Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Honor DAO Protocol
Website: honordao.finance
Platform: Binance Chain
Language: Solidity
Date: June 11th, 2022

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND 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 contracted by Honor DAO to perform the Security audit of the Honor DAO Protocol smart contracts 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 June 11th, 2022.

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

Honor DAO Protocol is a Defi Program which has functions like mint, swap, OpenTrade, burn, twap, spot, update, mock, info, transfer, set pool, claimable, zap, addLiquidity, cleanDust, etc. The Honor DAO contract inherits the IERC721, ERC20, SafeERC20, Ownable, ReentrancyGuard, Address, IUniswapV2Router02, IUniswapV2Pair, IERC20, Math, SafeMath, Initializable, ERC20Burnable, TransparentUpgradeableProxy 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 Honor DAO Protocol Smart Contracts**
Platform	**Binance Chain / Solidity**
File 1	Pool.sol
File 1 MD5 Hash	DBA2377A307942CC1A258820CE39ECE2
File 2	SwapStrategyPOL.sol
File 2 MD5 Hash	36814239B61233184D546EE4BAC29A24
File 3	Timelock.sol
File 3 MD5 Hash	94F559046B7CB4335EE0F49341A23DA0

File 4	HonorDaoChef.sol
File 4 MD5 Hash	723A4D8023F357BBBE1FAD15543B5CE1
File 5	HonorDaoStaking.sol
File 5 MD5 Hash	C558D0CEB8516B8D0506CD4FE3993C47
File 6	HonorDaoZapMMSwap.sol
File 6 MD5 Hash	8640A54BBC0B412D15BA74964A6DE86F
File 7	NFTController.sol
File 7 MD5 Hash	7B517FFAE5E28C8D3B7020747FFA8659
File 8	NFTControllerProxy.sol
File 8 MD5 Hash	E2E8A433C71CE32907140DCF3F28480D
File 9	Fund.sol
File 9 MD5 Hash	47370A0301A3BBA40747C7FFD8A18E6B
File 10	DaoFund.sol
File 10 MD5 Hash	03E88C7F44DA8F457B869D1458D326D2
File 11	DevFund.sol
File 11 MD5 Hash	B1D2A46F7B6251C4350579D463E4441E
File 12	HONORReserve.sol
File 12 MD5 Hash	585DD279DF053AAA9DDFAB3D54E8A809
File 13	TreasuryFund.sol
File 13 MD5 Hash	79648260D8B2C56BAC5C53D30AD1538E
File 14	MockERC20.sol
File 14 MD5 Hash	94278D4A01D92E76EBDE914556B3A6A0
File 15	MockTreasury.sol
File 15 MD5 Hash	92986372AD7DF3224D5001BD06A7E7E2
File 16	MasterOracle.sol
File 16 MD5 Hash	26FFB8A6EB84AABF384A830DB4572C0A
File 17	UniswapPairOracle.sol

File 17 MD5 Hash	37801A23DE6F4571ADD278A4A062C1D5
File 18	XToken.sol
File 18 MD5 Hash	E905290FA8FFB182588943AA4D60EAC6
File 19	YToken.sol
File 19 MD5 Hash	FFA9BDAB9AEE9D07DB46CB3A23A34696
File 20	HONOR.sol
File 20 MD5 Hash	2380C9128FD368380F28A873EFA536FC
File 21	KNIGHTTestToken.sol
File 21 MD5 Hash	423D5935E98AB8ADFDA1244E328504B1
File 22	XNIGHT.sol
File 22 MD5 Hash	D093B724D075730EA0AF7AE0ECE88401
File 23	HonorDaoTreasury.sol
File 23 MD5 Hash	D0C2F71B1DC8FE8F68A55DCDA5BD33FB
File 24	StratRecollateralize.sol
File 24 MD5 Hash	C612D113DFC66D0A846B36AA8CDA5700
File 25	StratReduceReserveLP.sol
File 25 MD5 Hash	15F43725DBE630F4EDEFD5F93261AF0B
Audit Date	June 11th,2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 Pool.sol • Refresh Cooldown: 1 hour • Ratio StepUp: 0.2% • Ratio StepDown: 0.2% • Price Target: 1 KNIGHT • Price Band: 0.004 • Minimum Collateral Ratio: 9,00,000 • YToken Slippage: 20% • Redemption Fee: 0.5% • Redemption Fee Maximum: 0.9% • Minting Fee: 0.3% • Minting Fee Maximum: 0.5%	**YES, This is valid.**
File 2 SwapStrategyPOL.sol • Swap Slippage: 20%	**YES, This is valid.**
File 3 Timelock.sol • Grace Period: 14 Days • Minimum Delay: 12 Hours • Maximum Delay: 30 Days	**YES, This is valid.**
File 4 HonorDaoChef.sol • Maximum Imumreward: 10 Tokens Per Second • NFT Boost Rate: 100	**YES, This is valid.**
File 5 HonorDaoStaking.sol • Rewards Duration: 1 week • Lock Duration: 4 weeks • Team Reward Percent: 20%	**YES, This is valid.**
File 6 HonorDaoZapMMSwap.sol HonorDaoZap is a ZapperFi's simplified version of	**YES, This is valid.**

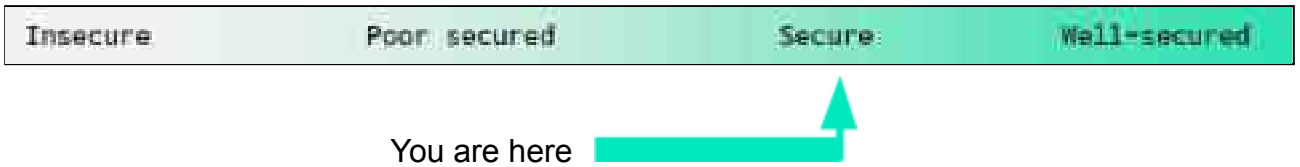
zapper contract which will: 1. use BNB to swap to target token 2. make LP between BNB and target token 3. add into HonorDaoChef farm	
File 7 Fund.sol • Fund has functions like: allocation, transfer, etc.	**YES, This is valid.**
File 8 DaoFund.sol • Allocation: 10% • Vesting Duration: 2 Years	**YES, This is valid.**
File 9 DevFund.sol • Allocation: 10% • Vesting Duration: 1 Years	**YES, This is valid.**
File 10 HONORReserve.sol • MDSReserve has functions like: initialize, setRewarder, etc.	**YES, This is valid.**
File 11 TreasuryFund.sol • Allocation: 10% • Vesting Duration: 1 Years	**YES, This is valid.**
File 12 MockERC20.sol • MockERC20 has functions like: mint, etc.	**YES, This is valid.**
File 13 MockTreasury.sol • MockTreasury has functions like: mock, etc.	**YES, This is valid.**
File 14 MasterOracle.sol • MasterOracle has functions like: getYTokenPrice, getYTokenTWAP, etc.	**YES, This is valid.**
File 15 UniswapPairOracle.sol • Period: 60-minute Twap (Time-weighted Average Price)	**YES, This is valid.**

• Maximum Period: 48 Hours • Minimum Period: 10 Minutes • Leniency: 12 Hours	
File 16 XToken.sol • XToken has functions like: mint, etc.	**YES, This is valid.**
File 17 YToken.sol • YToken has functions like: burn, etc.	**YES, This is valid.**
File 18 HONOR.sol • Total Supply: 7 Million ether	**YES, This is valid.**
File 19 KNIGHTTestToken.sol • Genesis Supply = 100 will be minted as total supply	**YES, This is valid.**
File 20 HonorDaoTreasury.sol • HonorDaoTreasury has functions like: balanceOf, requestFund, etc.	**YES, This is valid.**
File 21 StratRecollateralize.sol • StratRecollateralize has functions like: recollateralize.	**YES, This is valid.**
File 22 StratReduceReserveLP.sol • StratReduceReserveLPhas functions like: reduceReserve, swap.	**YES, This is valid.**
File 23 NFTController.sol • NFTController has functions like: setBoostRate, setWhitelist, etc.	**YES, This is valid.**
File 24 NFTControllerProxy.sol • NFTControllerProxy has access to the TransparentUpgradeableProxy contract.	**YES, This is valid.**

File 25 XNIGHT.sol • XNIGHT has functions like: OpenTrade, includeToWhitelist, etc.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts do 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 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 2 low and some very low level issues.

Investors Advice: 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	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	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	Moderated
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 not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 25 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Honor DAO 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 Honor DAO Protocol.

The Honor DAO team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are not well commented on smart contracts. We suggest using Ethereum's NatSpec style for the commenting.

Documentation

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

As mentioned above, code parts are not well commented. But the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website https://honordao.finance which provided rich information about the project architecture.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that 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

Pool.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	nonReentrant	modifier	Passed	No Issue
8	info	external	Passed	No Issue
9	usableCollateralBalance	read	Passed	No Issue
10	calcMint	read	Passed	No Issue
11	calcRedeem	read	Passed	No Issue
12	calcExcessCollateralBalance	read	Passed	No Issue
13	refreshCollateralRatio	write	Passed	No Issue
14	mint	external	Passed	No Issue
15	redeem	external	Passed	No Issue
16	collect	external	Passed	No Issue
17	recollateralize	external	Passed	No Issue
18	checkPriceFluctuation	internal	Passed	No Issue
19	toggle	write	access only Owner	No Issue
20	setCollateralRatioOptions	write	access only Owner	No Issue
21	toggleCollateralRatio	write	access only Owner	No Issue
22	setFees	write	access only Owner	No Issue
23	setMinCollateralRatio	external	access only Owner	No Issue
24	reduceExcessCollateral	external	access only Owner	No Issue
25	setSwapStrategy	external	access only Owner	No Issue
26	setOracle	external	access only Owner	No Issue
27	setYTokenSlippage	external	access only Owner	No Issue
28	setTreasury	external	Passed	No Issue
29	transferToTreasury	internal	Passed	No Issue

SwapStrategyPOL.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue

5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	lpBalance	read	Passed	No Issue
8	execute	external	Passed	No Issue
9	calculateSwapInAmount	internal	Passed	No Issue
10	swap	internal	Passed	No Issue
11	addLiquidity	internal	Passed	No Issue
12	cleanDust	external	access only Owner	No Issue
13	changeSlippage	external	access only Owner	No Issue

Timelock.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	setDelay	write	Passed	No Issue
3	acceptAdmin	write	Passed	No Issue
4	setPendingAdmin	write	Passed	No Issue
5	queueTransaction	write	Passed	No Issue
6	cancelTransaction	write	Passed	No Issue
7	executeTransaction	write	Passed	No Issue
8	getBlockTimestamp	internal	Passed	No Issue

HonorDaoChef.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	poolLength	read	Passed	No Issue
8	pendingReward	external	Passed	No Issue
9	updatePool	write	Passed	No Issue
10	massUpdatePools	write	Passed	No Issue
11	deposit	write	Passed	No Issue
12	withdraw	write	Passed	No Issue
13	harvest	write	Passed	No Issue
14	withdrawAndHarvest	write	Passed	No Issue
15	emergencyWithdraw	write	Passed	No Issue
16	harvestAllRewards	external	Passed	No Issue
17	checkPoolDuplicate	internal	Passed	No Issue
18	add	write	access only Owner	No Issue

19	set	write	access only Owner	No Issue
20	setRewardPerSecond	write	access only Owner	No Issue
21	setRewardMinter	external	Passed	No Issue
22	getBoost	read	Passed	No Issue
23	getSlots	read	Passed	No Issue
24	getTokenIds	read	Passed	No Issue
25	depositNFT	write	Passed	No Issue
26	withdrawNFT	write	Passed	No Issue
27	setNftController	write	access only Owner	No Issue
28	setNftBoostRate	write	access only Owner	No Issue

HonorDaoStaking.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	nonReentrant	modifier	Passed	No Issue
8	addReward	write	Function input parameters lack of check	Refer Audit Findings
9	approveRewardDistributor	external	access only Owner	No Issue
10	_rewardPerToken	internal	Passed	No Issue
11	_earned	internal	Passed	No Issue
12	lastTimeRewardApplicable	read	Passed	No Issue
13	getRewardForDuration	external	Passed	No Issue
14	claimableRewards	external	Passed	No Issue
15	totalBalance	external	Passed	No Issue
16	unlockedBalance	external	Passed	No Issue
17	earnedBalances	external	Passed	No Issue
18	withdrawableBalance	read	Passed	No Issue
19	stake	external	Passed	No Issue
20	mint	external	Function input parameters lack of check, Division before multiplication	Refer Audit Findings
21	withdraw	write	Passed	No Issue
22	getReward	write	Passed	No Issue
23	emergencyWithdraw	external	Critical operation lacks event log	Refer Audit Findings
24	_notifyReward	internal	Passed	No Issue
25	notifyRewardAmount	external	Passed	No Issue
26	recoverERC20	external	access only Owner	No Issue

27	setTeamWalletAddress	external	Passed	No Issue
28	setTeamRewardPercent	external	Passed	No Issue
29	updateReward	modifier	Passed	No Issue
30	rewardPerToken	external	Passed	No Issue
31	lockedBalances	external	Passed	No Issue
32	withdrawExpiredLocks	external	Passed	No Issue

HonorDaoZapMMSwap.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	nonReentrant	modifier	Passed	No Issue
8	zap	external	Passed	No Issue
9	swap	internal	Passed	No Issue
10	doSwapETH	internal	Passed	No Issue
11	approveToken	internal	Passed	No Issue
12	calculateSwapInAmount	internal	Passed	No Issue
13	addZap	external	access only Owner	No Issue
14	removeZap	external	access only Owner	No Issue

NFTController.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	initializer	modifier	Passed	No Issue
8	reinitializer	modifier	Passed	No Issue
9	onlyInitializing	modifier	Passed	No Issue
10	_disableInitializers	internal	Passed	No Issue
11	_setInitializedVersion	write	Passed	No Issue
12	initialize	write	Passed	No Issue
13	getBoostRate	external	Passed	No Issue
14	setWhitelist	external	access only Owner	No Issue
15	setDefaultBoostRate	external	access only Owner	No Issue

16	setBoostRate	external	access only Owner	No Issue

NFTControllerProxy.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	ifAdmin	modifier	Passed	No Issue
3	admin	external	access if Admin	No Issue
4	implementation	external	access if Admin	No Issue
5	changeAdmin	external	access if Admin	No Issue
6	upgradeTo	external	access if Admin	No Issue
7	upgradeToAndCall	external	access if Admin	No Issue
8	_admin	internal	Passed	No Issue
9	_beforeFallback	internal	Passed	No Issue

Fund.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	initializer	modifier	Passed	No Issue
8	reinitializer	modifier	Passed	No Issue
9	onlyInitializing	modifier	Passed	No Issue
10	_disableInitializers	internal	Passed	No Issue
11	_setInitializedVersion	write	Passed	No Issue
12	initialize	external	Passed	No Issue
13	allocation	read	Passed	No Issue
14	vestingStart	read	Passed	No Issue
15	vestingDuration	read	Passed	No Issue
16	currentBalance	read	Passed	No Issue
17	vestedBalance	read	Passed	No Issue
18	claimable	read	Passed	No Issue
19	transfer	external	access only Owner	No Issue

DaoFund.sol

Functions

Sl.	Functions	Type	Observation	Conclusion

1	constructor	write	Passed	No Issue
2	initialize	external	Passed	No Issue
3	allocation	read	Passed	No Issue
4	vestingStart	read	Passed	No Issue
5	vestingDuration	read	Passed	No Issue
6	currentBalance	read	Passed	No Issue
7	vestedBalance	read	Passed	No Issue
8	claimable	read	Passed	No Issue
9	transfer	external	access only Owner	No Issue
10	allocation	write	Passed	No Issue
11	vestingStart	write	Passed	No Issue
12	vestingDuration	write	Passed	No Issue

DevFund.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	external	Passed	No Issue
3	allocation	read	Passed	No Issue
4	vestingStart	read	Passed	No Issue
5	vestingDuration	read	Passed	No Issue
6	currentBalance	read	Passed	No Issue
7	vestedBalance	read	Passed	No Issue
8	claimable	read	Passed	No Issue
9	transfer	external	access only Owner	No Issue
10	allocation	write	Passed	No Issue
11	vestingStart	write	Passed	No Issue
12	vestingDuration	write	Passed	No Issue

HONORReserve.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializer	modifier	Passed	No Issue
3	reinitializer	modifier	Passed	No Issue
4	onlyInitializing	modifier	Passed	No Issue
5	_disableInitializers	internal	Passed	No Issue
6	_setInitializedVersion	write	Passed	No Issue
7	initialize	external	Passed	No Issue
8	setRewarder	external	Passed	No Issue
9	setPool	external	Passed	No Issue
10	transfer	external	Passed	No Issue

TreasuryFund.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	external	Passed	No Issue
3	allocation	read	Passed	No Issue
4	vestingStart	read	Passed	No Issue
5	vestingDuration	read	Passed	No Issue
6	currentBalance	read	Passed	No Issue
7	vestedBalance	read	Passed	No Issue
8	claimable	read	Passed	No Issue
9	transfer	external	access only Owner	No Issue
10	allocation	write	Passed	No Issue
11	vestingStart	write	Passed	No Issue
12	vestingDuration	write	Passed	No Issue

MockERC20.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	name	read	Passed	No Issue
3	symbol	read	Passed	No Issue
4	decimals	read	Passed	No Issue
5	totalSupply	read	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	transfer	write	Passed	No Issue
8	allowance	read	Passed	No Issue
9	approve	write	Passed	No Issue
10	transferFrom	write	Passed	No Issue
11	increaseAllowance	write	Passed	No Issue
12	decreaseAllowance	write	Passed	No Issue
13	_transfer	internal	Passed	No Issue
14	_mint	internal	Passed	No Issue
15	_burn	internal	Passed	No Issue
16	_approve	internal	Passed	No Issue
17	_spendAllowance	internal	Passed	No Issue
18	_beforeTokenTransfer	internal	Passed	No Issue
19	_afterTokenTransfer	internal	Passed	No Issue
20	mint	write	Passed	No Issue
21	decimals	read	Passed	No Issue

MockTreasury.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	mock	write	Passed	No Issue
3	info	read	Passed	No Issue

MasterOracle.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	getXTokenPrice	write	Passed	No Issue
8	getYTokenPrice	write	Passed	No Issue
9	getXTokenTWAP	write	Passed	No Issue
10	getYTokenTWAP	write	Passed	No Issue

UniswapPairOracle.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	setPeriod	external	access only Owner	No Issue
8	update	external	Passed	No Issue
9	twap	external	Passed	No Issue
10	spot	external	Passed	No Issue
11	currentBlockTimestamp	internal	Passed	No Issue
12	currentCumulativePrices	internal	Passed	No Issue

XToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	burn	write	Passed	No Issue
3	burnFrom	write	Passed	No Issue

4	onlyMinter	modifier	Passed	No Issue
5	setMinter	external	Passed	No Issue
6	mint	external	Unlimited Minting	Refer Audit Findings

YToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	burn	write	Passed	No Issue
3	burnFrom	write	Passed	No Issue

HONOR.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyMinter	modifier	Passed	No Issue
3	setMinter	external	Passed	No Issue
4	mint	external	access only Minter	No Issue
5	OpenTrade	external	Passed	No Issue
6	includeToWhitelist	write	Passed	No Issue
7	excludeFromWhitlist	write	Passed	No Issue
8	_transfer	internal	Passed	No Issue

KnightTestToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	burn	write	Passed	No Issue
3	burnFrom	write	Passed	No Issue
4	onlyMinter	modifier	Passed	No Issue
5	setMinter	external	Passed	No Issue
6	mint	external	Unlimited Minting	Refer Audit Findings

XNIGHT.sol

Functions

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

2	OpenTrade	external	Passed	No Issue
3	includeToWhitelist	write	Passed	No Issue
4	excludeFromWhitlist	write	Passed	No Issue
5	_transfer	internal	Passed	No Issue

HonorDaoTreasury.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	balanceOf	read	Passed	No Issue
8	requestFund	external	Passed	No Issue
9	addStrategy	external	access only Owner	No Issue
10	removeStrategy	external	access only Owner	No Issue

StratRecollateralize.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	recollateralize	external	access only Owner	No Issue
3	owner	read	Passed	No Issue
4	onlyOwner	modifier	Passed	No Issue
5	renounceOwnership	write	access only Owner	No Issue
6	transferOwnership	write	access only Owner	No Issue
7	_transferOwnership	internal	Passed	No Issue

StratReduceReserveLP.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	reduceReserve	external	access only Owner	No Issue
8	swap	internal	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 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, that can't have 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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Critical operation lacks event log:**- HonorDaoStaking.sol**

Missing event log for: emergencyWithdraw

Resolution: Write an event log for listed events.

(2) Function input parameters lack of check: **- HonorDaoStaking.sol**

Variable validation is not performed in below functions:

- addReward =_rewardsToken
- mint = user

Resolution: We advise to put validation like integer type variables should be greater than 0 and address type variables should not be address(0).

Very Low / Informational / Best practices:

(1) Unlimited Minting: **- XToken.sol**

Minter can mint unlimited tokens.

Resolution: We suggest putting a minting limit.

(2) Division before multiplication: **HonorDaoStaking.sol**

```
// Mint new tokens
// Minted tokens receive rewards normally but incur a 50% penalty when
// withdrawn before lockDuration has passed.
function mint(address user, uint256 amount) external updateReward(user) {
    require(minters[msg.sender], "MultiFeeDistribution::mint: Only minters allowed");
    totalSupply = totalSupply.add(amount);
    Balances storage bal = balances[user];
    bal.total = bal.total.add(amount);
    bal.earned = bal.earned.add(amount);
    uint256 unlockTime = block.timestamp.div(rewardsDuration).mul(rewardsDuration).add
    LockedBalance[] storage earnings = userEarnings[user];
```

Solidity being resource constraint language, dividing any amount and then multiplying will cause discrepancy in the outcome. Therefore always multiply the amount first and then divide it.

Resolution: Consider ordering multiplication before division.

Centralization

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

- toggle: Pool owner can Turn on / off minting and redemption.
- setCollateralRatioOptions: Pool owner can configure variables related to Collateral Ratio.
- toggleCollateralRatio: Pool Owner can pause or unpause collateral ratio updates.
- setFees: Pool owners can set the protocol fees.
- setMinCollateralRatio: Pool owner can set the minimum Collateral Ratio.
- reduceExcessCollateral: Pool owner can transfer the excess balance of WETH to FeeReserve.
- setSwapStrategy: Pool owner can set the address of Swapper utils.
- setOracle: Pool owner can set new oracle address.
- setYTokenSlippage: Pool owner can set yTokenSlipage.
- setTreasury: Pool owner can set the address of the Treasury.
- cleanDust: SwapStrategyPOL owner can clean dust.
- changeSlippage: SwapStrategyPOL owner can change slippage value.
- add: HonorDaoChef owner can add a new LP to the pool.
- set: HonorDaoChef owner can update the given pool's reward allocation point and `IRewarder` contract.
- setRewardPerSecond: HonorDaoChef owner can set the reward per second to be distributed.
- setRewardMinter: HonorDaoChef owner can set the address of rewardMinter.
- setNftController: HonorDaoChef owner can set NFT controller address.
- setNftBoostRate: HonorDaoChef owner can set NFT Boost Rate value.
- addReward: HonorDaoStaking owner can add a new reward token to be distributed to stakers.
- approveRewardDistributor: HonorDaoStaking owner can modify approval for an address to call notifyRewardAmount.
- recoverERC20:HonorDaoStaking owner can be added to support recovering LP Rewards from other systems such as BAL to be distributed to holders.

- setTeamWalletAddress:HonorDaoStaking owner can set the address of the team wallet.
- setTeamRewardPercent:HonorDaoStaking owner can set percent of team reward.
- addZap: HonorDaoZapMMSwap owner can add new zap configuration.
- removeZap: HonorDaoZapMMSwap owner can deactivate a Zap configuration.
- setWhitelist: NFTController owner can set whitelist address,
- setDefaultBoostRate: NFTController owner can set default boost rate value.
- setBoostRate: NFTController owner can set boost rate value.
- transfer: Fund owners can transfer tokens.
- transfer: HONORReserve::transfer owner can only allow funds to withdraw.
- setPeriod: UniswapPairOracle owner can set maximum and minimum period.
- setMinter: XToken minter can set minter for XToken.
- mint: XToken minter can mint new XToken.
- OpenTrade: HONOR owners can trade openly.
- includeToWhitelist: HONOR owner can include address to whitelist.
- excludeFromWhitlist: HONOR owner can exclude address to whitelist.
- OpenTrade: XNIGHT owners can trade openly.
- includeToWhitelist: XNIGHT owner can include address to whitelist.
- excludeFromWhitlist: XNIGHT owner can exclude address to whitelist.
- addStrategy: HonorDaoTreasury owner can add new strategy.
- removeStrategy: HonorDaoTreasury owner can remove current strategy.
- allocateFee:HonorDaoTreasury owner can allocate protocol's fee to stakers.
- recollateralize: StratRecollateralize owner can recollateralize the minting pool.
- reduceReserve: StratReduceReserveLP owner can remove liquidity, buy back YToken and burn.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the airdrop smart contract once its function is completed.

Conclusion

We were given a contract code in the form of files. And we have used all possible tests based on given objects as files. We have not observed any major issues in the smart contracts. **So, the smart contracts are ready for the mainnet deployment**.

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

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

Security state of the reviewed 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 whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. 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, 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 in accordance with the best industry practices at the date of this report, in relation to: 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).

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

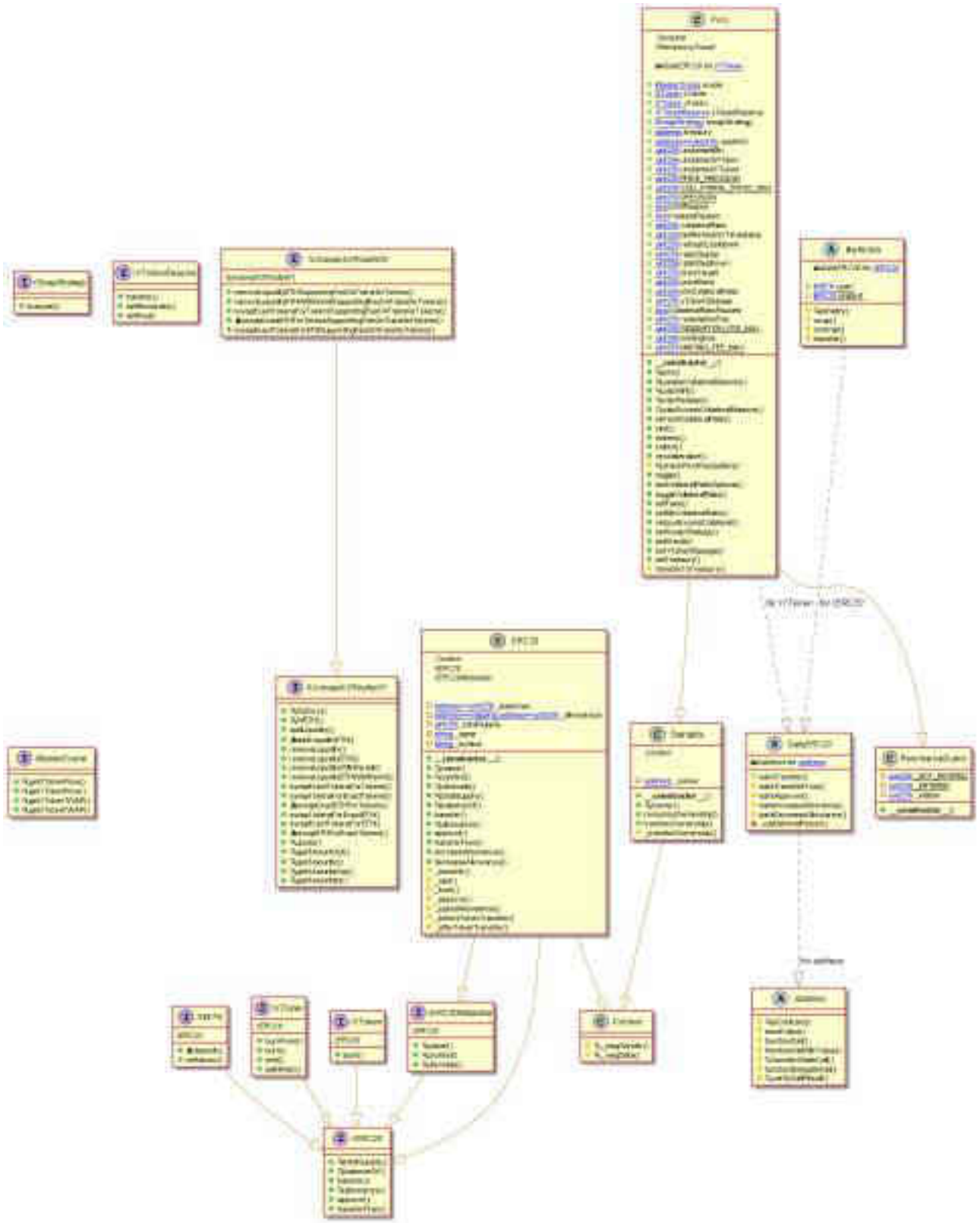
Technical Disclaimer

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 explicit security of the audited smart contracts.

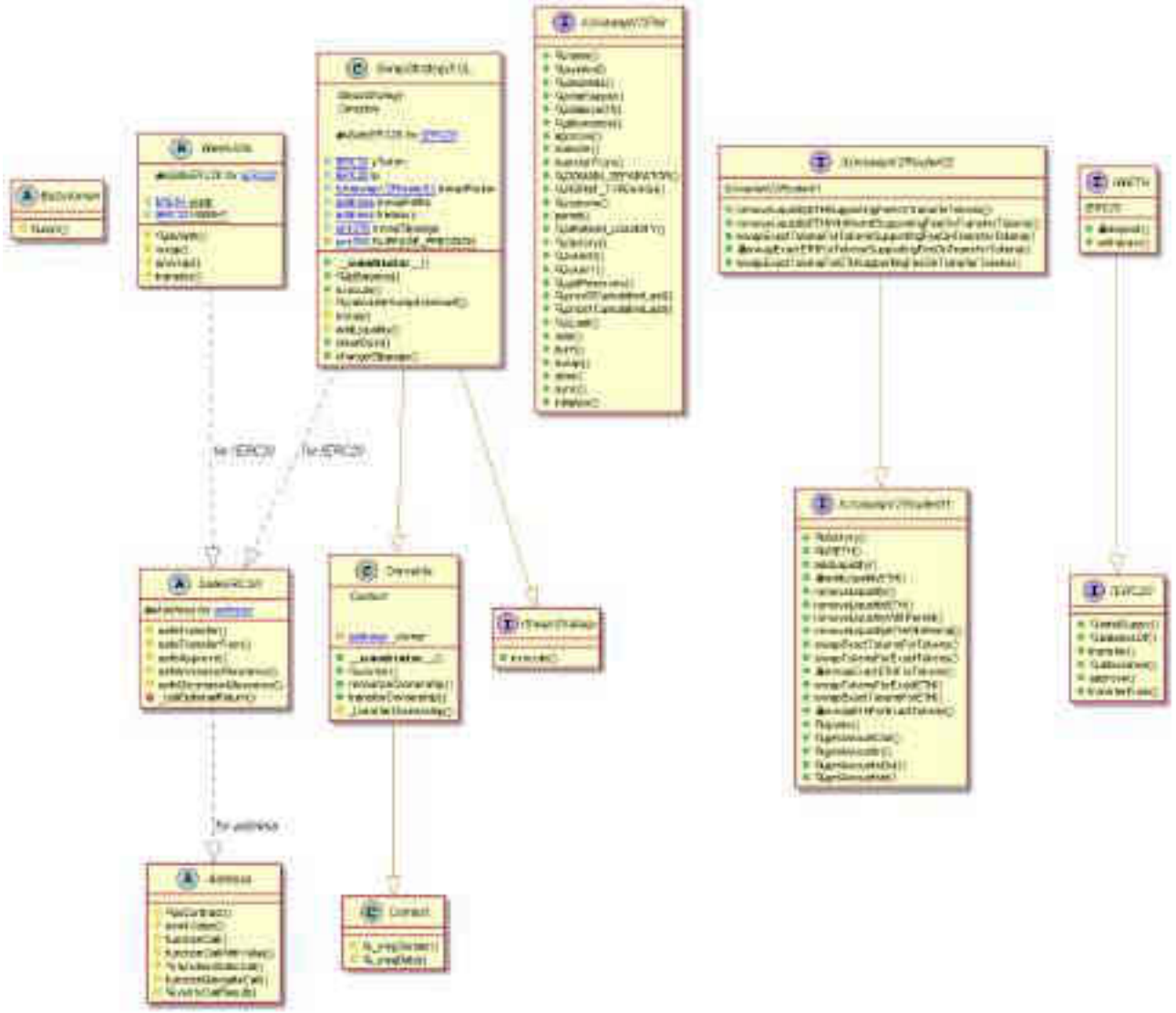
Appendix

Code Flow Diagram - Honor DAO Protocol

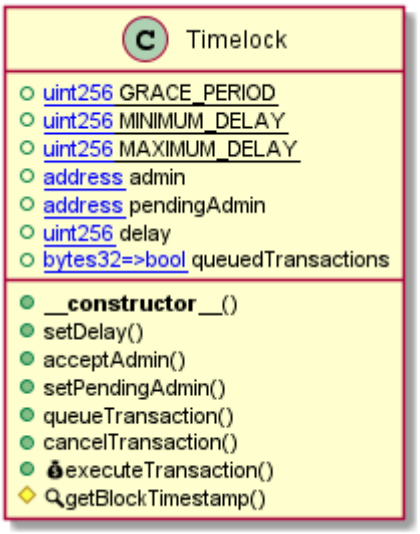
Pool Diagram



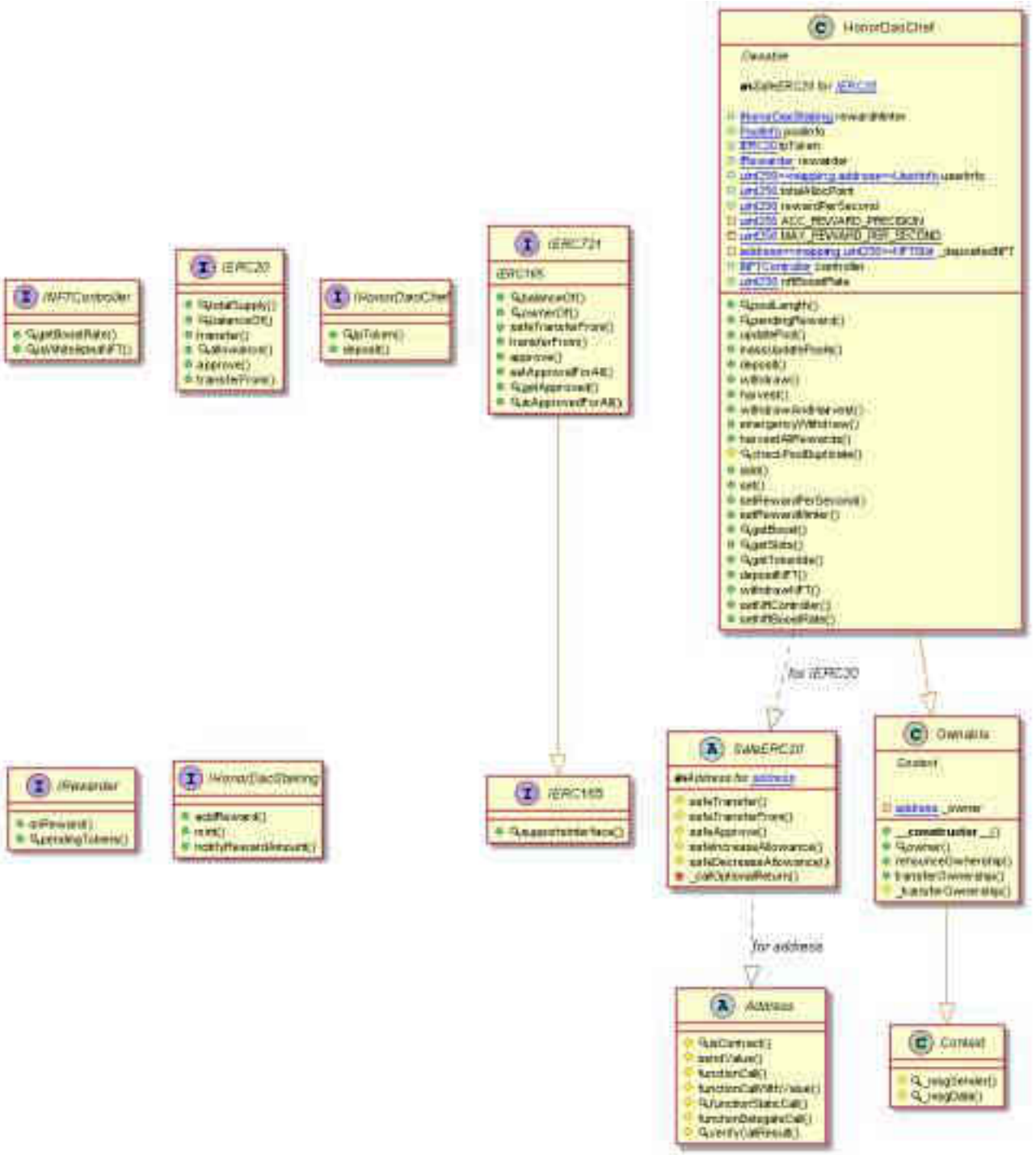
SwapStrategyPOL Diagram



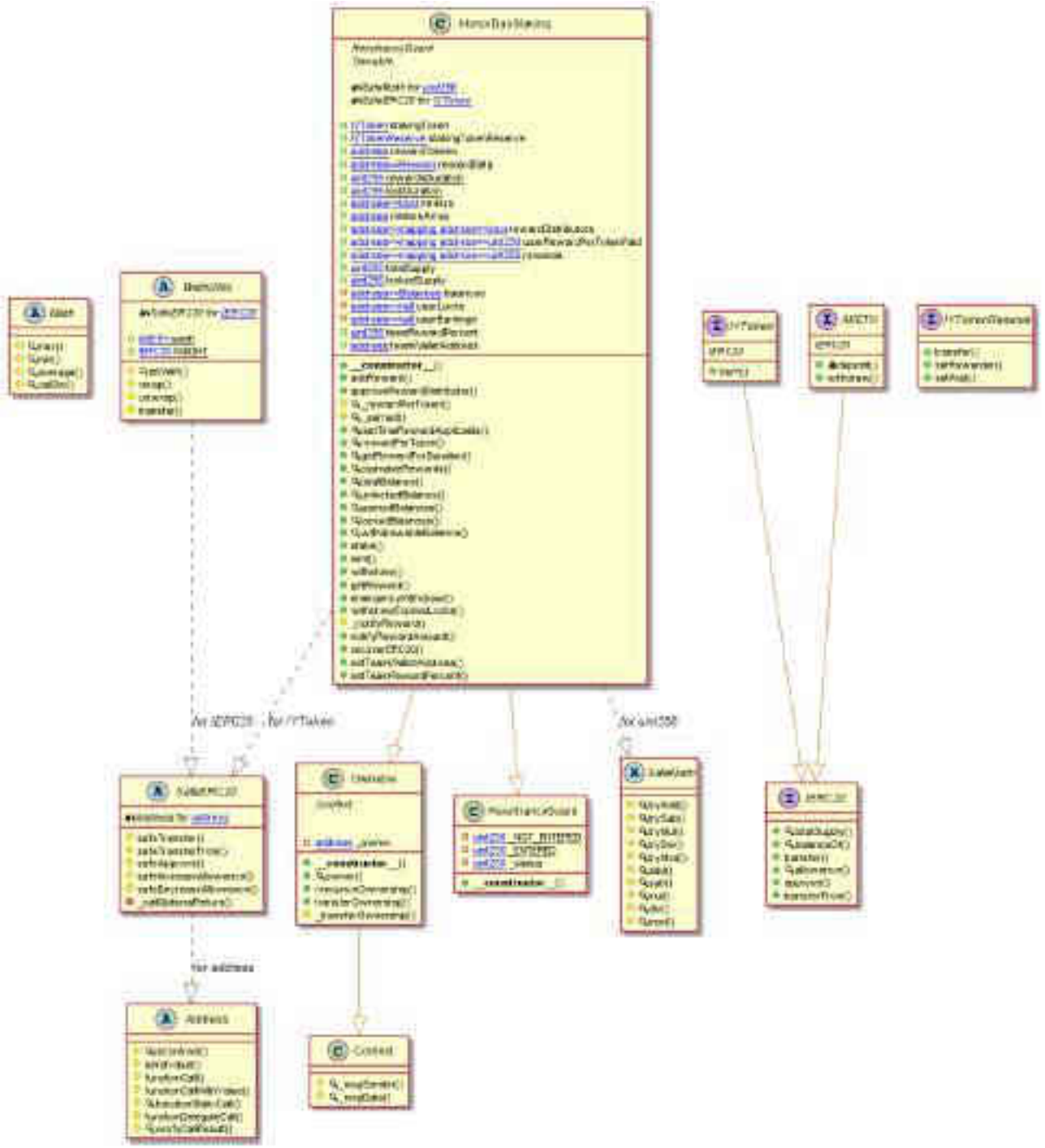
Timelock Diagram



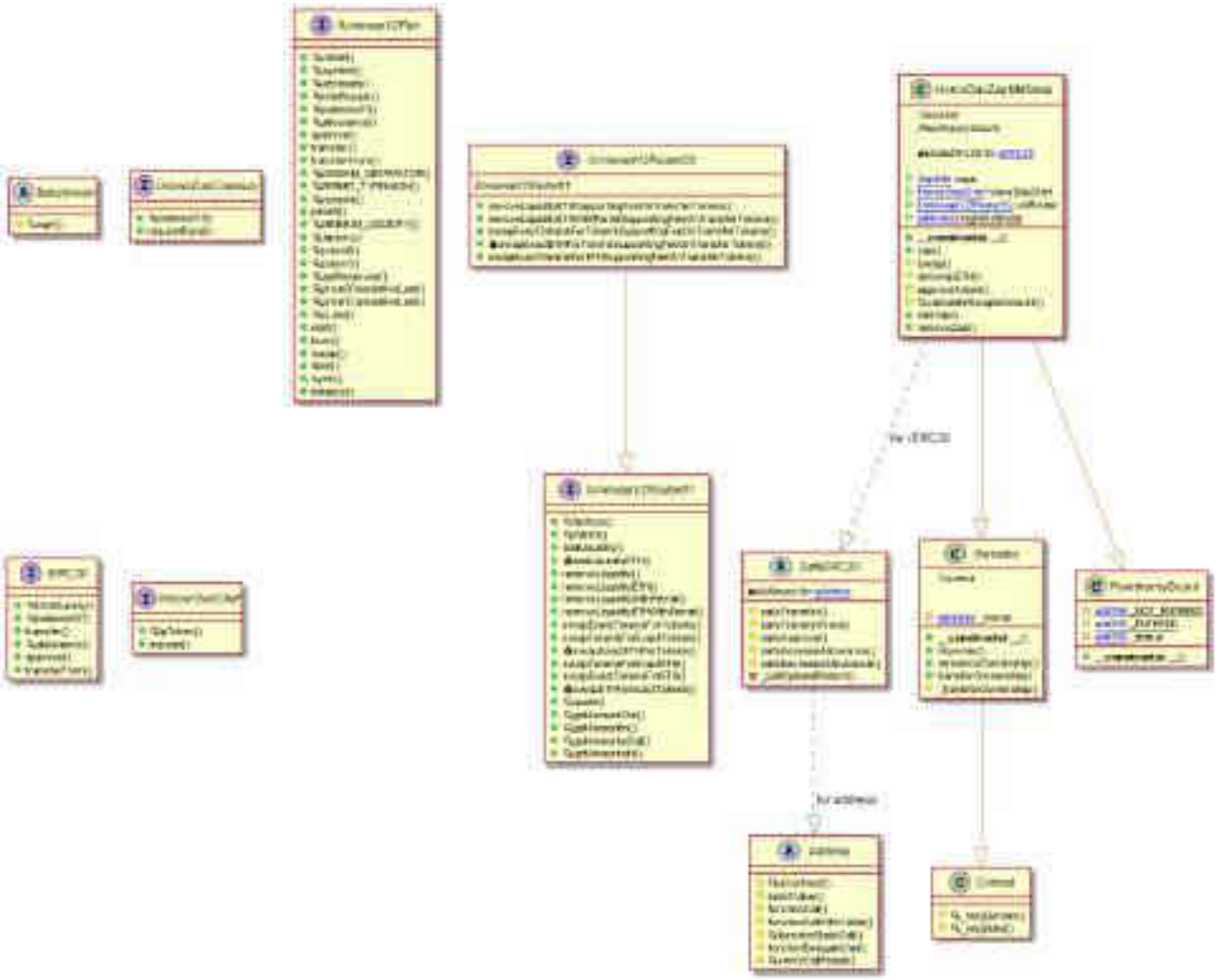
HonorDaoChef Diagram



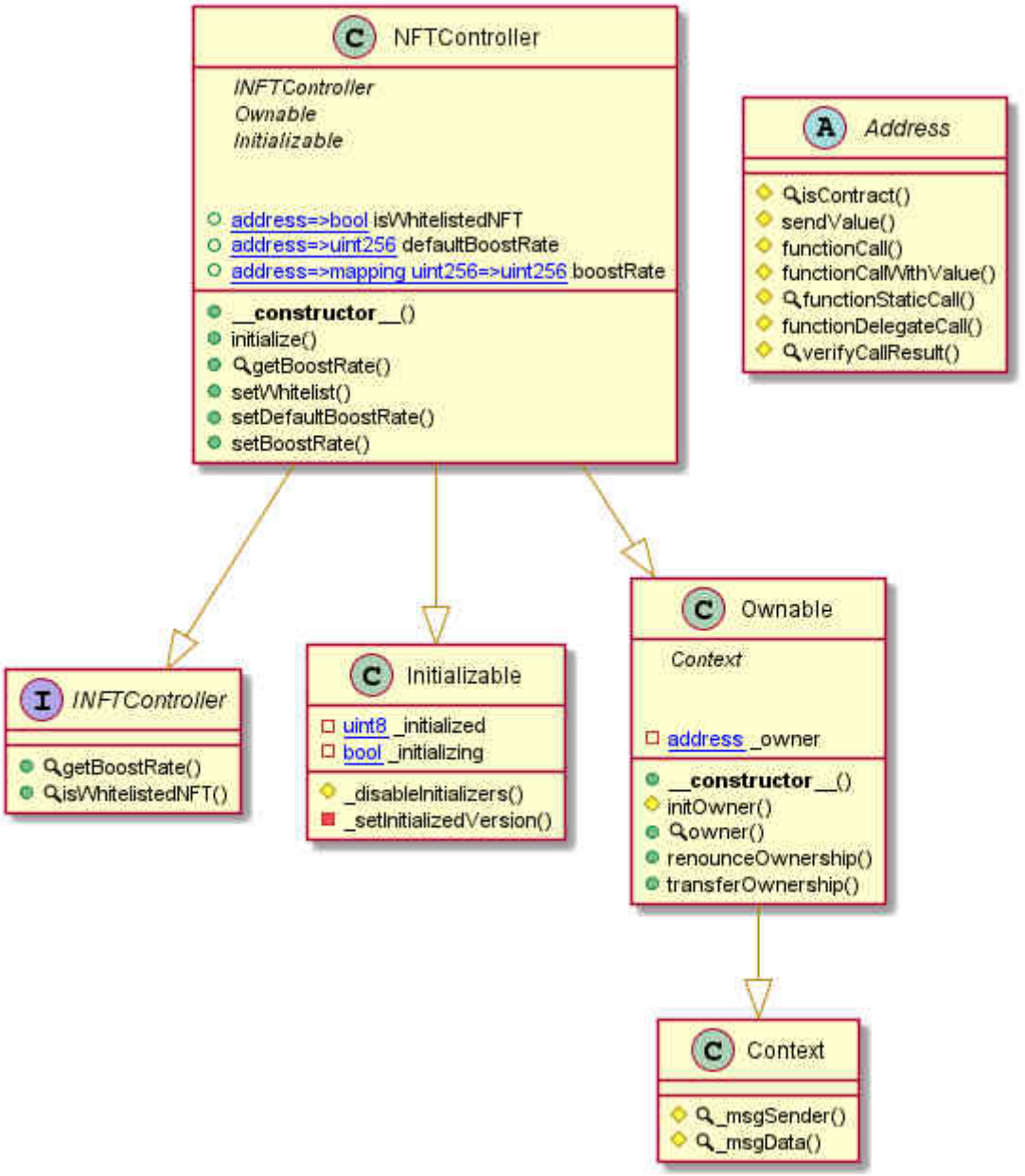
HonorDaoStaking Diagram



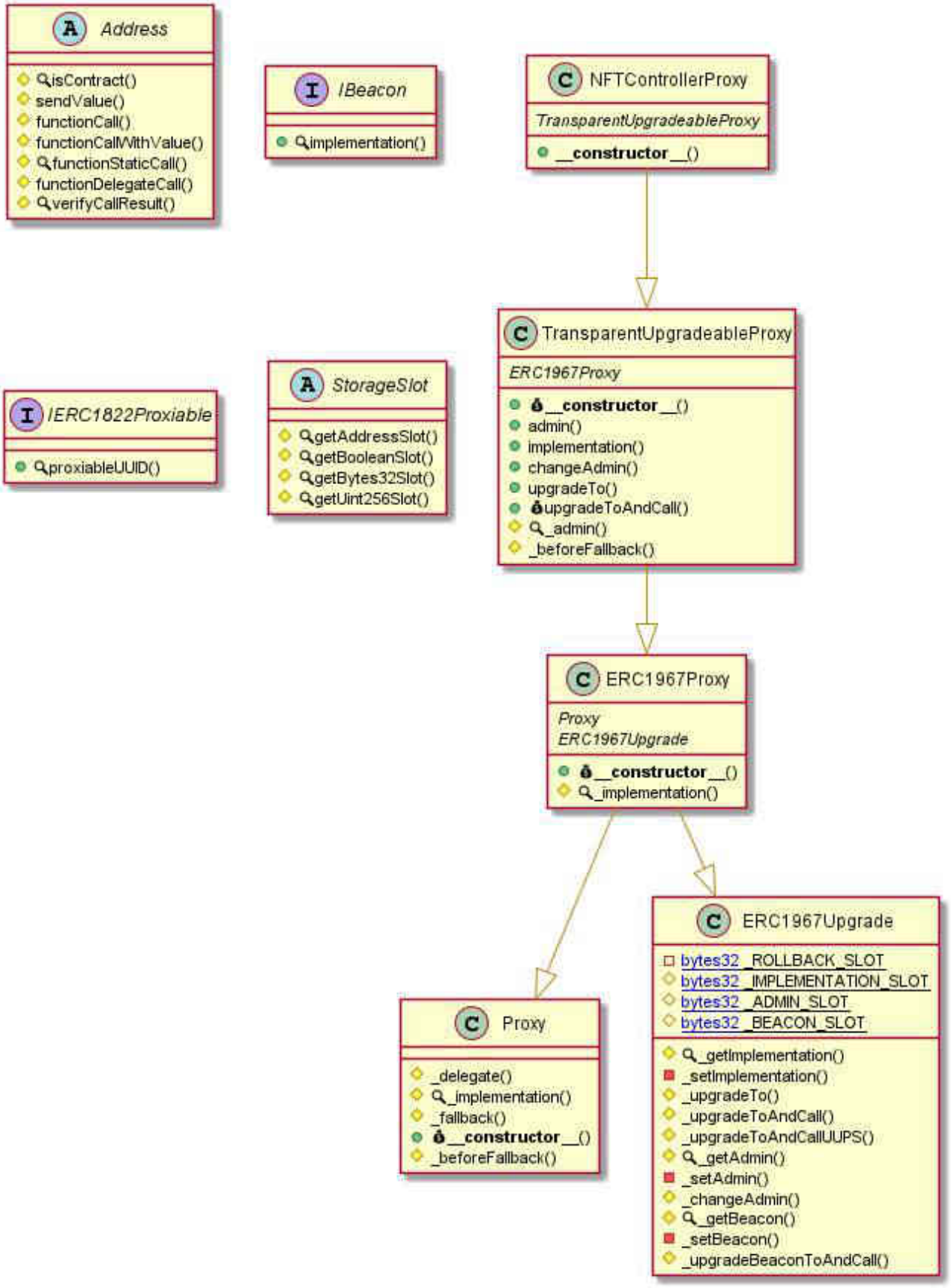
HonorDaoZapMMSwap Diagram



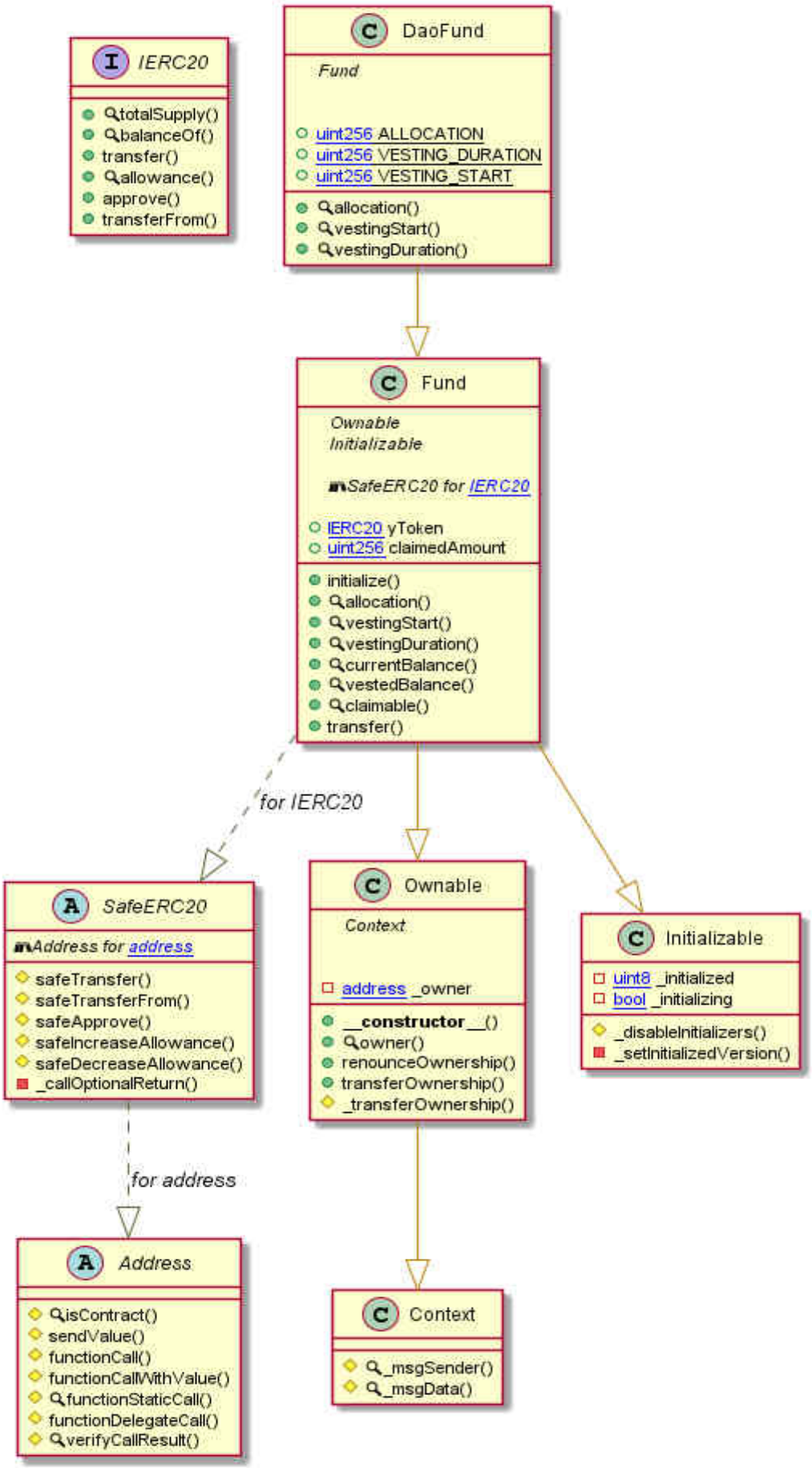
NFTController Diagram



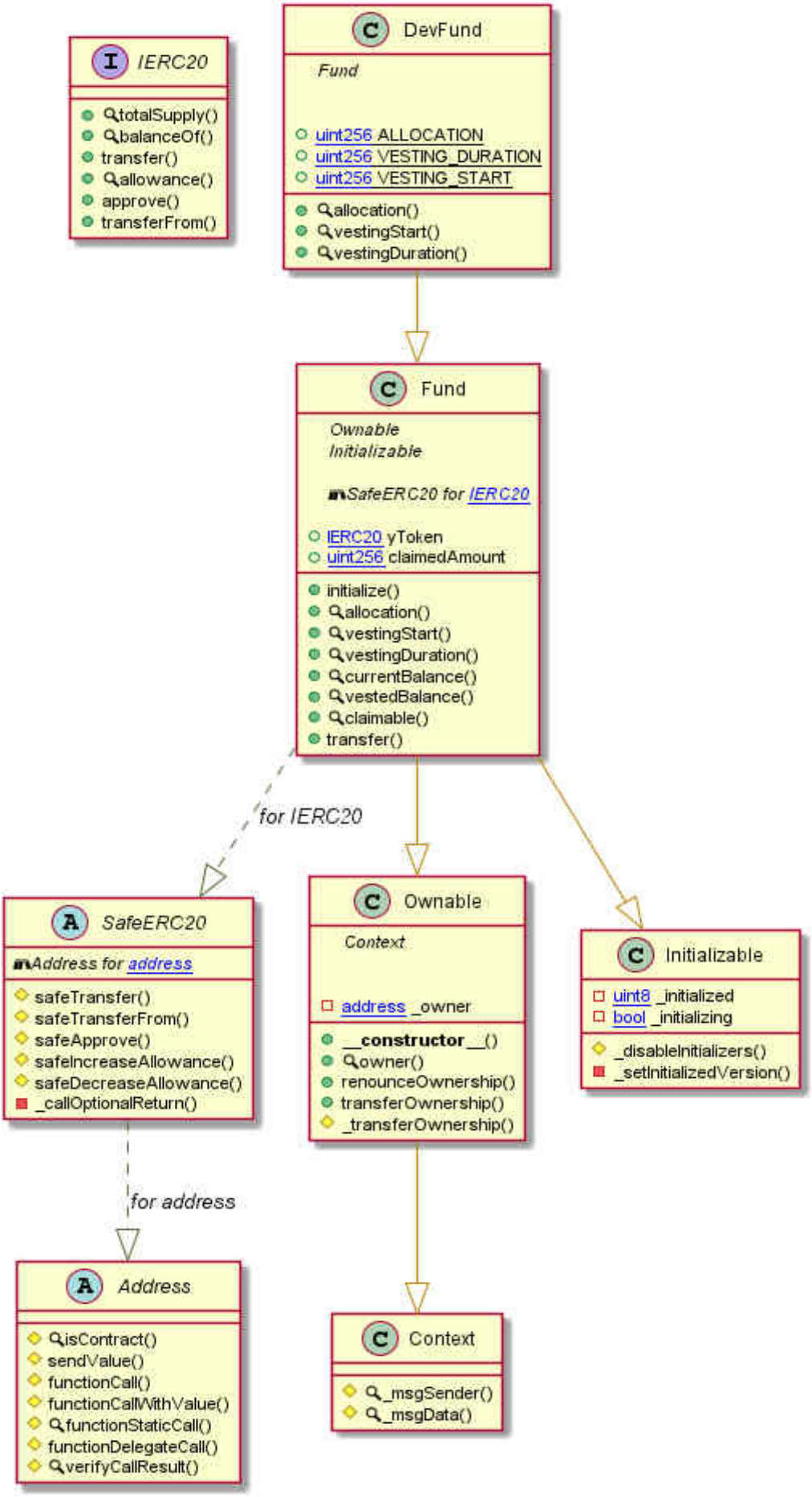
NFTControllerProxy Diagram



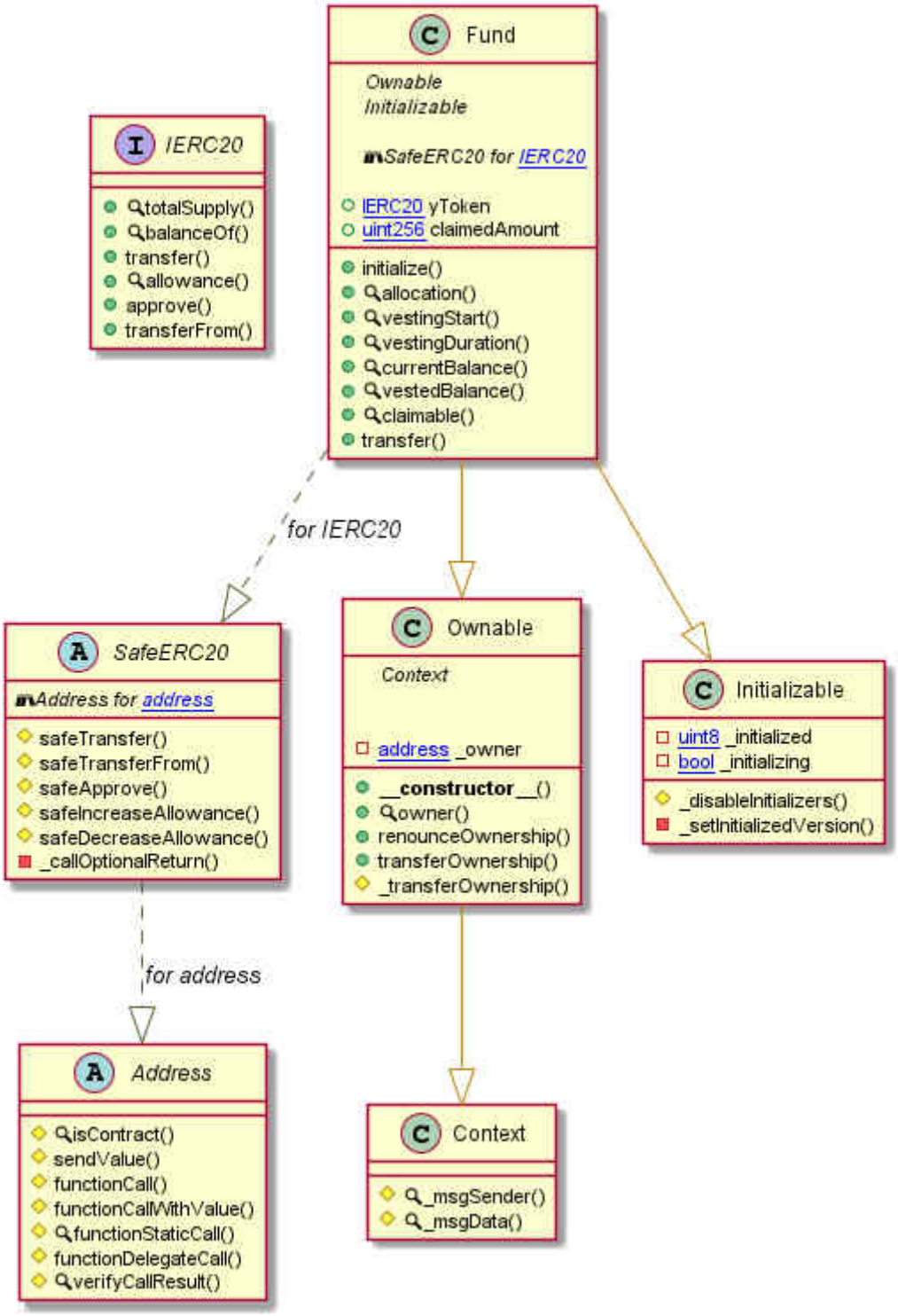
DaoFund Diagram



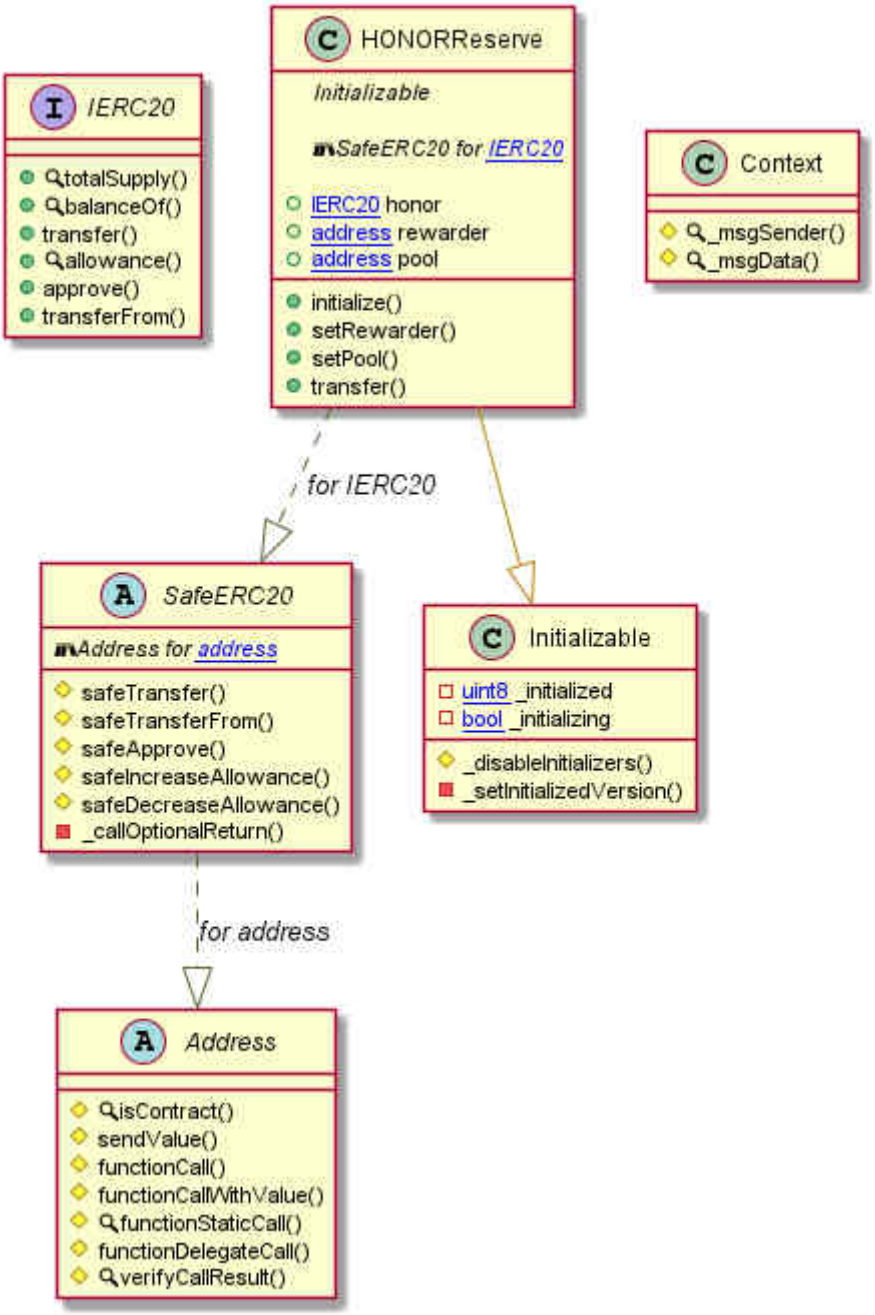
DevFund Diagram



Fund Diagram

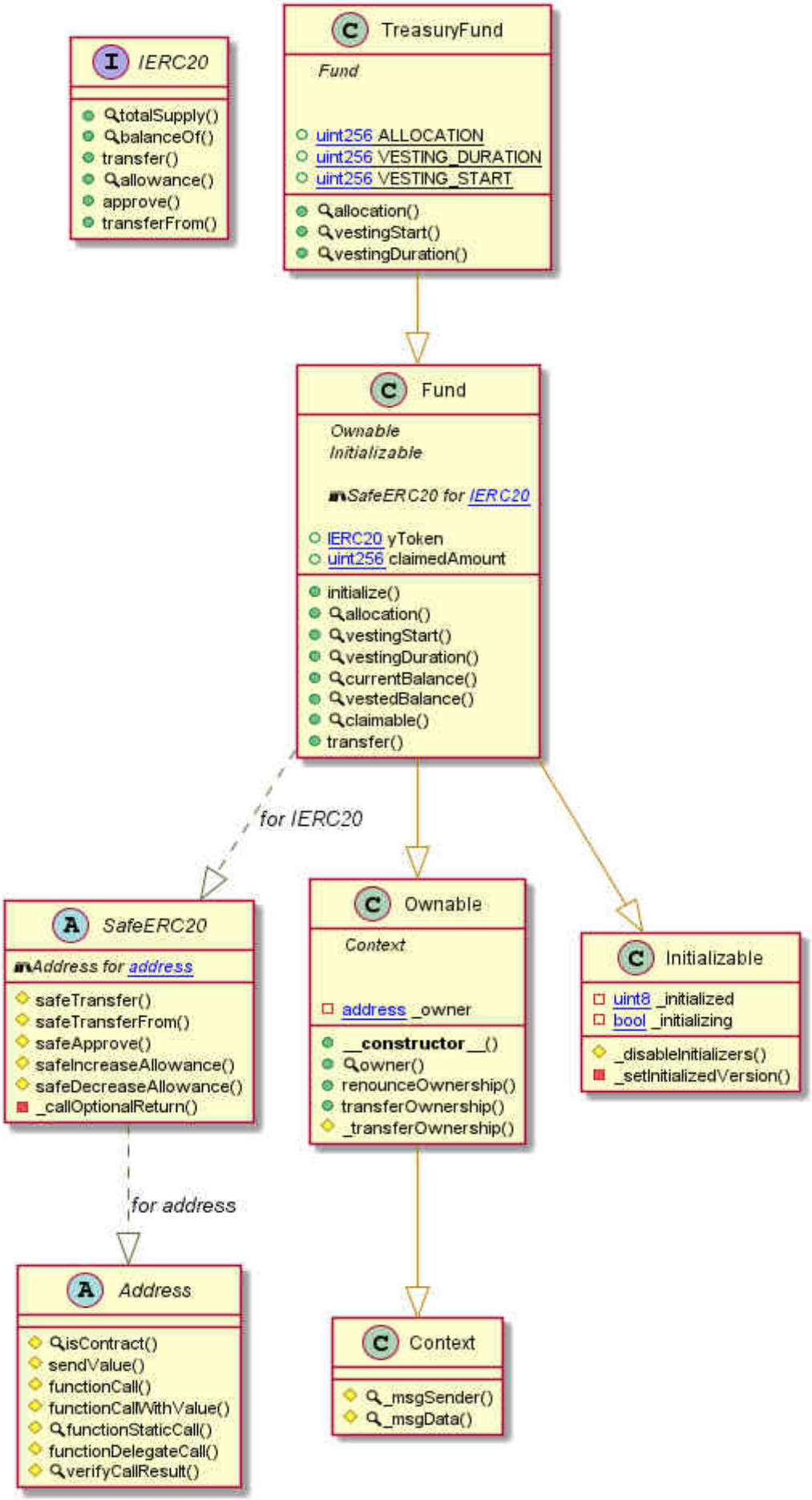


HONORReserve Diagram

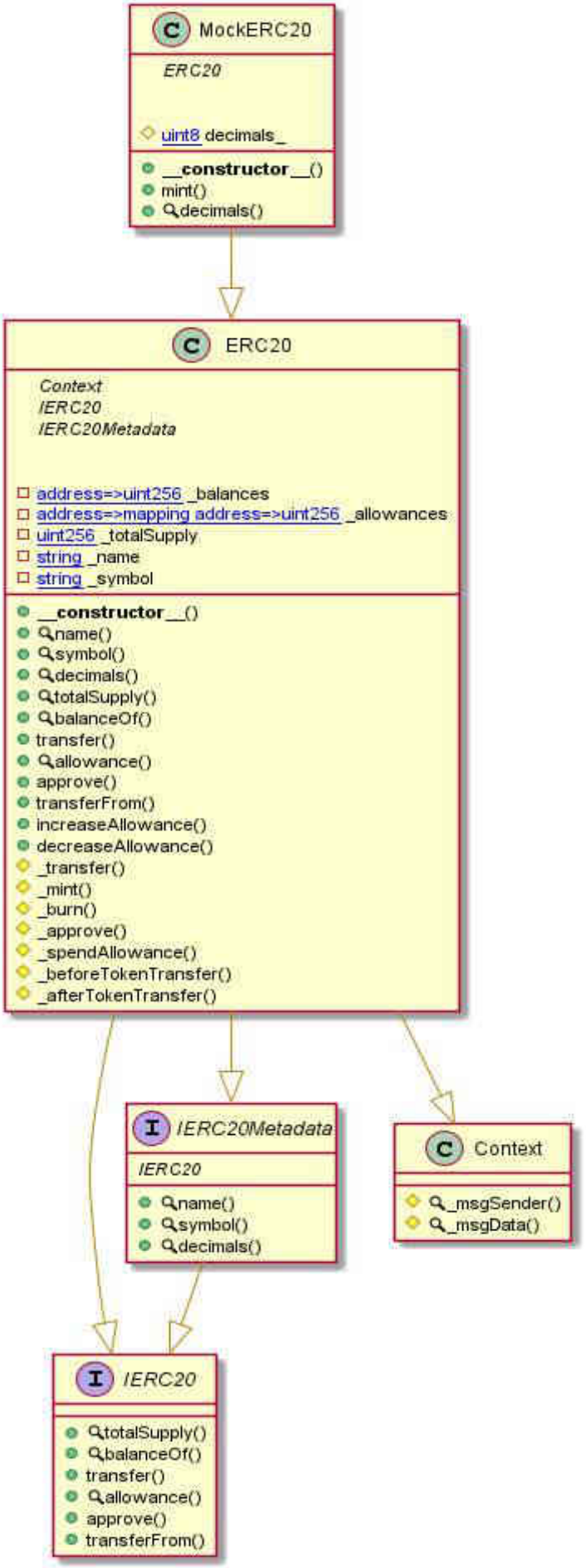


MockTreasury Diagram

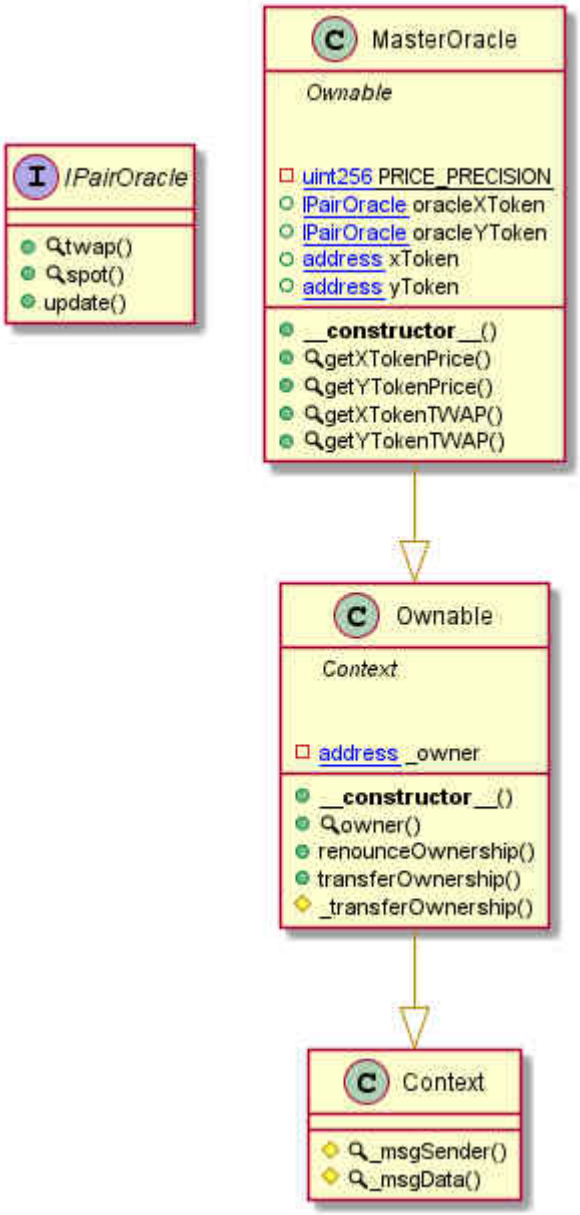
TreasuryFund Diagram



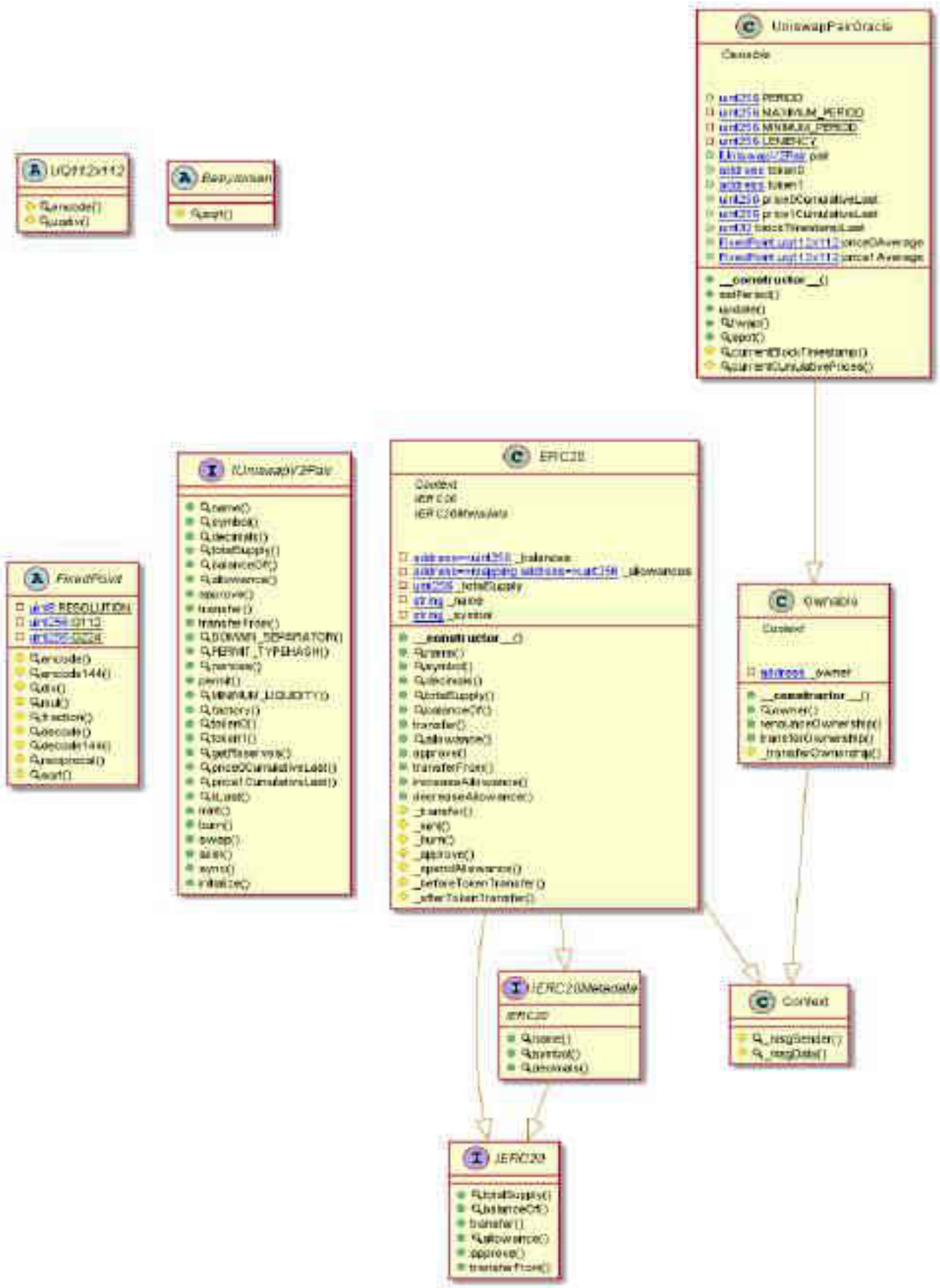
MockERC20 Diagram



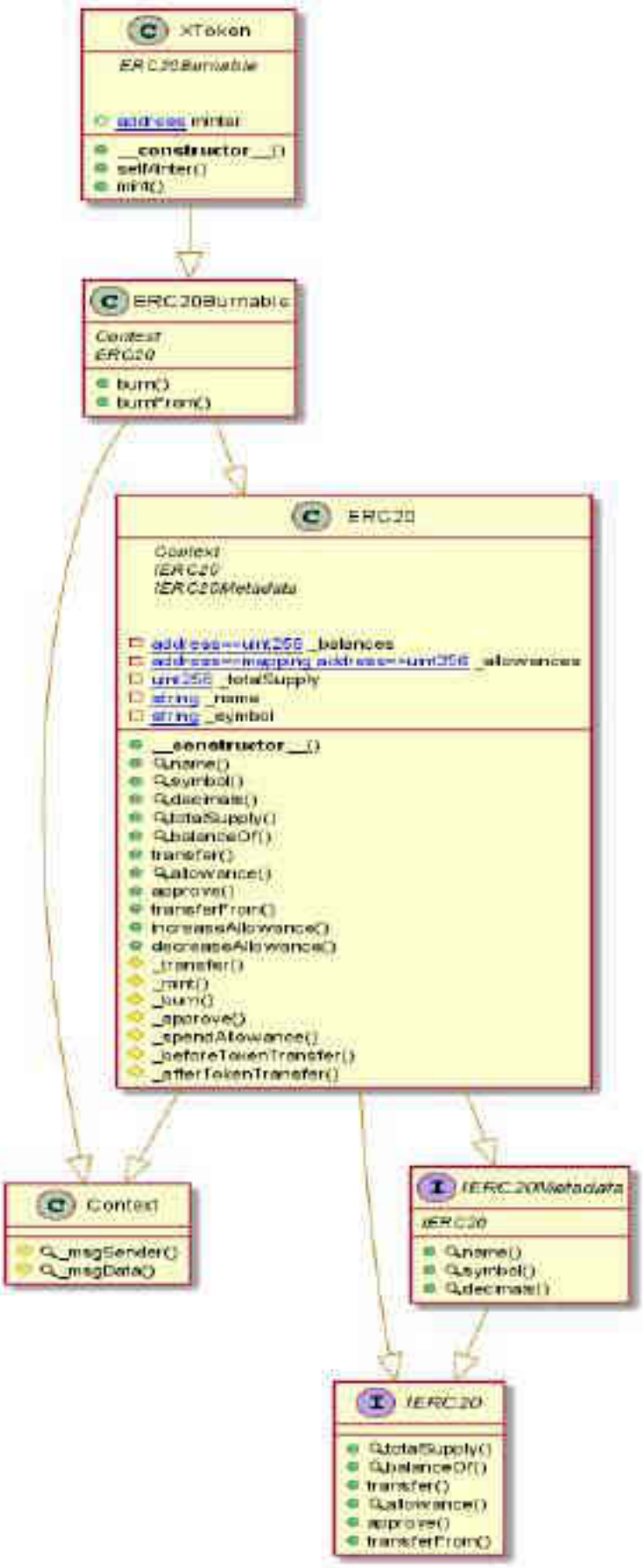
MasterOracle Diagram



UniswapPairOracle Diagram

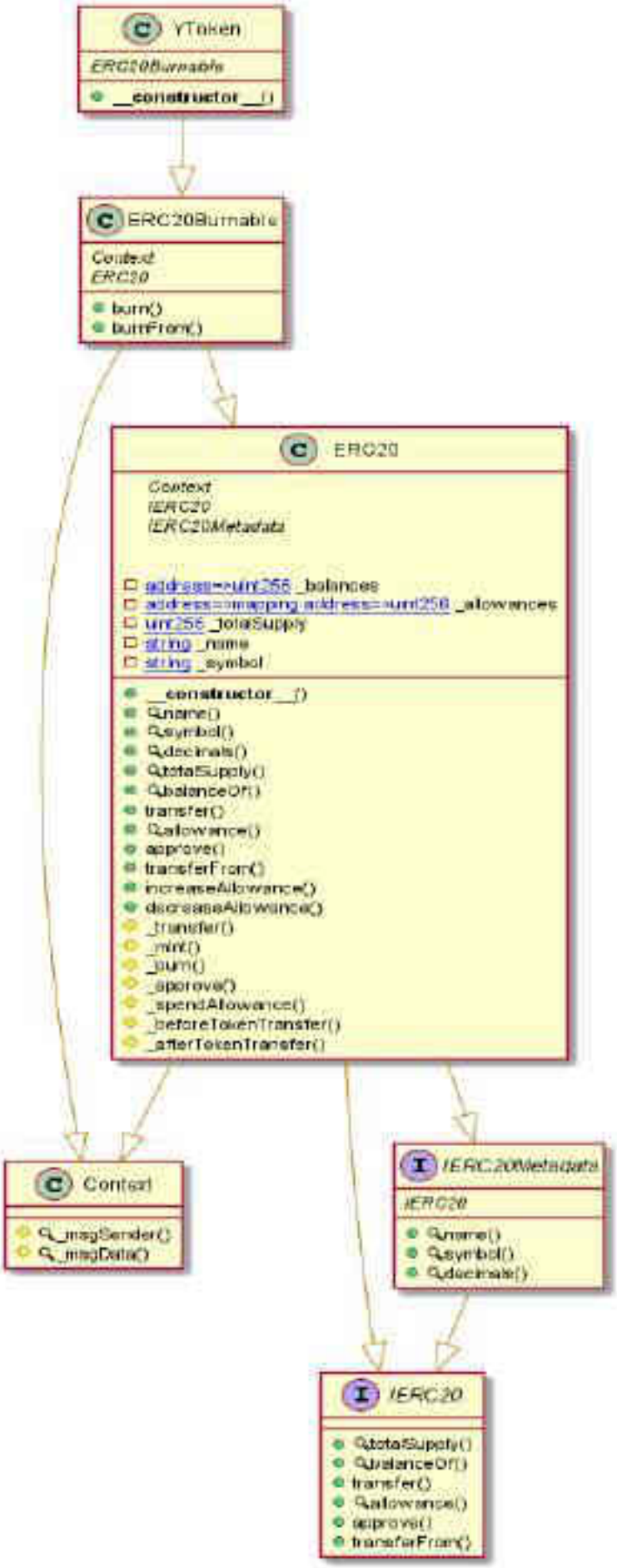


XToken Diagram



Email: audit@EtherAuthority.io

YToken Diagram

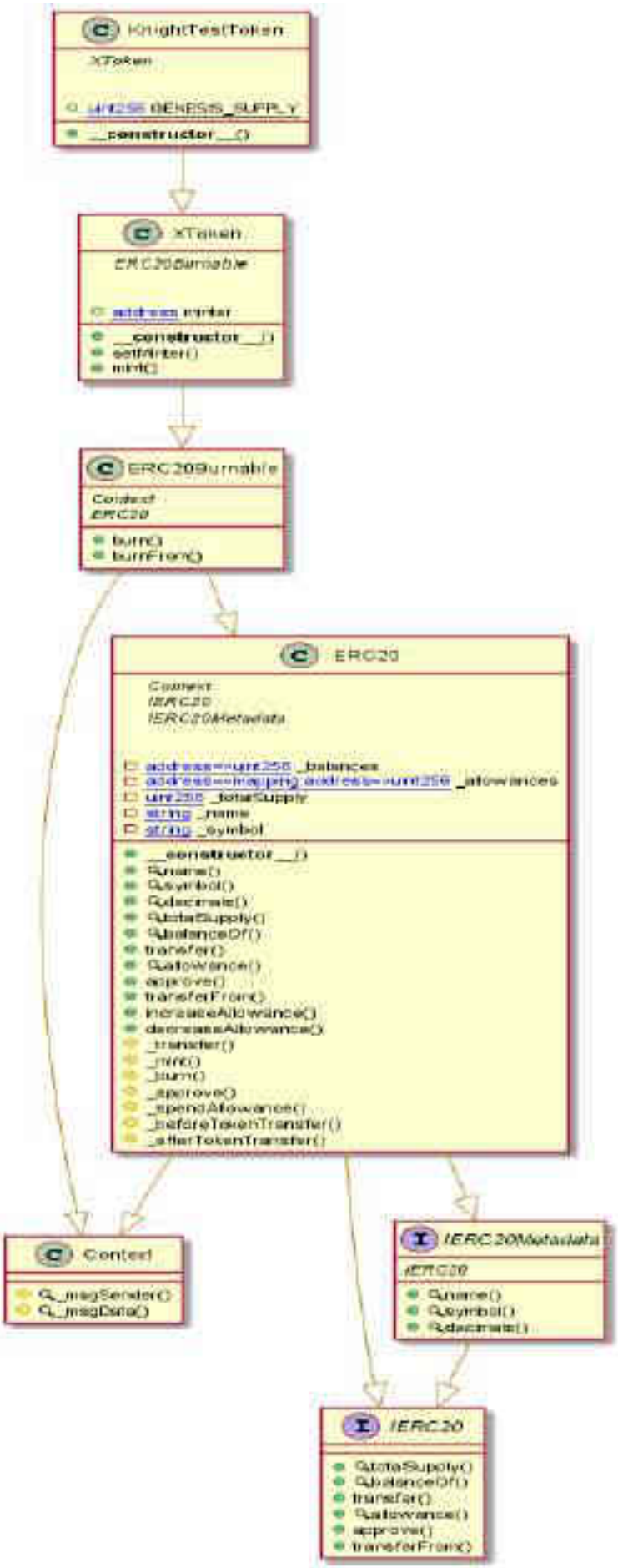


Email: audit@EtherAuthority.io

HONOR Diagram

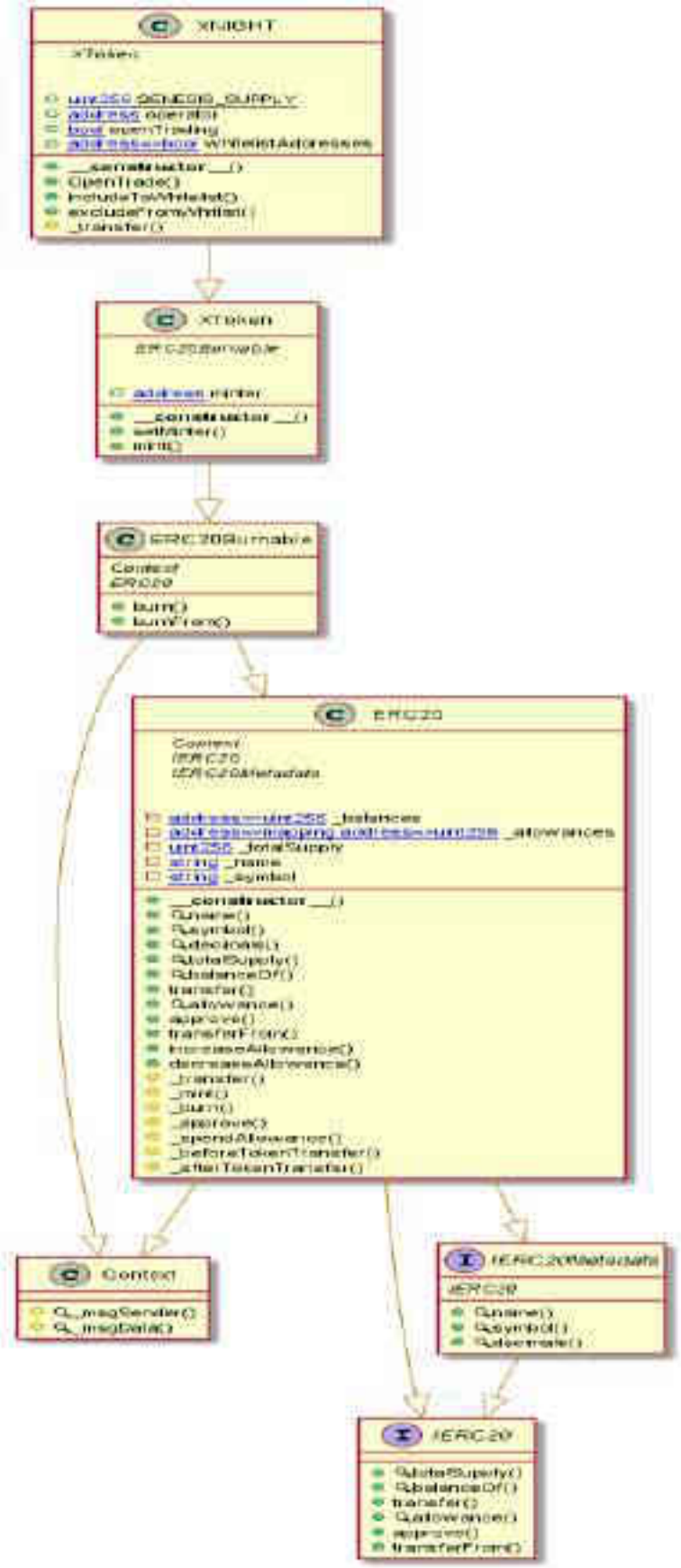
Email: audit@EtherAuthority.io

KNIGHTTestToken Diagram



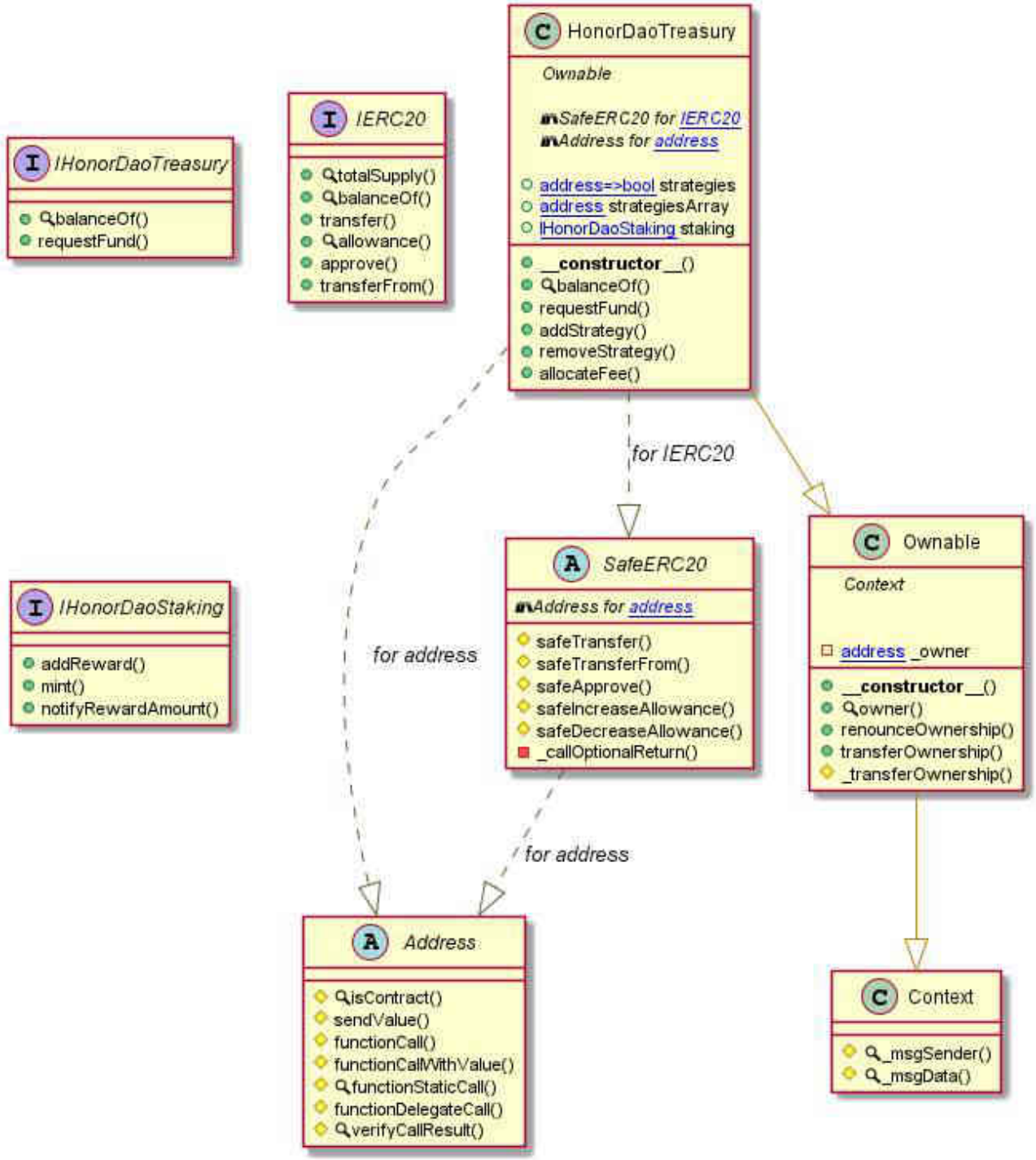
Email: audit@EtherAuthority.io

XNIGHT Diagram

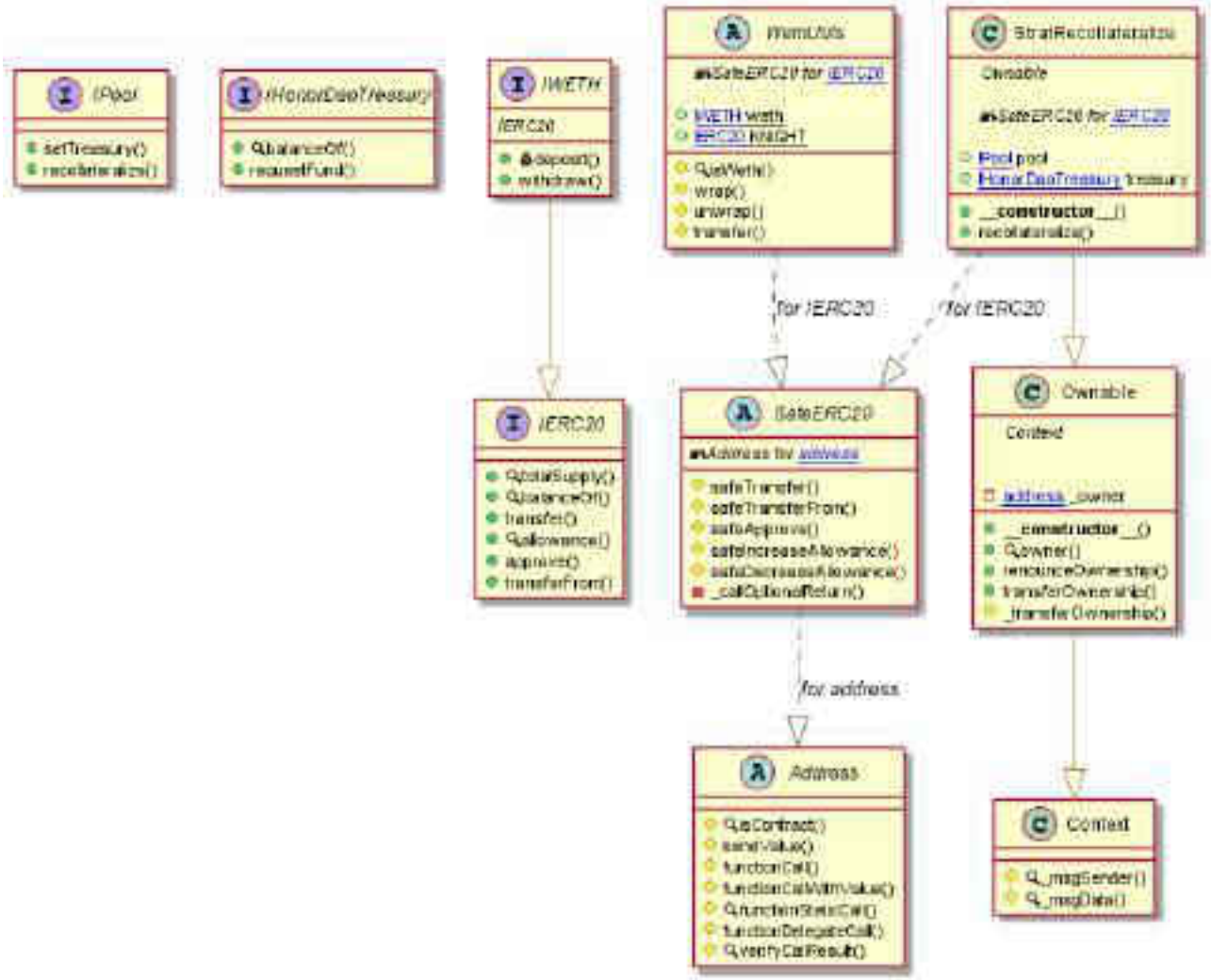


Email: audit@EtherAuthority.io

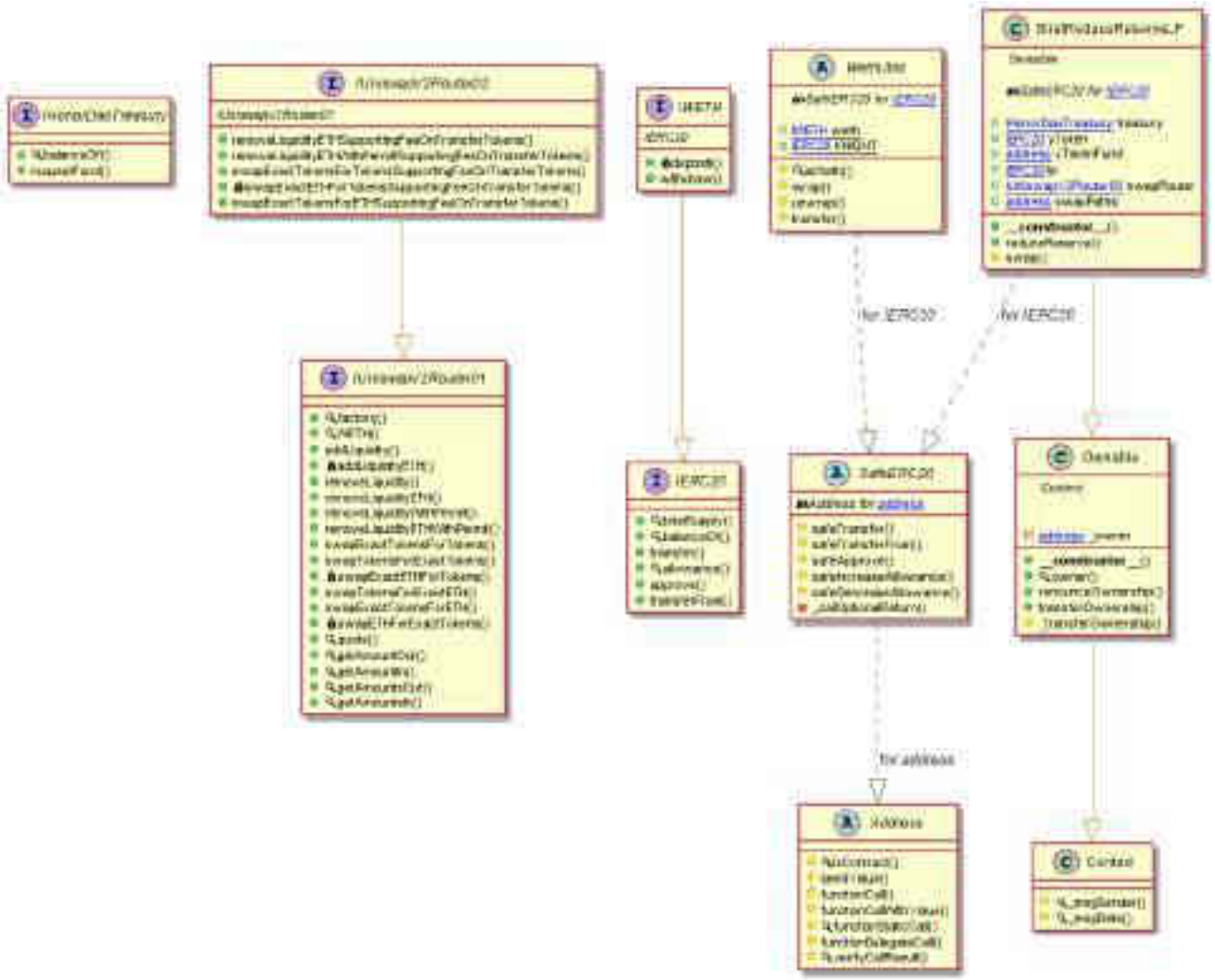
HonorDaoTreasury Diagram



StratRecollateralize Diagram

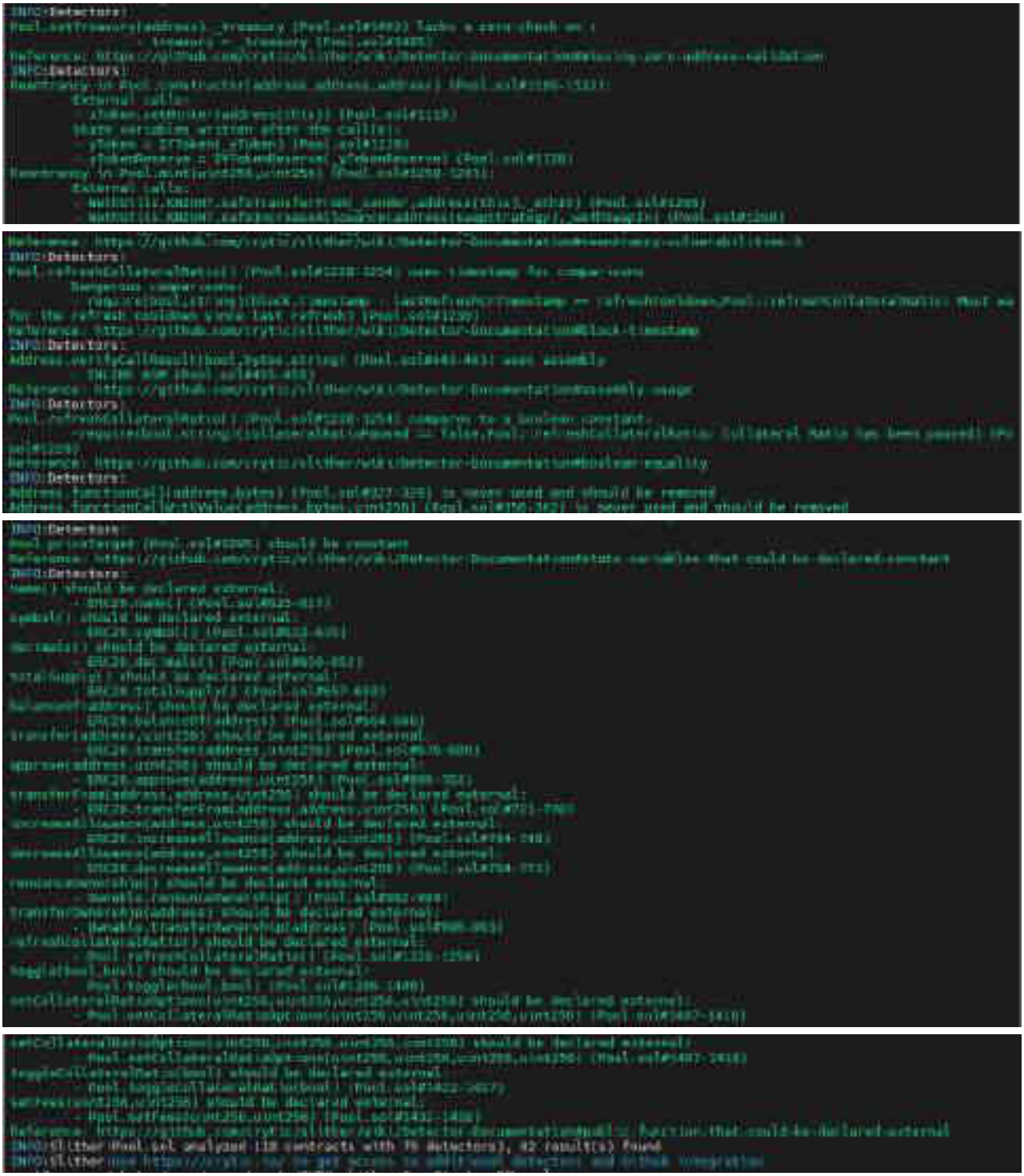


StratReduceReserveLP Diagram

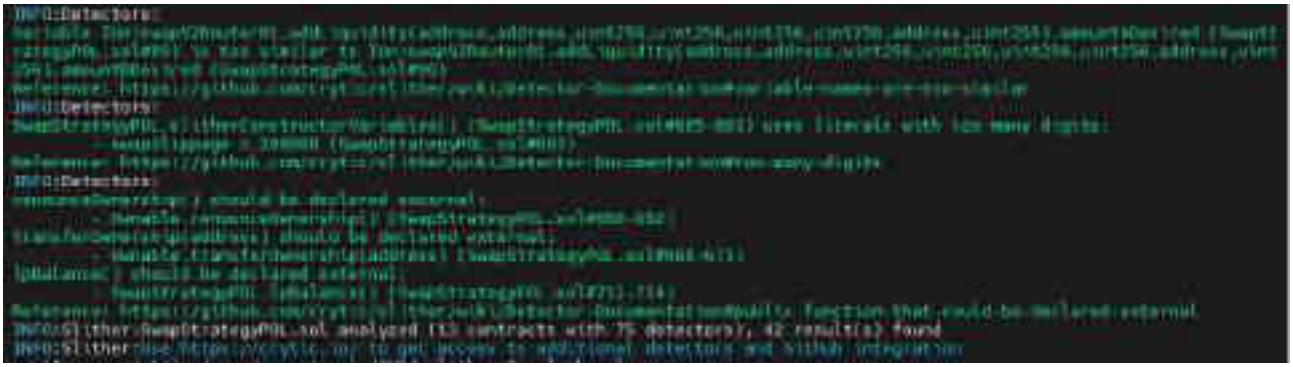


Slither Results Log

Slither log >> Pool.sol



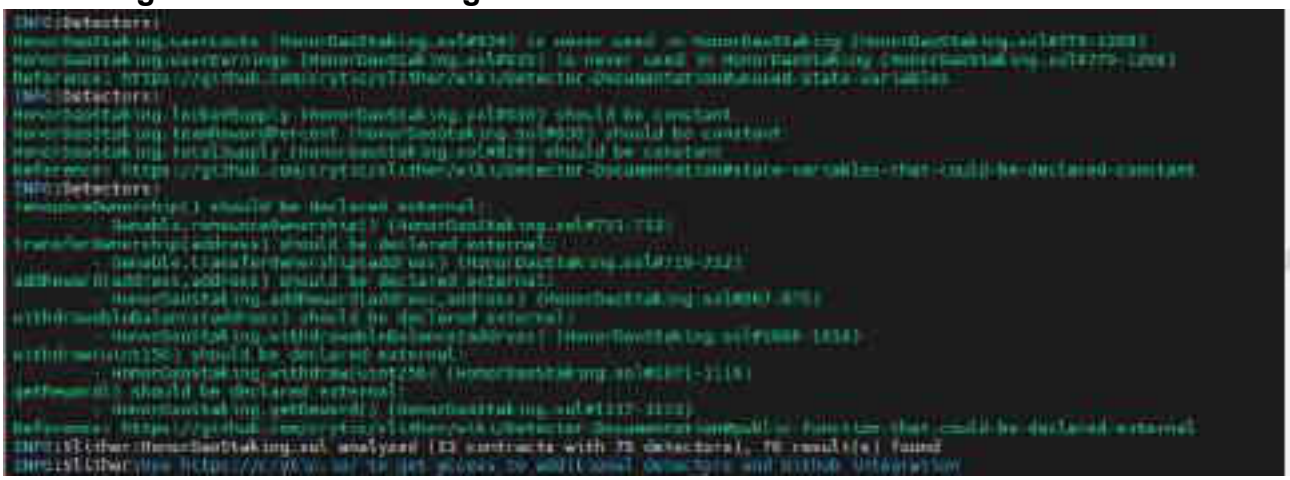
Slither log >> SwapStrategyPOL.sol



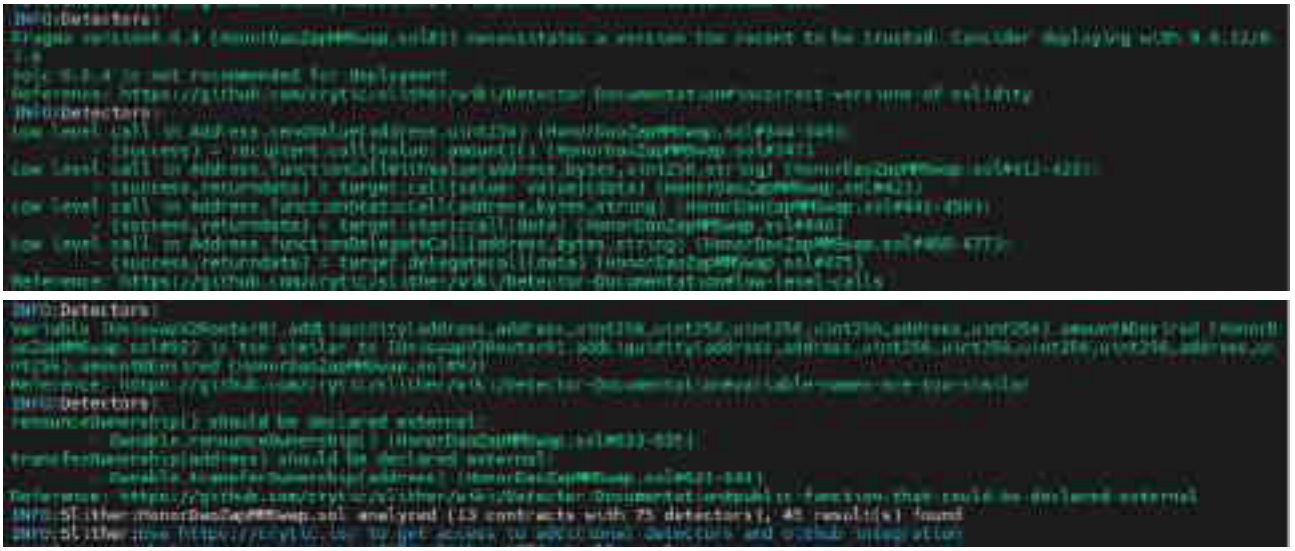
Slither log >> Timelock.sol

Slither log >> HonorDaoChef.sol

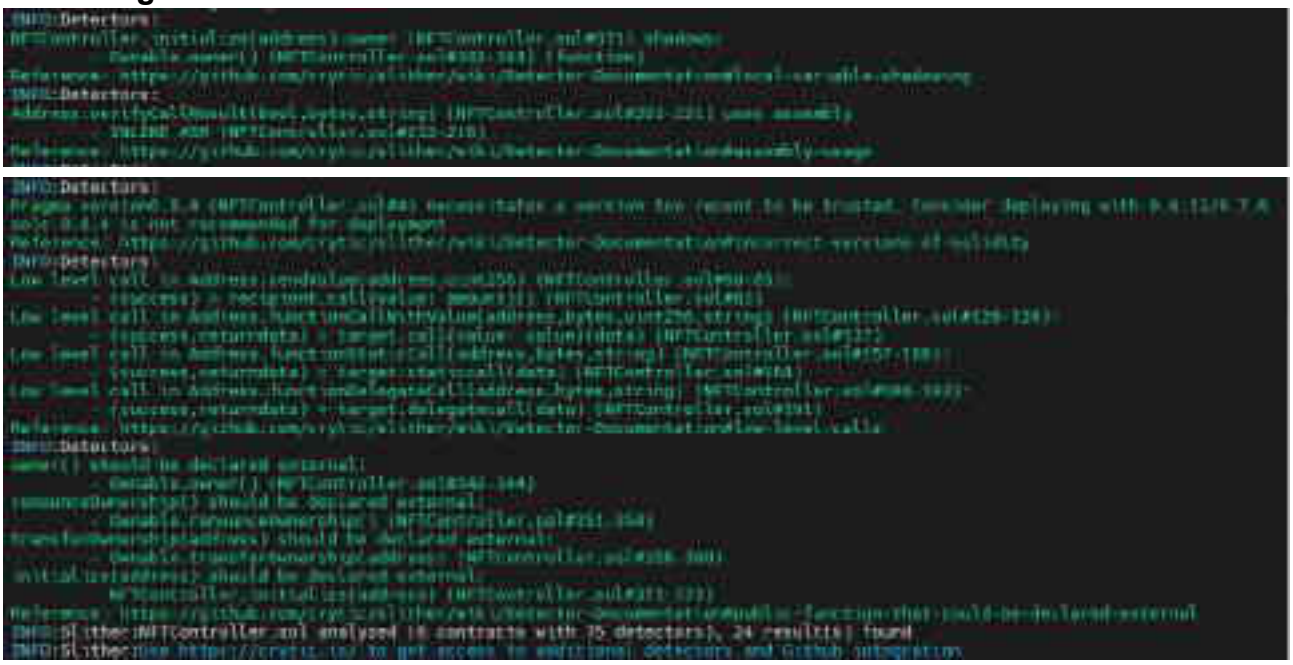
Slither log >> HonorDaoStaking.sol



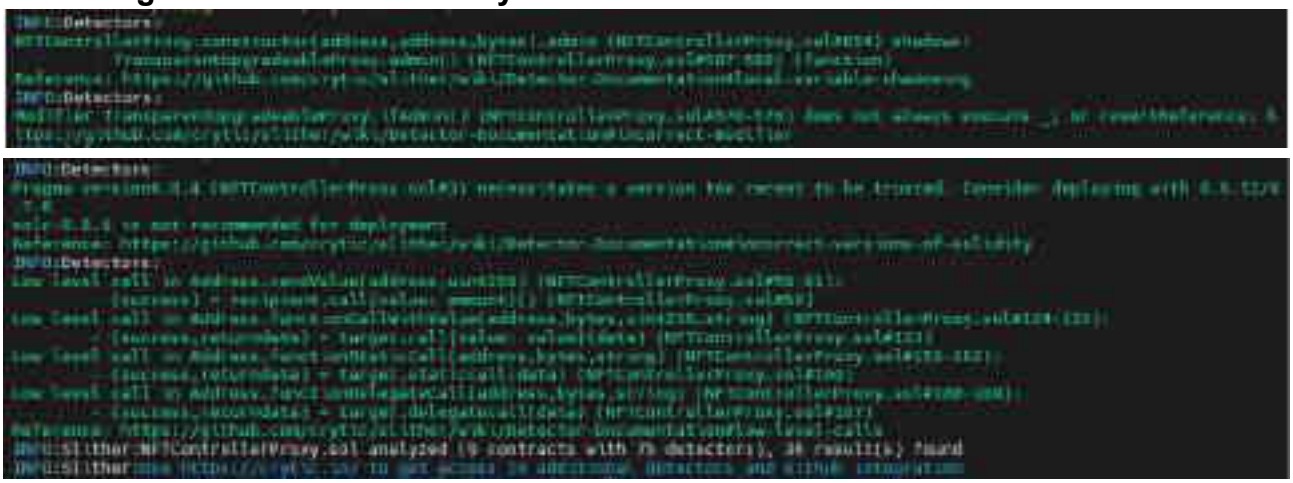
Slither log >> HonorDaoZapMMSwap.sol



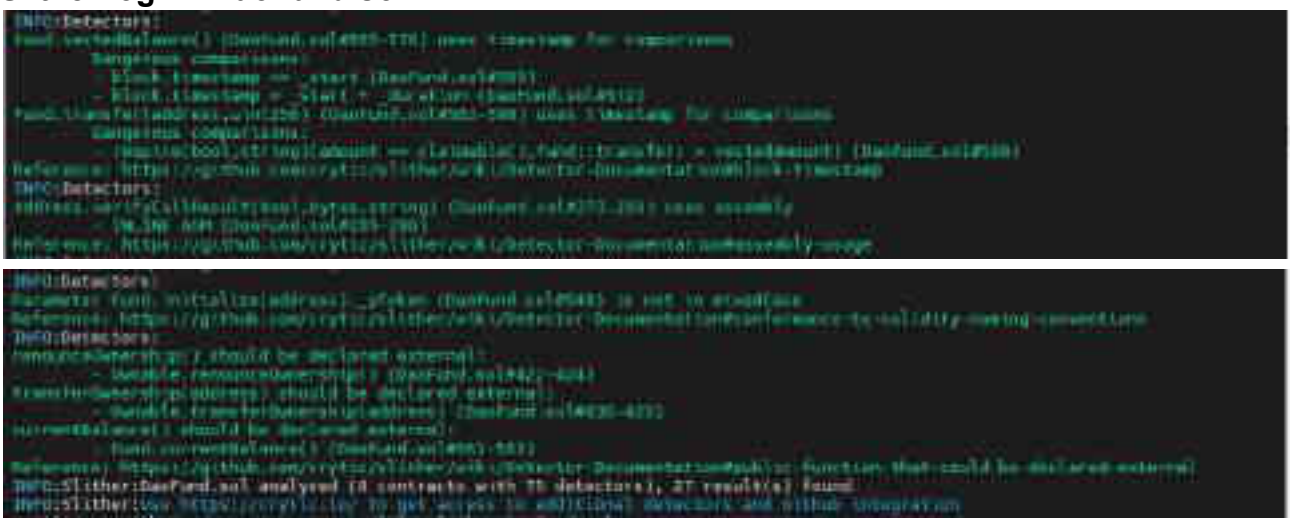
Slither log >> NFTController.sol



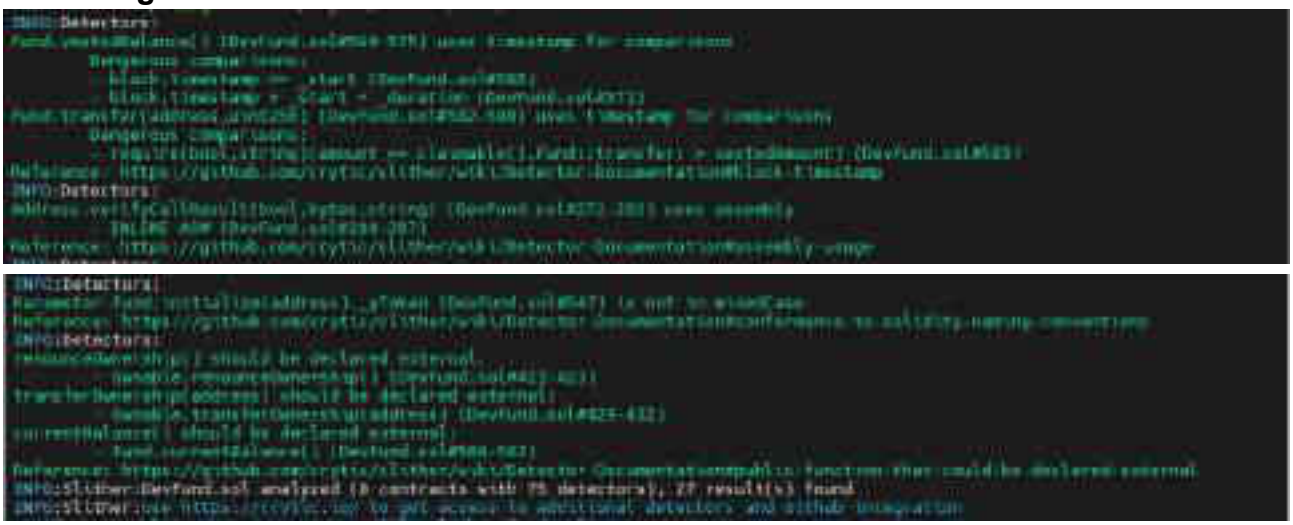
Slither log >> NFTControllerProxy.sol



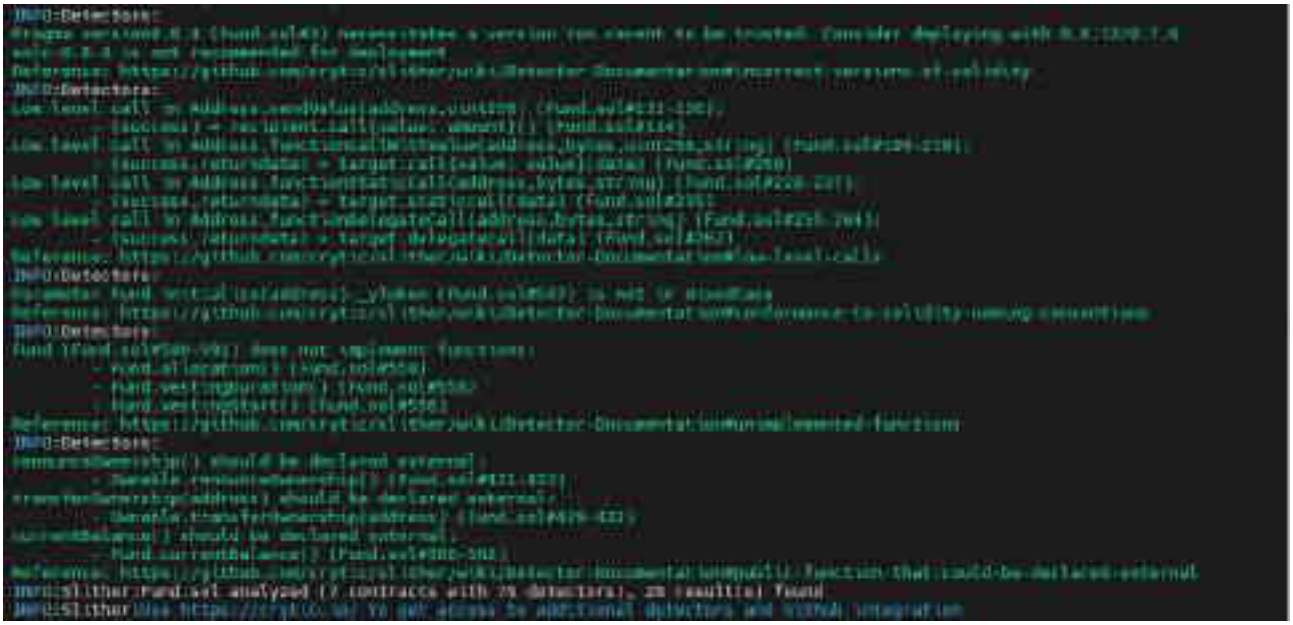
Slither log >> DaoFund.sol



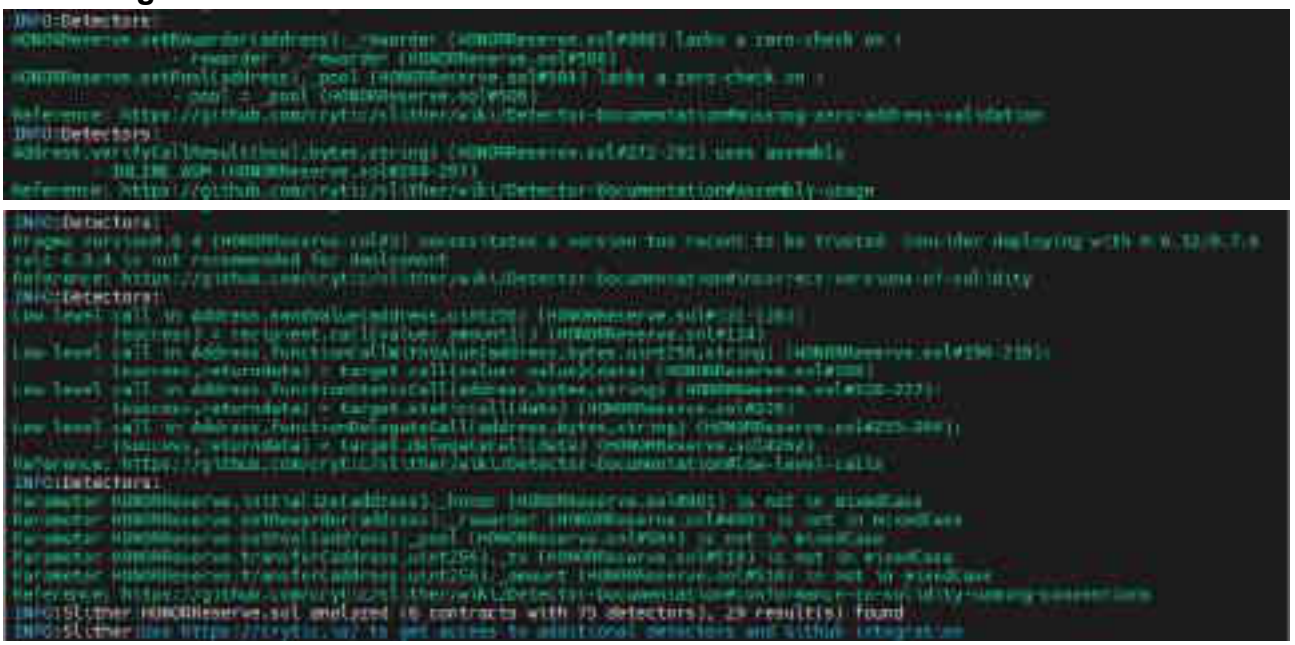
Slither log >> DevFund.sol



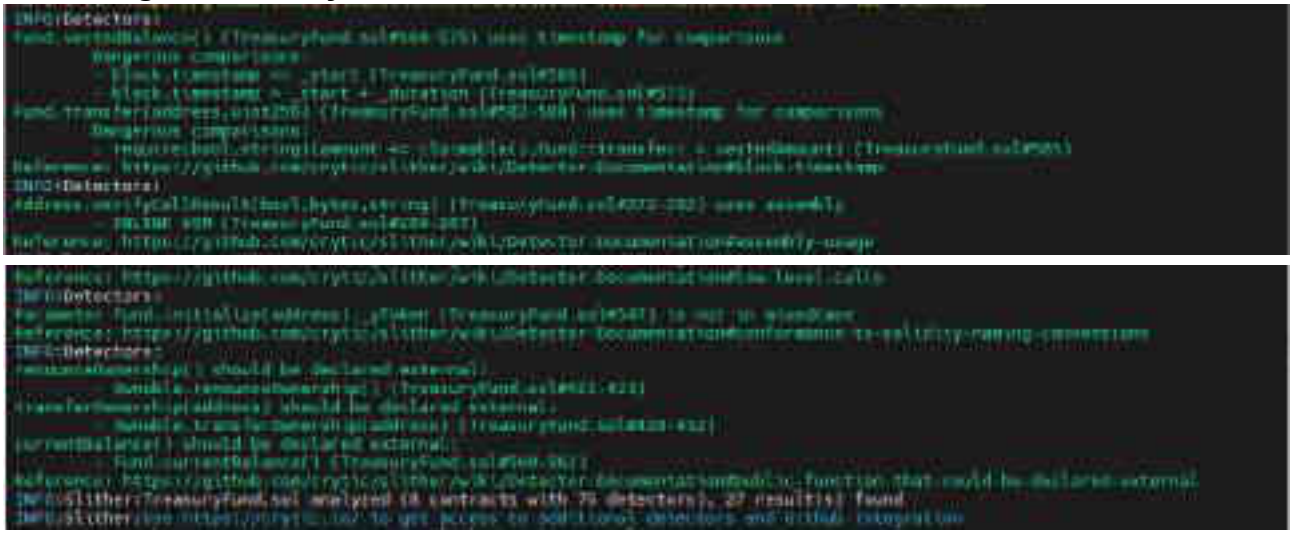
Slither log >> Fund.sol



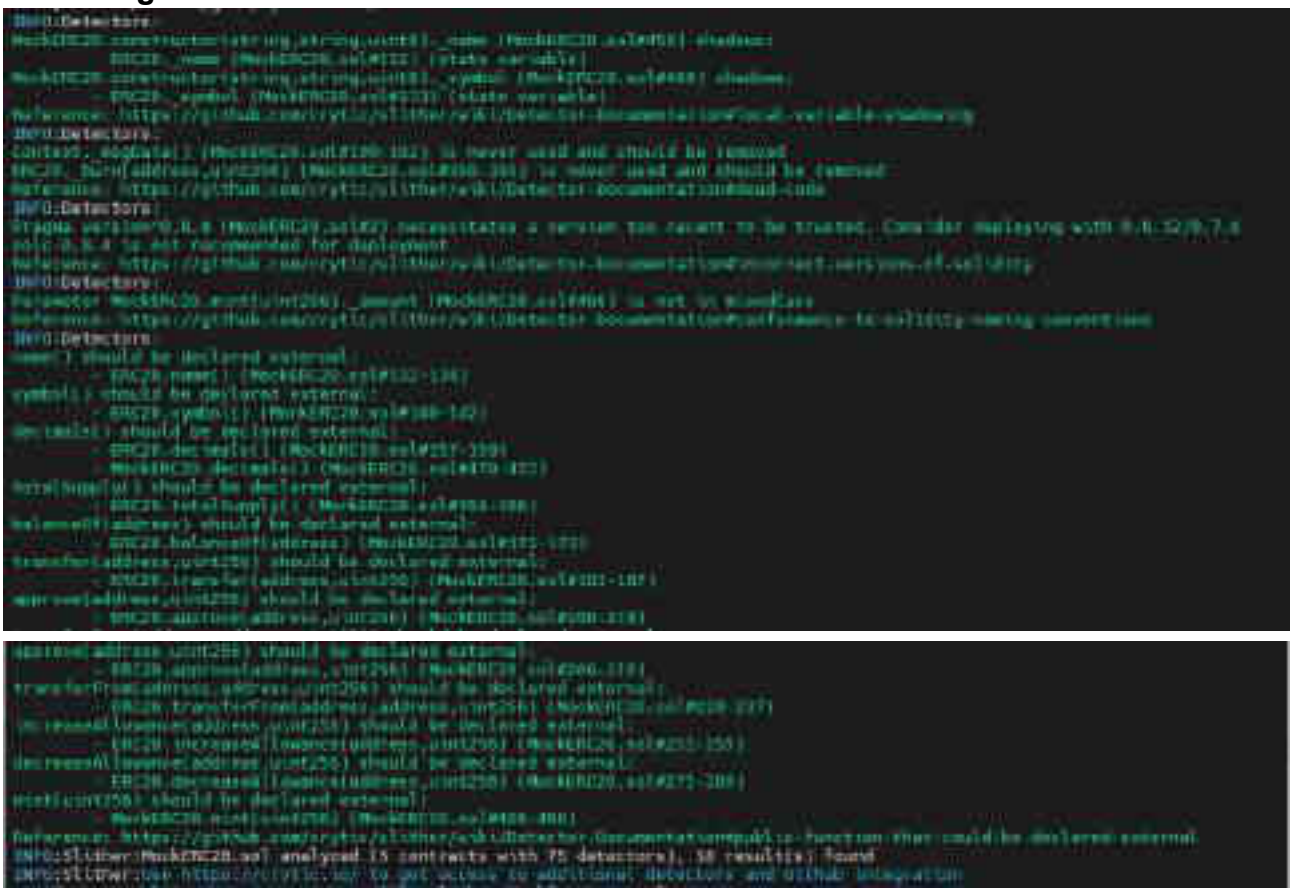
Slither log >> HONORReserve.sol



Slither log >> TreasuryFund.sol

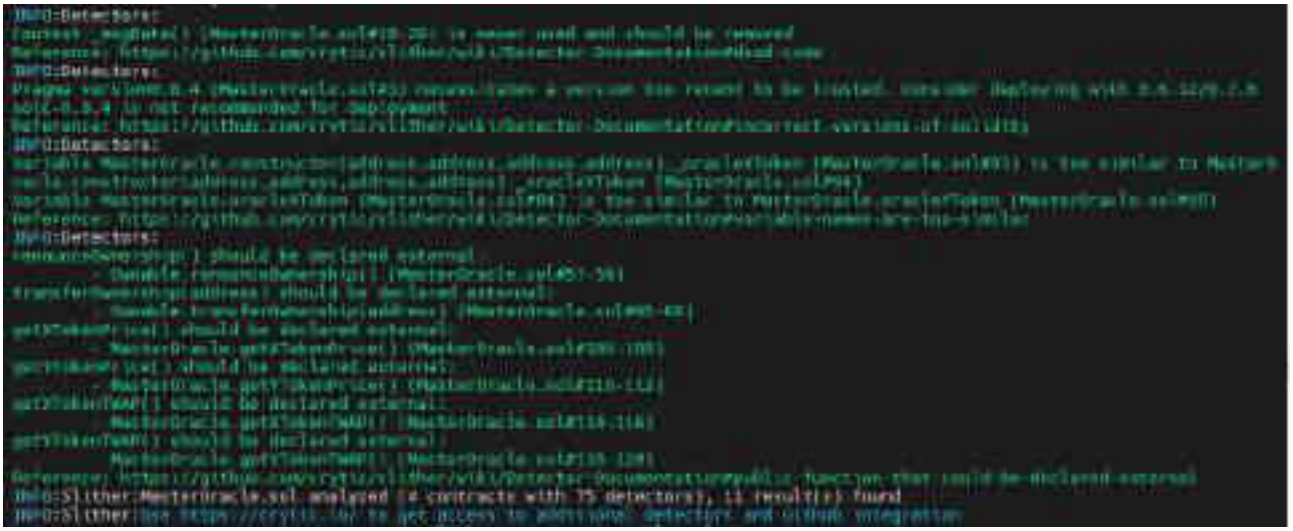


Slither log >> MockERC20.sol

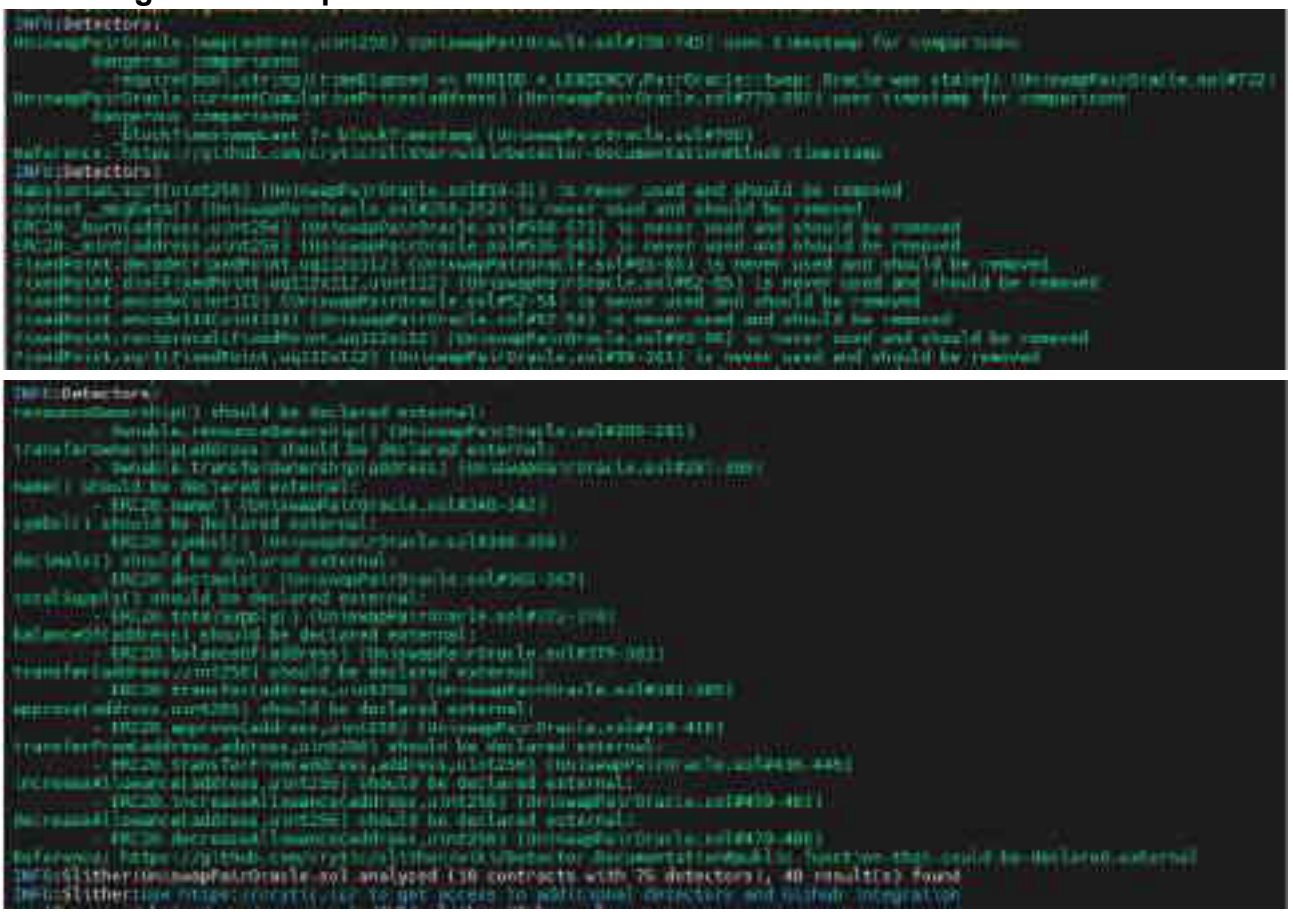


Slither log >> MockTreasury.sol

Slither log >> MasterOracle.sol

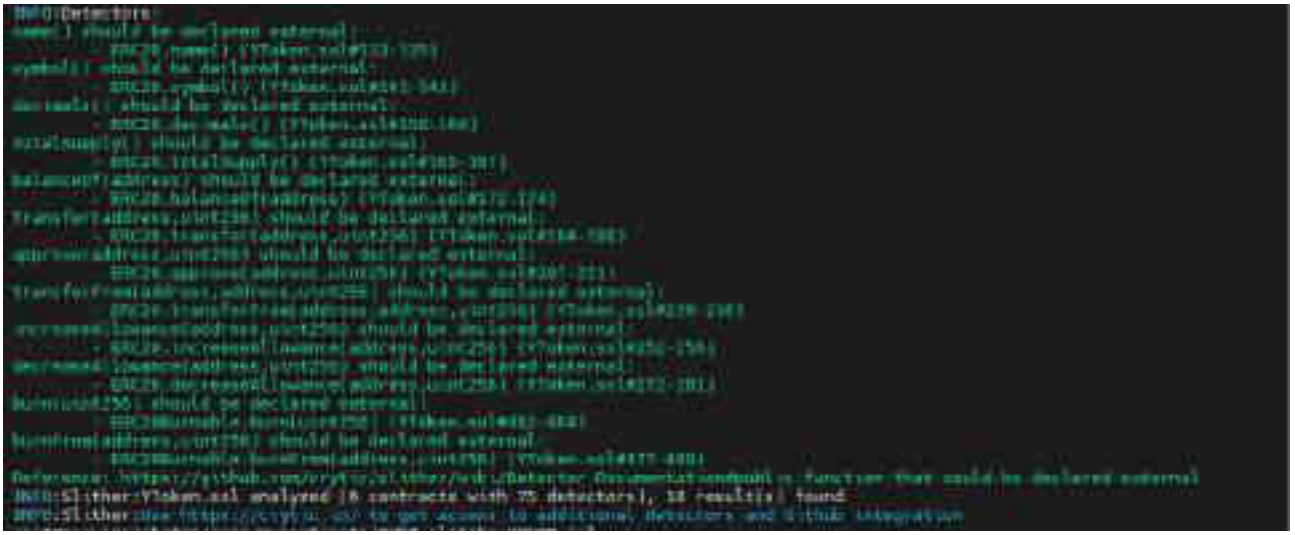


Slither log >> UniswapPairOracle.sol

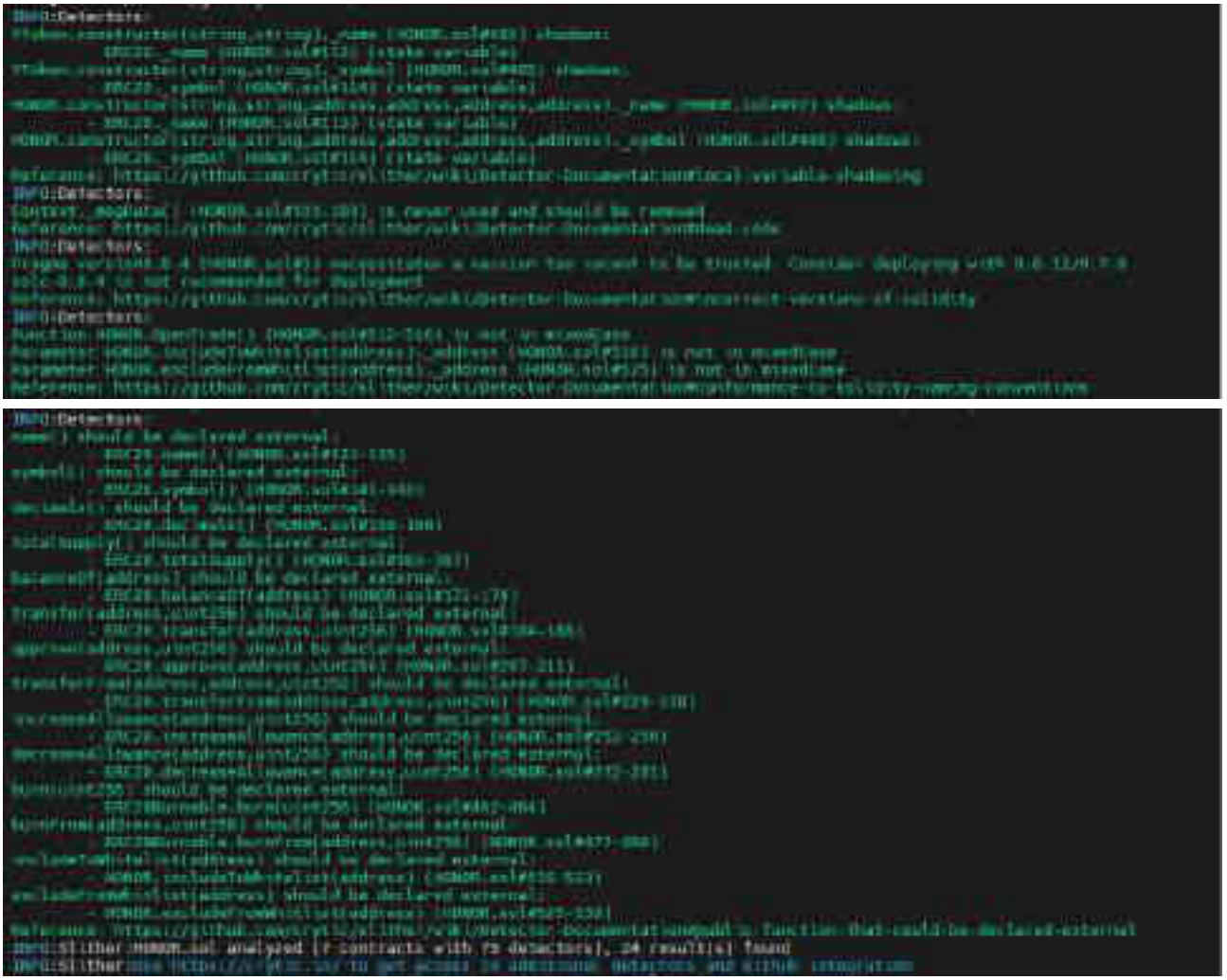


Slither log >> XToken.sol

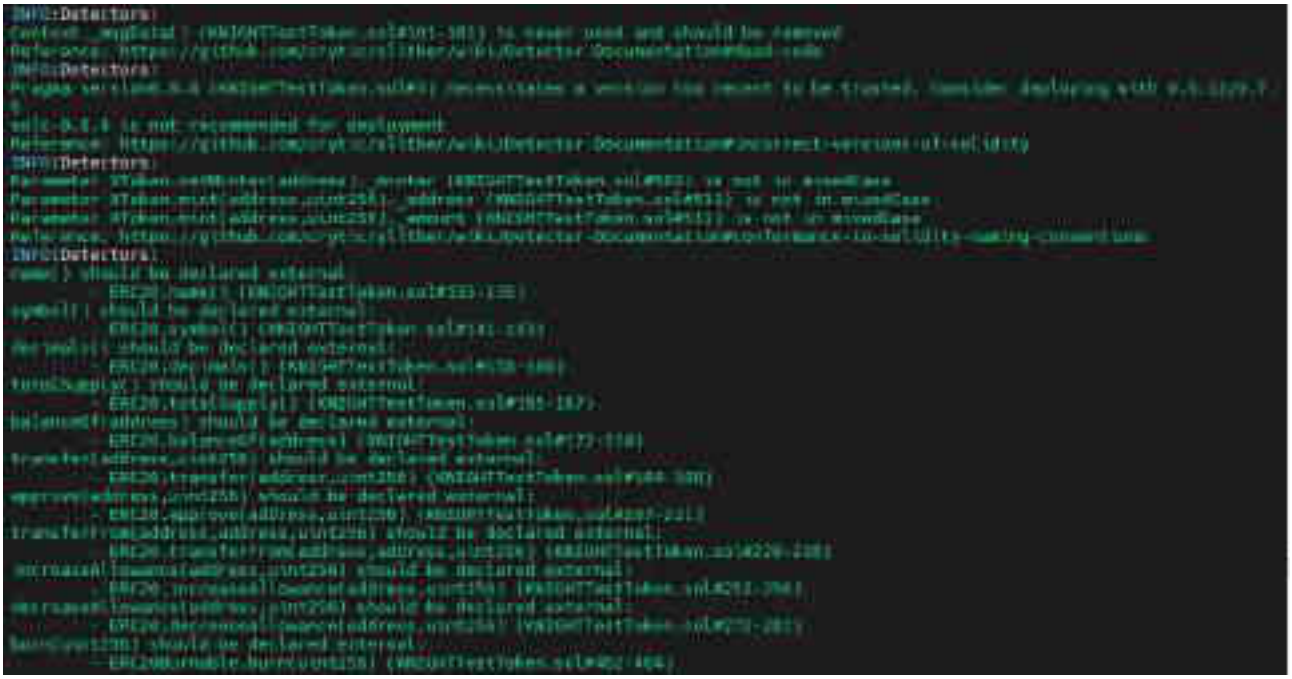
Slither log >> YToken.sol



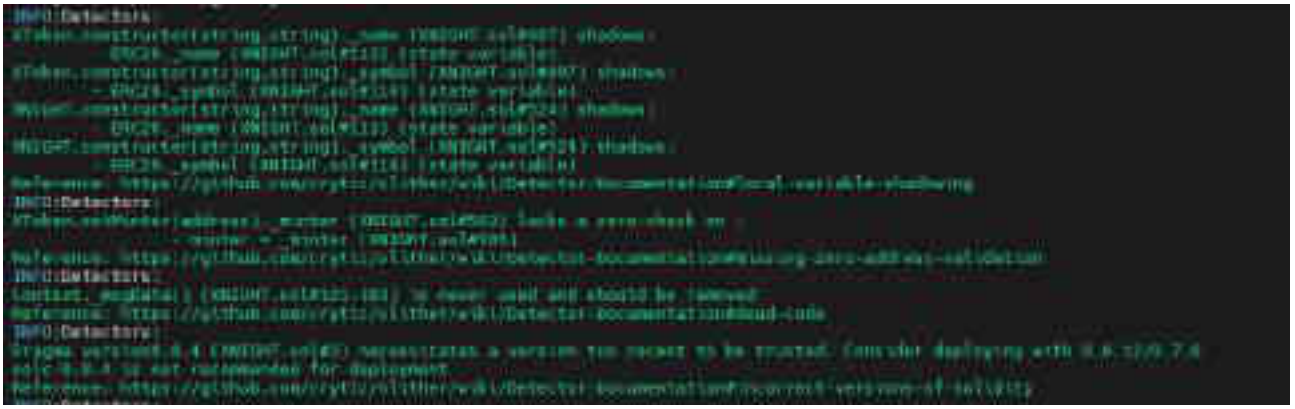
Slither log >> HONOR.sol

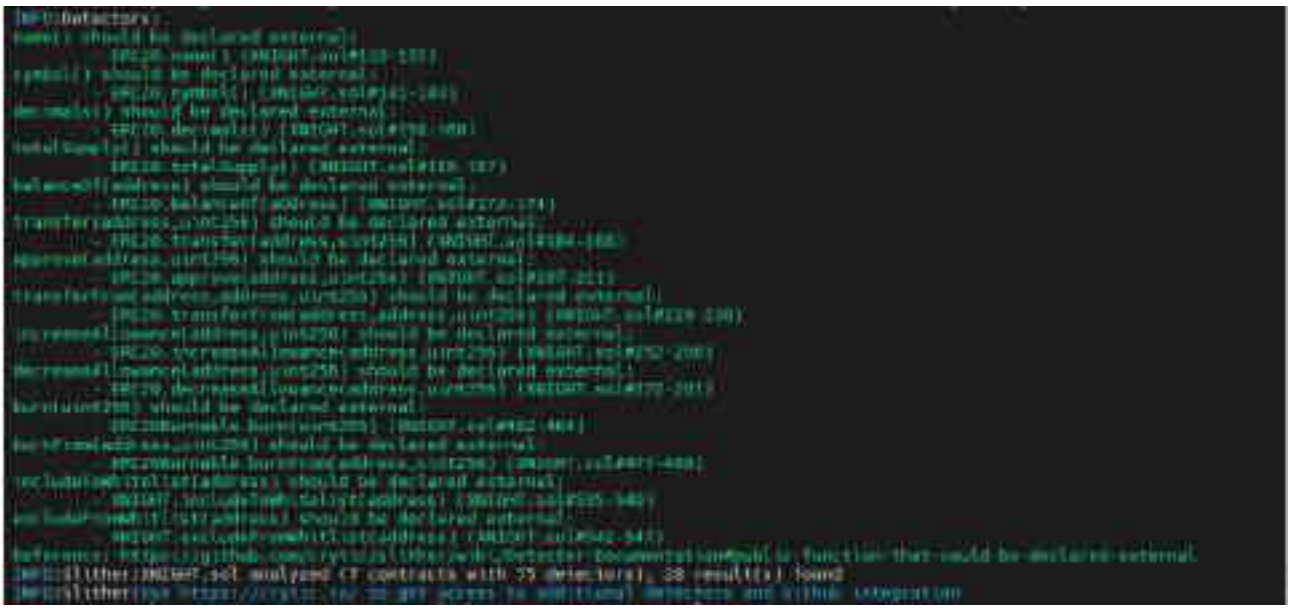


Slither log >> KNIGHTTestToken.sol

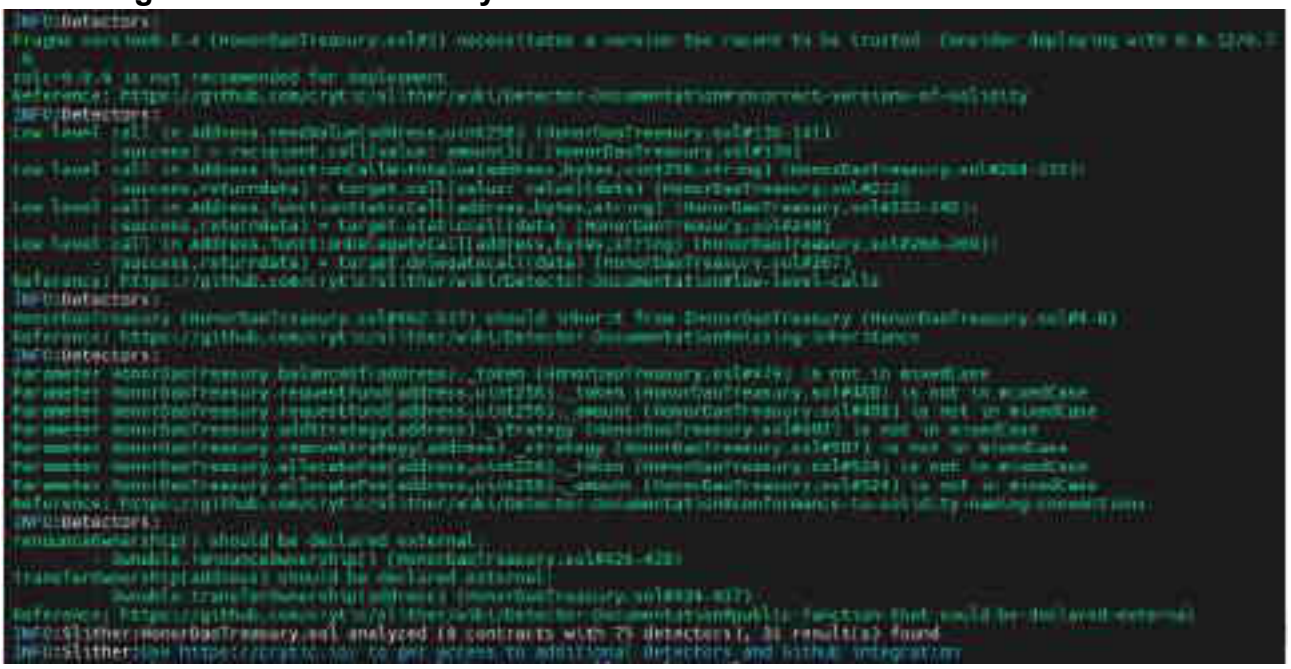


Slither log >> XNIGHT.sol

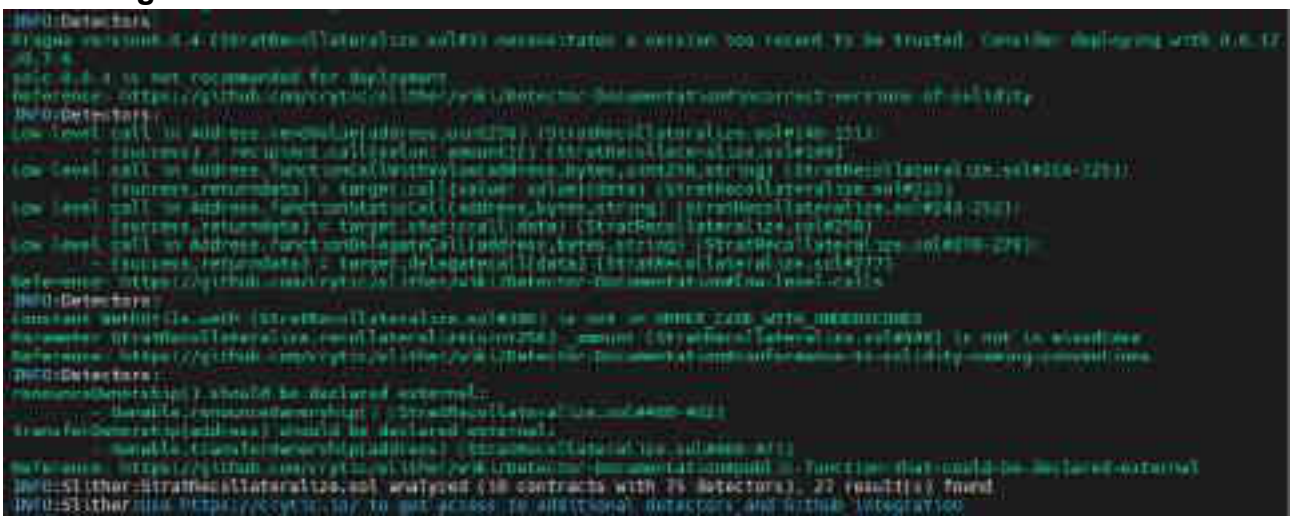




Slither log >> HonorDaoTreasury.sol



Slither log >> StratRecollateralize.sol



Slither log >> StratReduceReserveLP.sol

Solidity Static Analysis

Pool.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Pool.redeem(uint256,uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1283:4:

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.

more

Pos: 1252:33:

Gas & Economy

Gas costs:

Gas requirement of function Pool.collect 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: 1323:4:

Miscellaneous

Constant/View/Pure functions:

Pool.transferToTreasury(uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1490:4:

Similar variable names:

Pool.(address,address,address) : Variables have very similar names "xToken" and "_yToken". Note: Modifiers are currently not considered by this static analysis.

Pos: 1119:25:

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.

more

Pos: 1230:8:

Email: audit@EtherAuthority.io

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: 1202:29:

SwapStrategyPOL.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SwapStrategyPOL.addLiquidity(uint256,uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 763:4:

Gas & Economy

Gas costs:

Gas requirement of function SwapStrategyPOL.yToken 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: 688:4:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

more

Pos: 37:4:

Miscellaneous

Constant/View/Pure functions:

WethUtils.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 612:4:

Similar variable names:

SwapStrategyPOL.addLiquidity(uint256,uint256,uint256) : Variables have very similar names "_amountA" and "_amountB". Note: Modifiers are currently not considered by this static analysis.

Pos: 784:54:

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.

more

Pos: 794:8:

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: 771:34:

Timelock.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Timelock.executeTransaction(address,uint256,string,bytes,uint256): Could potentially lead to re-entrancy vulnerability.

more

Pos: 86:4:

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.

more

Pos: 120:15:

Gas & Economy

Gas costs:

Gas requirement of function Timelock.queueTransaction 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: 54:4:

Miscellaneous

Similar variable names:

Timelock.(address,uint256) : Variables have very similar names "MINIMUM_DELAY" and "MAXIMUM_DELAY".

Pos: 24:26:

Email: audit@EtherAuthority.io

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.

more

Pos: 112:8:

HonorDaoChef.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in HonorDaoChef.withdrawNFT(uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 990:4:

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.

more

Pos: 890:72:

Gas & Economy

Gas costs:

Gas requirement of function HonorDaoChef.pendingReward 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: 681:4:

Miscellaneous

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.

more

Pos: 1010:8:

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: 789:28:

HonorDaoStaking.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in HonorDaoStaking.getReward(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1117:4:

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.
more
Pos: 1055:29:

Gas & Economy

Gas costs:

Gas requirement of function HonorDaoStaking.lockDuration 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: 816:4:

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.
more
Pos: 1138:8:

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.

more

Pos: 1118:8:

Miscellaneous

Constant/View/Pure functions:

HonorDaoStaking.lockedBalances(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 972:4:

Similar variable names:

HonorDaoStaking.claimableRewards(address) : Variables have very similar names "balance" and "balances". Note: Modifiers are currently not considered by this static analysis.

Pos: 930:69:

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.

more

Pos: 1072:8:

HonorDaoZapMMSwap.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in HonorDaoZapMMSwap.swap(address,uint256,address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 793:4:

Gas & Economy

Gas costs:

Gas requirement of function HonorDaoZapMMSwap.zap 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: 738:4:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 39:4:

Miscellaneous

Constant/View/Pure functions:

HonorDaoZapMMSwap.approveToken(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 829:4:

Similar variable names:

HonorDaoZapMMSwap.swap(address,uint256,address) : Variables have very similar names "_token" and "_token0". Note: Modifiers are currently not considered by this static analysis.
Pos: 798:8:

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.
more
Pos: 857:8:

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: 848:15:

NFTController.sol

Security

Low level calls:

Use of "delegatecall" should be avoided whenever possible. External code that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.

more

Pos: 191:50:

Gas & Economy

Gas costs:

Gas requirement of function NFTController.getBoostRate 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: 375:4:

Miscellaneous

Constant/View/Pure functions:

Address.verifyCallResult(bool,bytes,string) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 201:4:

Similar variable names:

NFTController.setBoostRate(address,uint256,uint256) : Variables have very similar names "token" and "tokenId". Note: Modifiers are currently not considered by this static analysis.

Pos: 401:18:

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.

more

Pos: 357:8:

NFTControllerProxy.sol

Security

Low level calls:

Use of "delegatecall" should be avoided whenever possible. External code that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.

more

Pos: 187:50:

Gas & Economy

Gas costs:

Gas requirement of function TransparentUpgradeableProxy.upgradeToAndCall 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: 631:4:

Miscellaneous

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.

more

Pos: 646:8:

DaoFund.sol

Security

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.

more

Pos: 575:31:

Gas & Economy

Gas costs:

Gas requirement of function DaoFund.transfer 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: 583:4:

Miscellaneous

Constant/View/Pure functions:

SafeERC20._callOptionalReturn(contract IERC20,bytes) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 365:4:

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.

more

Pos: 586:8:

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 these yield rational constants.

Pos: 575:15:

DevFund.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SafeERC20.safeDecreaseAllowance(contract IERC20,address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 345:4:

Gas & Economy

Gas costs:

Gas requirement of function DevFund.transfer 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: 582:4:

Miscellaneous

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.

more

Pos: 585:8:

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: 574:15:

Fund.sol

Security

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.

more

Pos: 574:31:

Miscellaneous

Constant/View/Pure functions:

SafeERC20._callOptionalReturn(contract IERC20,bytes) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 364:4:

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.

more

Pos: 585:8:

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: 574:15:

HONORReserve.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SafeERC20.safeDecreaseAllowance(contract IERC20,address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 345:4:

Gas & Economy

Gas costs:

Gas requirement of function HONORReserve.transfer 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: 510:4:

Miscellaneous

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.

more

Pos: 512:8:

MockTreasury.sol

Security

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.

more

Pos: 574:31:

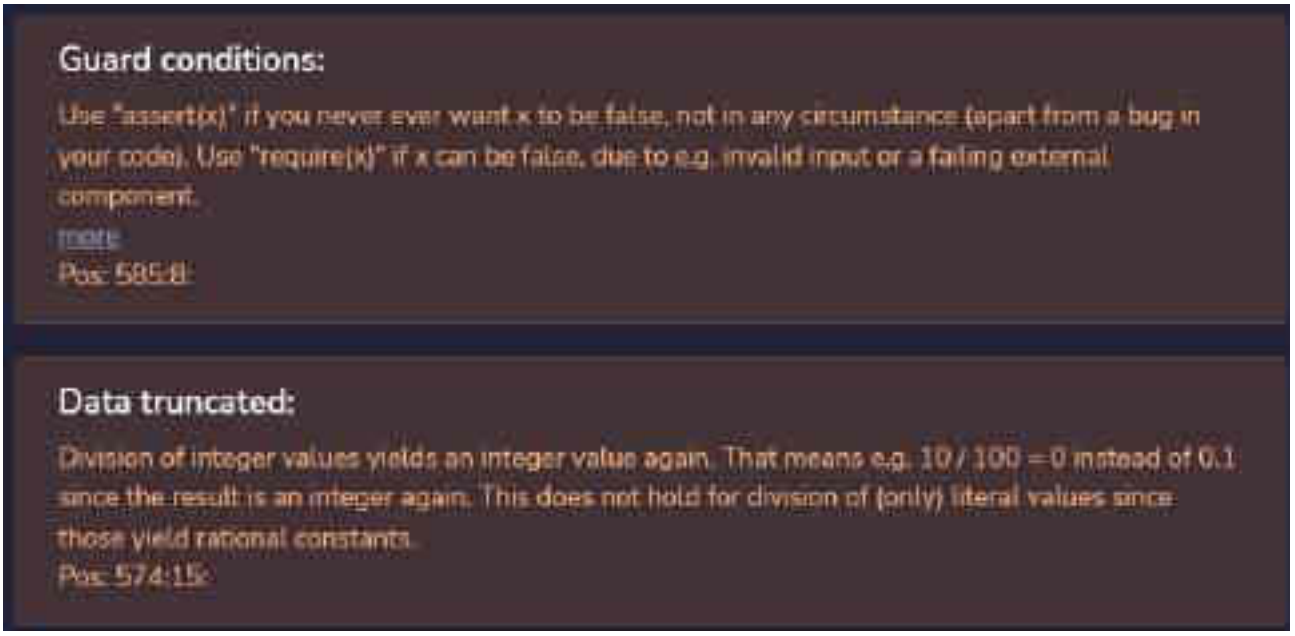
Gas & Economy

Gas costs:

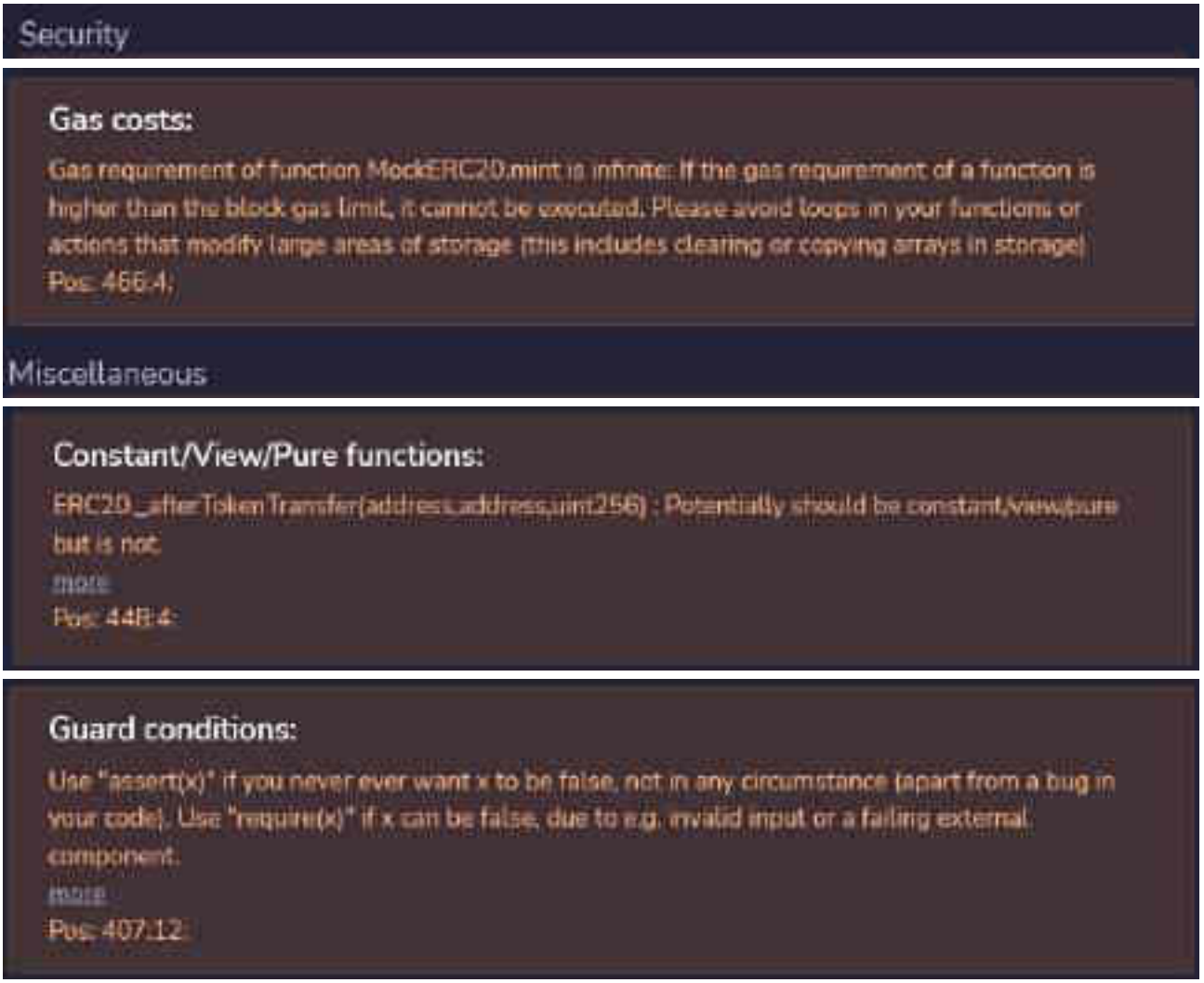
Gas requirement of function TreasuryFund.transfer is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 582:4:

Miscellaneous



MockERC20.sol



TreasuryFund.sol

Security

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.

more

Pos: 574:31:

Gas & Economy

Gas costs:

Gas requirement of function TreasuryFund.currentBalance 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: 560:4:

Miscellaneous

Constant/View/Pure functions:

SafeERC20._callOptionalReturn(contract IERC20,bytes) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 364:4:

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.

more

Pos: 585:8:

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: 574:15:

MasterOracle.sol

Gas & Economy

Gas costs:

Gas requirement of function MasterOracle.getYTokenTWAP 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: 118:4:

Miscellaneous

Similar variable names:

MasterOracle.getYTokenTWAP() : Variables have very similar names "xToken" and "yToken". Note: Modifiers are currently not considered by this static analysis.
Pos: 119:33:

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.
more
Pos: 99:8:

UniswapPairOracle.sol

Security

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.
more
Pos: 766:22:

Gas & Economy

Gas costs:

Gas requirement of function UniswapPairOracle.spot 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: 747:4:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 110:4:

Miscellaneous

Constant/View/Pure functions:

UniswapPairOracle.currentCumulativePrices(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 770:4:

Similar variable names:

UniswapPairOracle.update() : Variables have very similar names "price0CumulativeLast" and "price1CumulativeLast". Note: Modifiers are currently not considered by this static analysis.
Pos: 724:12:

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.
more
Pos: 753:8:

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: 758:21:

XToken.sol

Gas & Economy

Gas costs:

Gas requirement of function ERC20.decreaseAllowance 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: 272:4:

Miscellaneous

Constant/View/Pure functions:

ERC20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
Pos: 449:4:

Similar variable names:

ERC20Burnable.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 479:23

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: 504:8

YToken.sol

Gas & Economy

Gas costs:

Gas requirement of function ERC20.decreaseAllowance 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: 272:4:

Miscellaneous

Constant/View/Pure functions:

ERC20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not.

more

Pos: 449:4:

Similar variable names:

ERC20Burnable.burnFrom(address,uint256) : Variables have very similar names "account" and "amount".

Pos: 479:23:

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.

more

Pos: 408:12:

HONOR.sol

Gas & Economy

Gas costs:

Gas requirement of function HONOR.burnFrom 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: 477:4:

Miscellaneous

Constant/View/Pure functions:

HONOR._transfer(address,address,uint256) : Potentially should be constant/view/pure but is not.

more

Pos: 532:4:

Similar variable names:

HONOR.(string,string,address,address,address,address) : Variables have very similar names "_daoFund" and "_devFund".

Pos: 506:14:

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.

more

Pos: 537:8:

KNIGHTTestToken.sol

XNIGHT.sol

HonorDaoTreasury.sol

Gas & Economy

Gas costs:

Gas requirement of function HonorDaoTreasury.balanceOf 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: 479:4:

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.

more

Pos: 511:8:

Miscellaneous

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.

more

Pos: 526:8:

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.

more

Pos: 509:8:

StratRecollateralize.sol

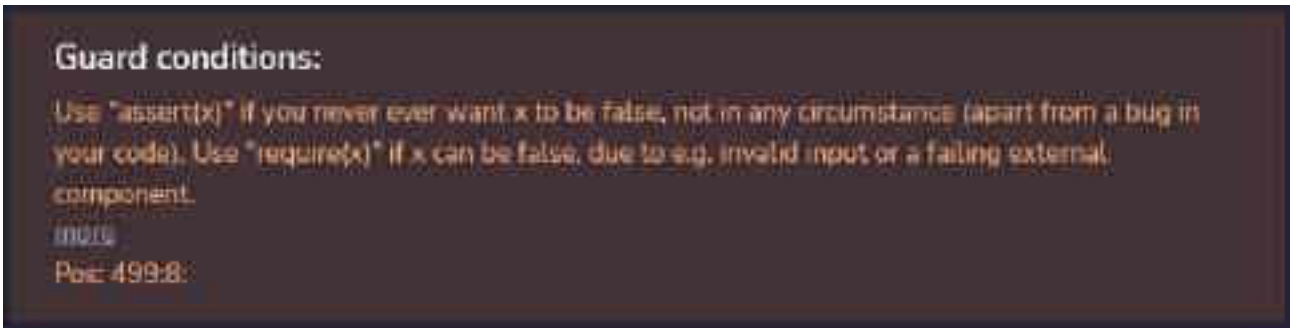
Security

Check-effects-interaction:

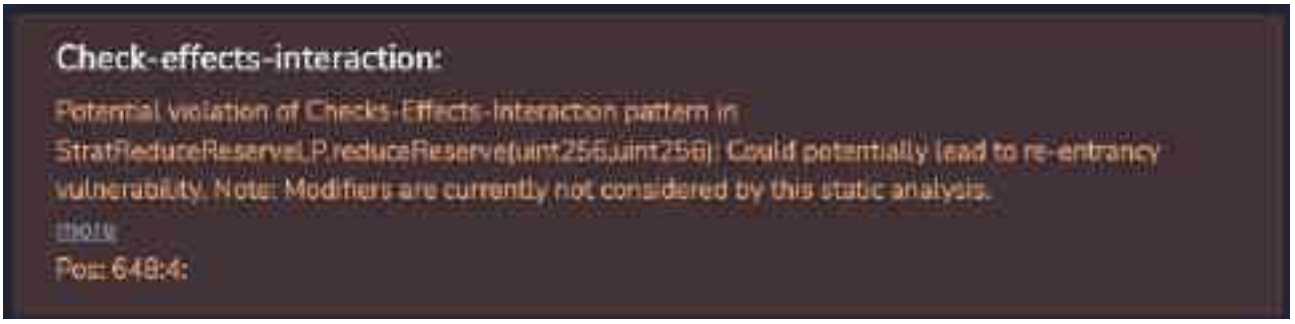
Potential violation of Checks-Effects-Interaction pattern in StratRecollateralize.recollateralize(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

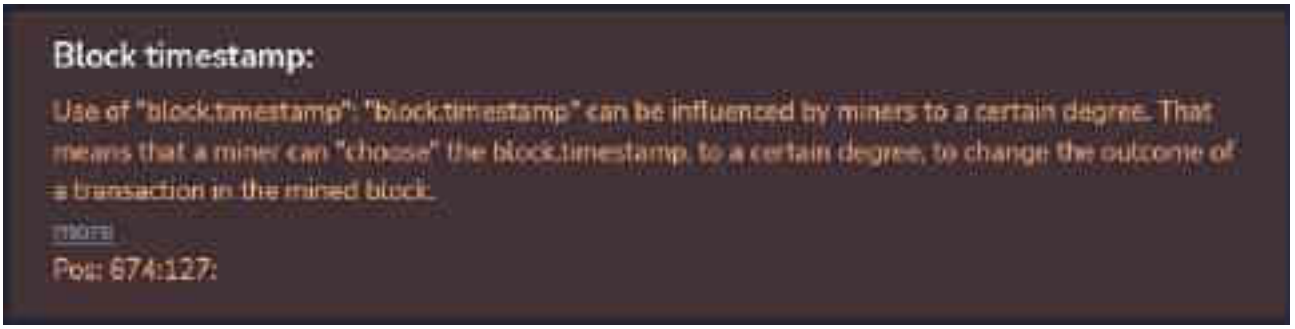
more

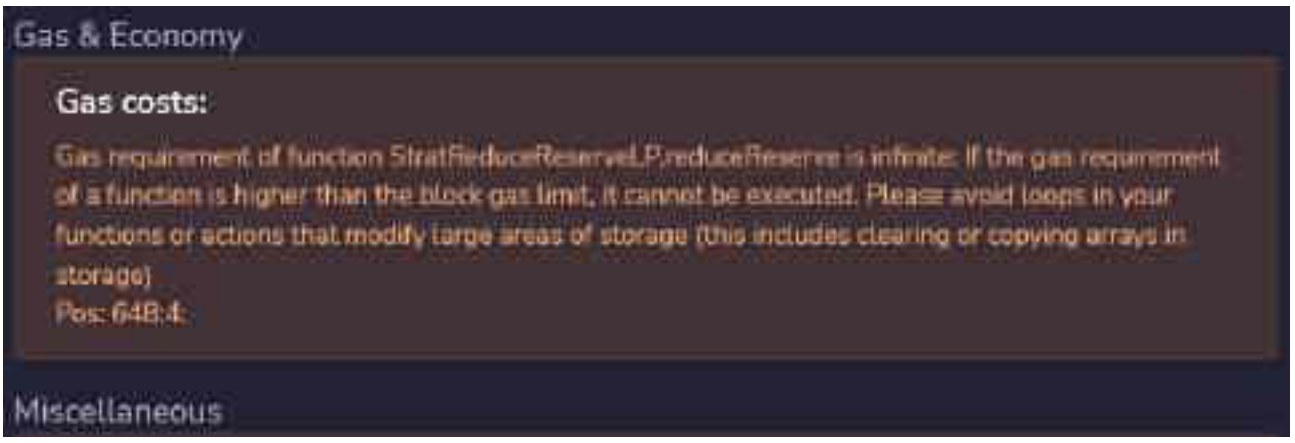
Pos: 498:4:



StratReduceReserveLP.sol







Constant/View/Pure functions:

WethUtils.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 544:4

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.

more

Pos: 649:8

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.

more

Pos: 650:8

Solhint Linter

Pool.sol

```
Pool.sol:521:18: Error: Parse error: missing ';' at '{'
Pool.sol:768:18: Error: Parse error: missing ';' at '{'
Pool.sol:801:18: Error: Parse error: missing ';' at '{'
Pool.sol:850:18: Error: Parse error: missing ';' at '{'
Pool.sol:901:22: Error: Parse error: missing ';' at '{'
```

SwapStrategyPOL.sol

```
SwapStrategyPOL.sol:560:18: Error: Parse error: missing ';' at '{'
```

Timelock.sol

```
Timelock.sol:3:1: Error: Compiler version 0.8.4 does not satisfy the
r semver requirement
Timelock.sol:23:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Timelock.sol:111:51: Error: Avoid using low level calls.
Timelock.sol:120:16: Error: Avoid to make time-based decisions in
your business logic
```

HonorDaoChef.sol

```
HonorDaoChef.sol:526:18: Error: Parse error: missing ';' at '{'
```

HonorDaoStaking.sol

```
HonorDaoStaking.sol:48:18: Error: Parse error: missing ';' at '{'
HonorDaoStaking.sol:61:18: Error: Parse error: missing ';' at '{'
HonorDaoStaking.sol:73:18: Error: Parse error: missing ';' at '{'
HonorDaoStaking.sol:90:18: Error: Parse error: missing ';' at '{'
HonorDaoStaking.sol:102:18: Error: Parse error: missing ';' at '{'
HonorDaoStaking.sol:198:18: Error: Parse error: missing ';' at '{'
HonorDaoStaking.sol:221:18: Error: Parse error: missing ';' at '{'
HonorDaoStaking.sol:247:18: Error: Parse error: missing ';' at '{'
HonorDaoStaking.sol:617:18: Error: Parse error: missing ';' at '{'
```

HonorDaoZapMMSwap.sol

```
HonorDaoZapMMSwap.sol:563:18: Error: Parse error: missing ';' at '{'
```

Fund.sol

```
Fund.sol:350:18: Error: Parse error: missing ';' at '{'
```

NFTController.sol

```
NFTController.sol:4:1: Error: Compiler version 0.8.4 does not satisfy
the r semver requirement
NFTController.sol:63:28: Error: Avoid using low level calls.
NFTController.sol:137:51: Error: Avoid using low level calls.
NFTController.sol:191:51: Error: Avoid using low level calls.
NFTController.sol:213:17: Error: Avoid using inline assembly. It is
acceptable only in rare cases
NFTController.sol:334:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
NFTController.sol:334:20: Error: Code contains empty blocks
NFTController.sol:368:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
NFTController.sol:368:19: Error: Code contains empty blocks
```

NFTControllerProxy.sol

```
NFTControllerProxy.sol:3:1: Error: Compiler version 0.8.4 does not
satisfy the r semver requirement
NFTControllerProxy.sol:59:28: Error: Avoid using low level calls.
NFTControllerProxy.sol:133:51: Error: Avoid using low level calls.
NFTControllerProxy.sol:187:51: Error: Avoid using low level calls.
NFTControllerProxy.sol:209:17: Error: Avoid using inline assembly. It
is acceptable only in rare cases
NFTControllerProxy.sol:261:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
NFTControllerProxy.sol:271:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
NFTControllerProxy.sol:281:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
NFTControllerProxy.sol:291:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
NFTControllerProxy.sol:303:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
NFTControllerProxy.sol:365:49: Error: Code contains empty blocks
NFTControllerProxy.sol:541:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
```

```
NFTControllerProxy.sol:558:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
NFTControllerProxy.sol:654:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
NFTControllerProxy.sol:654:114: Error: Code contains empty blocks
```

DaoFund.sol

```
DaoFund.sol:351:18: Error: Parse error: missing ';' at '{'
```

DevFund.sol

```
DevFund.sol:350:18: Error: Parse error: missing ';' at '{'
```

HONORReserve.sol

```
HONORReserve.sol:350:18: Error: Parse error: missing ';' at '{'
```

MockTreasury.sol

```
MockTreasury.sol:350:18: Error: Parse error: missing ';' at '{'
```

MockERC20.sol

```
MockERC20.sol:275:18: Error: Parse error: missing ';' at '{'
MockERC20.sol:308:18: Error: Parse error: missing ';' at '{'
MockERC20.sol:357:18: Error: Parse error: missing ';' at '{'
MockERC20.sol:408:22: Error: Parse error: missing ';' at '{'
```

TreasuryFund.sol

```
TreasuryFund.sol:350:18: Error: Parse error: missing ';' at '{'
```

MasterOracle.sol

```
MasterOracle.sol:3:1: Error: Compiler version 0.8.4 does not satisfy
the r semver requirement
MasterOracle.sol:31:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
MasterOracle.sol:90:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
```

UniswapPairOracle.sol

```
UniswapPairOracle.sol:483:18: Error: Parse error: missing ';' at '{'
UniswapPairOracle.sol:516:18: Error: Parse error: missing ';' at '{'
UniswapPairOracle.sol:565:18: Error: Parse error: missing ';' at '{'
UniswapPairOracle.sol:616:22: Error: Parse error: missing ';' at '{'
UniswapPairOracle.sol:708:18: Error: Parse error: missing ';' at '{'
UniswapPairOracle.sol:787:18: Error: Parse error: missing ';' at '{'
```

XToken.sol

```
XToken.sol:276:18: Error: Parse error: missing ';' at '{'
XToken.sol:309:18: Error: Parse error: missing ';' at '{'
XToken.sol:358:18: Error: Parse error: missing ';' at '{'
XToken.sol:409:22: Error: Parse error: missing ';' at '{'
```

YToken.sol

```
YToken.sol:276:18: Error: Parse error: missing ';' at '{'
YToken.sol:309:18: Error: Parse error: missing ';' at '{'
YToken.sol:358:18: Error: Parse error: missing ';' at '{'
YToken.sol:409:22: Error: Parse error: missing ';' at '{'
```

HONOR.sol

```
HONOR.sol:276:18: Error: Parse error: missing ';' at '{'
HONOR.sol:309:18: Error: Parse error: missing ';' at '{'
HONOR.sol:358:18: Error: Parse error: missing ';' at '{'
HONOR.sol:409:22: Error: Parse error: missing ';' at '{'
```

KNIGHTTestToken.sol

```
KNIGHTTestToken.sol:276:18: Error: Parse error: missing ';' at '{'
KNIGHTTestToken.sol:309:18: Error: Parse error: missing ';' at '{'
KNIGHTTestToken.sol:358:18: Error: Parse error: missing ';' at '{'
KNIGHTTestToken.sol:409:22: Error: Parse error: missing ';' at '{'
```

XNIGHT.sol

```
XNIGHT.sol:276:18: Error: Parse error: missing ';' at '{'
XNIGHT.sol:309:18: Error: Parse error: missing ';' at '{'
XNIGHT.sol:358:18: Error: Parse error: missing ';' at '{'
XNIGHT.sol:409:22: Error: Parse error: missing ';' at '{'
```

HonorDaoTreasury.sol

```
HonorDaoTreasury.sol:355:18: Error: Parse error: missing ';' at '{'
```

StratRecollateralize.sol

```
StratRecollateralize.sol:365:18: Error: Parse error: missing ';' at
'{'
```

StratReduceReserveLP.sol

```
StratReduceReserveLP.sol:491:18: Error: Parse error: missing ';' at
'{'
```

Software analysis result:

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

Email: audit@EtherAuthority.io