the call(s):
	- _mint(receiver,shares) (o33.sol#2272)
		- _balances[from] = fromBalance - value (o33.sol#1550)
		- _balances[to] += value (o33.sol#1562)
	- _mint(receiver,shares) (o33.sol#2272)
		- _totalSupply += value (o33.sol#1542)
		- _totalSupply -= value (o33.sol#1557)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in o33._withdraw(address,address,address,uint256,uint256) (o33.sol#2277-2296):
	External calls:
	- voteModule.withdraw(assets) (o33.sol#2291)
	Event emitted after the call(s):
	- Withdraw(caller,receiver,owner,assets,shares) (o33.sol#2295)
```

```
Reentrancy in o33.claimRebase() (o33.sol#2034-2083):
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
o33.isUnlocked() (o33.sol#2238-2250) uses timestamp for comparisons
        Dangerous comparisons:
        - timeLeftInPeriod <= 3600 (o33.sol#2245)
o33.isCooldownActive() (o33.sol#2253-2257) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= unlockTime (o33.sol#2256)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Parameter o33.rescue(address,uint256)._token (o33.sol#2135) is not in mixedCase
Parameter o33.transferOperator(address)._newOperator (o33.sol#2158) is not in mixedCase
Parameter o33.whitelistAggregator(address,bool)._aggregator (o33.sol#2169) is not in
mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
o33.activePeriod (o33.sol#1957) should be immutable
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-decl
ared-immutable
INFO:Slither:o33.sol analyzed (18 contracts with 93 detectors), 53 result(s) found
```

Slither Log >> xTFY.sol

```
INFO:Detectors:
xTFY.constructor(address,address,address,address,address,address)._voteModule
(xTFY.sol#1633) lacks a zero-check on :
            - VOTE_MODULE = _voteModule (xTFY.sol#1641)
xTFY.migrateOperator(address)._operator (xTFY.sol#1818) lacks a zero-check on :
            - operator = _operator (xTFY.sol#1821)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Variable xTFY.TFY (xTFY.sol#1607) is not in mixedCase
Variable xTFY.VOTER (xTFY.sol#1608) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Slither:xTFY.sol analyzed (15 contracts with 93 detectors), 100 result(s) found
```

Slither Log >> VoterV4.sol

```
INFO:Detectors:
VoterV4._distribute(address,uint256,uint256) (VoterV4.sol#3468-3530) ignores return value by IERC20(_tfy).transfer(_gauge,_tfyClaimable) (VoterV4.sol#3489)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
VoterV4.tfy (VoterV4.sol#3164) is never initialized. It is used in:
	- VoterV4._distribute(address,uint256,uint256) (VoterV4.sol#3468-3530)
	- VoterV4.killGauge(address) (VoterV4.sol#3574-3604)
	- VoterV4.stuckEmissionsRecovery(address,uint256) (VoterV4.sol#3617-3635)
	- VoterV4._createGaugeForCLPool(address,address) (VoterV4.sol#3723-3781)
	- VoterV4.notifyRewardAmount(uint256) (VoterV4.sol#3826-3834)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
VoterV4.poke(address) (VoterV4.sol#3343-3378) uses a dangerous strict equality:
	- _lastVoted == period (VoterV4.sol#3370)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Parameter VoterV4.createIchiVaultGauge(address,address[2],address)._initialShareRecorder (VoterV4.sol#4080) is not in mixedCase
Parameter VoterV4.setShareRecorder(address,address)._ichiVaultGauge (VoterV4.sol#4132) is not in mixedCase
Parameter VoterV4.setShareRecorder(address,address)._recorder (VoterV4.sol#4132) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
VoterV4.xTFY (VoterV4.sol#3182) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Slither:VoterV4.sol analyzed (37 contracts with 93 detectors), 230 result(s) found
```

Slither Log >> AccessHub.sol

```
INFO:Detectors:
AccessHub.setCooldownExemption(address[],bool[]) (AccessHub.sol#2689-2696) has external calls inside a loop: voteModule.setCooldownExemption(_candidates[i],_exempt[i]) (AccessHub.sol#2694)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Parameter AccessHub.setCooldownExemption(address[],bool[])._exempt (AccessHub.sol#2691)
```

```
is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:AccessHub.sol analyzed (26 contracts with 93 detectors), 103 result(s) found
```

Slither Log >> FeeCollector.sol

```
INFO:Detectors:
FeeCollector.safeTransferWithLogging(IERC20,address,uint256,string,string) (FeeCollector.sol#915-939) uses a dangerous strict equality:
	- to == address(0) || amount == 0 (FeeCollector.sol#922)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
FeeCollector.setTreasury(address)._treasury (FeeCollector.sol#896) lacks a zero-check on :
		- treasury = _treasury (FeeCollector.sol#898)
FeeCollector.setFeeDistributor(address)._feeDist (FeeCollector.sol#909) lacks a zero-check on :
		- feeDist = _feeDist (FeeCollector.sol#911)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Parameter FeeCollector.setFeeDistributor(address)._feeDist (FeeCollector.sol#909) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
FeeCollector (FeeCollector.sol#826-1113) does not implement functions:
	- IFeeCollector.collectProtocolFees(IAlgebraPool) (FeeCollector.sol#109)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Slither:FeeCollector.sol analyzed (10 contracts with 93 detectors), 15 result(s) found
```

Slither Log >> IchiBribeDistributor.sol

```
INFO:Detectors:
Function IBribeDistributor._depositVoteWeight(address,uint256,uint256) (IchiBribeDistributor.sol#51) is not in mixedCase
Function IVoter.BASIS() (IchiBribeDistributor.sol#243) is not in mixedCase
Function IchiBribeDistributor._depositVoteWeight(address,uint256,uint256) (IchiBribeDistributor.sol#2852-2865) is not in mixedCase
Reference:
```

```
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:IchiBribeDistributor.sol analyzed (10 contracts with 93 detectors), 27 result(s) found
```

Slither Log >> IchiVaultGauge.sol

```
INFO:Detectors:
Reentrancy in IchiVaultGauge.claimRewardsForPeriod(uint256,address[])
(IchiVaultGauge.sol#4004-4020):
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Reentrancy in IchiVaultGauge.depositExternalLPReward(address,uint256)
(IchiVaultGauge.sol#4088-4108):
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Parameter IchiVaultGauge.removeRewardWhitelist(address)._reward (IchiVaultGauge.sol#4075)
is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:IchiVaultGauge.sol analyzed (10 contracts with 93 detectors), 78 result(s) found
```

Slither Log >> Minter.sol

```
INFO:Detectors:
Minter.updatePeriod() (Minter.sol#2484-2519) ignores return value by
tfy.approve(voter,_weeklyEmissions) (Minter.sol#2503)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Minter.updatePeriod() (Minter.sol#2484-2519) uses timestamp for comparisons
      Dangerous comparisons:
      - getPeriod() > period (Minter.sol#2489)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Low level call in Minter.updatePeriod() (Minter.sol#2484-2519):
      - (success,None) = xTFY.call(data) (Minter.sol#2510)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Minter.accessHub (Minter.sol#2439) should be immutable
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:Minter.sol analyzed (9 contracts with 93 detectors), 39 result(s) found
```

Slither Log >> PositionOracle.sol

```
INFO:Detectors:
PositionOracle._submitPositionData(address,uint128[]) (PositionOracle.sol#379-388) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Parameter PositionOracle.setOperator(address)._operator (PositionOracle.sol#339) is not in mixedCase
Parameter PositionOracle.setEmergencyAdmin(address)._admin (PositionOracle.sol#348) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:PositionOracle.sol analyzed (5 contracts with 93 detectors), 6 result(s) found
```

Slither Log >> RevenueToRebaseManager.sol

```
INFO:Detectors:
RevenueToRebaseManager (RevenueToRebaseManager.sol#1849-2630) is an upgradeable contract that does not protect its initialize functions:
RevenueToRebaseManager.initialize(address,address,address,address,address) (RevenueToRebaseManager.sol#2041-2072). Anyone can delete the contract with: UUPSUpgradeable.upgradeToAndCall(address,bytes) (RevenueToRebaseManager.sol#1759-1762)Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-contract
INFO:Detectors:
Reentrancy in RevenueToRebaseManager.executeWeeklyRevenue() (RevenueToRebaseManager.sol#2101-2151):
        - RevenueToRebaseManager.canCollect() (RevenueToRebaseManager.sol#2446-2448)
        - RevenueToRebaseManager.getTotalStats() (RevenueToRebaseManager.sol#2547-2557)
        - RevenueToRebaseManager.lastCollectionTime (RevenueToRebaseManager.sol#1882)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
RevenueToRebaseManager._executeRebase(uint256) (RevenueToRebaseManager.sol#2202-2220) ignores return value by token.approve(voteModule,0) (RevenueToRebaseManager.sol#2219)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
RevenueToRebaseManager.__gap (RevenueToRebaseManager.sol#2011) is never used in RevenueToRebaseManager (RevenueToRebaseManager.sol#1849-2630)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Slither:RevenueToRebaseManager.sol analyzed (18 contracts with 93 detectors), 59 result(s) found
```

Slither Log >> Thirdfy.sol

```
INFO:Detectors:
Thirdfy.constructor(address)._minter (Thirdfy.sol#968) lacks a zero-check on :
        - minter = _minter (Thirdfy.sol#970)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Thirdfy.minter (Thirdfy.sol#965) should be immutable
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:Thirdfy.sol analyzed (17 contracts with 93 detectors), 23 result(s) found
```

Slither Log >> ThirdfyTimelock.sol

```
INFO:Detectors:
TimelockController.getOperationState(bytes32) (ThirdfyTimelock.sol#532-543) uses a dangerous strict equality:
    - timestamp == 0 (ThirdfyTimelock.sol#534)
TimelockController.getOperationState(bytes32) (ThirdfyTimelock.sol#532-543) uses a dangerous strict equality:
    - timestamp == _DONE_TIMESTAMP (ThirdfyTimelock.sol#536)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
TimelockController.getOperationState(bytes32) (ThirdfyTimelock.sol#532-543) uses timestamp for comparisons
    Dangerous comparisons:
    - timestamp == 0 (ThirdfyTimelock.sol#534)
    - timestamp == _DONE_TIMESTAMP (ThirdfyTimelock.sol#536)
    - timestamp > block.timestamp (ThirdfyTimelock.sol#538)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
TimelockController._encodeStateBitmap(TimelockController.OperationState) (ThirdfyTimelock.sol#788-790) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Slither:ThirdfyTimelock.sol analyzed (12 contracts with 93 detectors), 17 result(s) found
```

Slither Log >> VoteModule.sol

```
INFO:Detectors:
VoteModule.notifyRewardAmount(uint256) (VoteModule.sol#3118-3161) uses arbitrary from in transferFrom: transferred = underlying.transferFrom(xTFY,address(this),amount) (VoteModule.sol#3129)
```

```
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:Parameter VoteModule.setNewCooldown(uint256)._cooldownInSeconds
(VoteModule.sol#3245) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:VoteModule.sol analyzed (9 contracts with 93 detectors), 75 result(s) found
```

Slither Log >> ClGaugeFactory.sol

```
INFO:Detectors:
ClGaugeFactory.gauges (ClGaugeFactory.sol#23) is never initialized. It is used in:
	- ClGaugeFactory.gaugesLength() (ClGaugeFactory.sol#44-46)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
ClGaugeFactory.lastGauge (ClGaugeFactory.sol#18) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Slither:ClGaugeFactory.sol analyzed (2 contracts with 93 detectors), 8 result(s) found
```

Slither Log >> IchiBribeDistributorFactory.sol

```
INFO:Detectors:
Contract locking ether found:
	Contract IchiBribeDistributorFactory (IchiBribeDistributorFactory.sol#3617-3706) has
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
Parameter IchiBribeDistributorFactory.setImplementation(address)._newImplementation
(IchiBribeDistributorFactory.sol#3696) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:IchiBribeDistributorFactory.sol analyzed (20 contracts with 93 detectors), 61
result(s) found
```

Slither Log >> IchiVaultGaugeFactory.sol

```
INFO:Detectors:
IchiVaultGaugeFactory (IchiVaultGaugeFactory.sol#4798-4885) is an upgradeable contract that
does not protect its initialize functions: IchiVaultGaugeFactory.initialize(address,address)
(IchiVaultGaugeFactory.sol#4830-4835). Anyone can delete the contract with:
```

```
UUPSUpgradeable.upgradeToAndCall(address,bytes)
(IchiVaultGaugeFactory.sol#4731-4734)Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-contract
INFO:Detectors:
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Parameter IchiVaultGaugeFactory.createVaultGauge(address[2],address)._initialShareRecorder
(IchiVaultGaugeFactory.sol#4866) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:IchiVaultGaugeFactory.sol analyzed (20 contracts with 93 detectors), 103 result(s)
found
```

Solidity Static Analysis

o33.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 259:22:

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
Pos: 313:50:

Gas costs:
Gas requirement of function o33.rescue is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 231:4:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 339:17:

xTFY.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 299:17:

Gas costs:
Gas requirement of function xTFY.setExemptionTo is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 359:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 326:8:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 366:8:

Similar variable names:
xTFY.createVest(uint256) : Variables have very similar names "MIN_VEST" and "MAX_VEST". Note: Modifiers are currently not considered by this static analysis.
Pos: 260:34:

VoterV4.sol

Transaction origin:
Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
Pos: 506:26:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 649:16:

Gas costs:
Gas requirement of function RewardClaimers.claimRewards is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 770:4:

Delete dynamic array:
The "delete" operation when applied to a dynamically sized array in Solidity generates code to

delete each of the elements contained. If the array is large, this operation can surpass the block gas limit and raise an OOG exception. Also nested dynamically sized objects can produce the same results.
Pos: 255:12:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 183:8:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 333:8:

Delete from dynamic array:
Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
Pos: 245:16:

AccessHub.sol

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
Pos: 149:27:

Gas costs:
Gas requirement of function AccessHub.setNewGovernorInVoter is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 174:4:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage

values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 184:8:

FeeCollector.sol

Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in FeeCollector.safeTransferWithLogging(contract IERC20,address,uint256,string,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.Note: Import aliases are currently not supported by this static analysis.
Pos: 100:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 88:8:

IchiBribeDistributor.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 193:15:

Gas costs:
Gas requirement of function IchiBribeDistributor.claimBribes is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 93:4:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at

maximum you can pass to such functions to make it successful.
Pos: 94:8:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 55:8:

IchiVaultGauge.sol

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
Pos: 323:44:

Gas costs:
Gas requirement of function IchiVaultGauge.isWhitelisted is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 300:4:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 120:8:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 204:8:

Minter.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.

That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 174:17:

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
Pos: 105:35:

Gas costs:
Gas requirement of function Minter.startEmissions is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 117:4:

PositionOracle.sol

Gas costs:
Gas requirement of function PositionOracle.batchSubmitPositionData is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 106:4:

Constant/View/Pure functions:
PositionOracle.setFallbackMode(bool) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
Pos: 71:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 46:8:

RevenueToRebaseManager.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 332:55:

Gas costs:
Gas requirement of function RevenueToRebaseManager.executeWeeklyRevenue is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 292:4:

Similar variable names:
RevenueToRebaseManager.createDistributionProposal(uint256,uint256) : Variables have very similar names "proposals" and "proposalId". Note: Modifiers are currently not considered by this static analysis.
Pos: 464:52:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 804:8:

Delete from dynamic array:
Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
Pos: 622:12:

Thirdfy.sol

Gas costs:
Gas requirement of function Thirdfy.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 24:4:

VoteModule.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 244:20:

Gas costs:
Gas requirement of function VoteModule.withdrawAll is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your

functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 227:4:

Similar variable names:
VoteModule.delegate(address) : Variables have very similar names "delegates" and "delegatee".
Note: Modifiers are currently not considered by this static analysis.
Pos: 415:34:

Delete from dynamic array:
Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
Pos: 407:12:

ClGaugeFactory.sol

No return:
IClGaugeFactory.createGauge(address): Defines a return type but never explicitly returns a value.
Pos: 6:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 21:8:

IchiBribeDistributorFactory.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 193:15:

Gas costs:
Gas requirement of function IchiBribeDistributorFactory.initialize is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 42:4:

No return:
IIchiBribeDistributorFactory.createDistributor(address,address): Defines a return type but never

explicitly returns a value.
Pos: 19:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 78:8:

IchiVaultGaugeFactory.sol

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
Pos: 323:44:

For loop over dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 183:8:

Similar variable names:
IchiVaultGauge.(address[2],address,address) : Variables have very similar names "ichiVault1" and "_ichiVaults". Note: Modifiers are currently not considered by this static analysis.
Pos: 85:21:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 326:8:

Solhint Linter

o33.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Contract name must be in CamelCase
Pos: 1:16
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:65
Avoid to use low level calls.
Pos: 51:312
Avoid making time-based decisions in your business logic
Pos: 17:360
```

xTFY.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Contract name must be in CamelCase
Pos: 1:13
Variable name must be in mixedCase
Pos: 5:32
Explicitly mark visibility of state
Pos: 5:37
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:61
Avoid making time-based decisions in your business logic
Pos: 18:298
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 13:302
```

VoterV4.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:14
Use double quotes for string literals
Pos: 32:40
Contract has 41 states declarations but allowed no more than 15
Pos: 1:45
Code contains empty blocks
```

```
Pos: 53:988
Variable "poolAddress" is unused
Pos: 9:954
Variable "period" is unused
Pos: 9:955
Avoid to use low level calls.
Pos: 23:1026
```

AccessHub.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Code contains empty blocks
Pos: 88:90
Avoid to use low level calls.
Pos: 28:148
```

FeeCollector.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Use double quotes for string literals
Pos: 71:25
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:55
Code contains empty blocks
Pos: 17:66
```

IchiBribeDistributor.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:52
Error message for require is too long
Pos: 9:53
Avoid making time-based decisions in your business logic
Pos: 16:192
```

IchiVaultGauge.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:76
Avoid making time-based decisions in your business logic
Pos: 16:305
Avoid to use low level calls.
Pos: 45:313
Avoid to use low level calls.
Pos: 45:322
```

Minter.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:46
Avoid to use low level calls.
Pos: 36:104
Avoid making time-based decisions in your business logic
Pos: 18:173
```

PositionOracle.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:44
Error message for require is too long
Pos: 9:79
Code contains empty blocks
Pos: 29:109
```

RevenueToRebaseManager.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Contract has 21 states declarations but allowed no more than 15
Pos: 1:39
Avoid making time-based decisions in your business logic
Pos: 44:781
```

```
Error message for require is too long
Pos: 9:801
```

Thirdfy.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:14
```

ThirdfyTimelock.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:6
Code contains empty blocks
Pos: 65:11
```

VoteModule.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Contract has 27 states declarations but allowed no more than 15
Pos: 1:39
Avoid making time-based decisions in your business logic
Pos: 34:131
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:148
Provide an error message for require
Pos: 9:391
Avoid making time-based decisions in your business logic
Pos: 51:651
```

ClGaugeFactory.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
```

```
global import of path ICLGaugeInterfaces.sol is not allowed. Specify
names to import individually or bind all exports of the module into a
name (import "path" as Name)
Pos: 1:3
Error message for revert is too long
Pos: 9:35
```

IchiBribeDistributorFactory.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Code contains empty blocks
Pos: 91:103
```

IchiVaultGaugeFactory.sol

```
Compiler version ^0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:76
Avoid making time-based decisions in your business logic
Pos: 16:305
```

Software analysis result:

This software reported many false positive results, and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io