Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Decentralized Foundation
Website: https://defo.app
Platform: Avalanche Network
Language: Solidity
Date: May 12th, 2022

Table of contents

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.

Introduction

EtherAuthority was contracted by the Defo Contracts team to perform the Security audit of the Defo Contracts 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 May 12th, 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

- Decentralized Foundation (DEFO) is a DeFi-As-A-Service protocol on the AVAX network that aims to leverage the world of DeFi to generate funds while also helping the most in need through the charity donations.
- The Defo Contracts have functions like burn, mint, initialize, fallback, receive, isPair, RedeemMint, MintGem, BoostGem, RedeemMint, transferLog, Compound, unstakeTokens, etc.
- The Defo Contracts inherit the Ownable, ERC20, ERC20Burnable, AccessControlUpgradeable, OwnableUpgradeable, Initializable, IERC20, SafeERC20, IERC721Enumerable, SafeMath, Address, Context, Strings, console 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 Defo Contracts Protocol Smart Contracts
Platform	**Avalanche / Solidity**
File 1	DefoErc20.sol
File 1 MD5 Hash	B8FEE41D9A557EFC48B3AEC65503F64A
File 2	DefoLimiter.sol
File 2 MD5 Hash	0ABD9395AB9D4294244AFB56652571FC
File 3	Diamond.sol
File 3 MD5 Hash	C5C1B4ECBF45A50FEF6F03467D06CC0D
File 4	DiamondInit.sol
File 4 MD5 Hash	A526ABD7D1B125AD8A768B1C0BDB9F43
File 5	LpManager.sol
File 5 MD5 Hash	32336CBEFBCC63799C439768750364EB
File 6	Redeem.sol
File 6 MD5 Hash	652B270D7F0A3833F190AEAD9E5DE5CE
File 7	DiamondCutFacet.sol
File 7 MD5 Hash	FB675C8189FD9D6CD4CAF93BB200A98F
File 8	DiamondLoupeFacet.sol
File 8 MD5 Hash	84553C4947D9917B1C73F83BFCF62B68
File 9	ERC721Enumerable.sol
File 9 MD5 Hash	30EA4861F52D9AD443509CB25D101CAD
File 10	ERC721Facet.sol
File 10 MD5 Hash	F7636C5D8C29AE0C59BD5D7EF9DE4847
File 11	GemFacet.sol
File 11 MD5 Hash	198FD8793B0BCC22198CF2223828B723

File 12	GemGettersFacet.sol
File 12 MD5 Hash	6AE044E55A2E950003F4C8C2ABD1122E
File 13	NodeLimiterFacet.sol
File 13 MD5 Hash	608D61E7914E26EB76546F82EC9FC450
File 14	OwnerFacet.sol
File 14 MD5 Hash	EA1DC6EABD981C327EF21C348B17D616
File 15	OwnershipFacet.sol
File 15 MD5 Hash	D392728362597E7666A0FCFF960A8CB9
File 16	VaultStakingFacet.sol
File 16 MD5 Hash	5A29BA843AB1E10EE747D70FF6E3F658
Audit Date	May 12th,2022

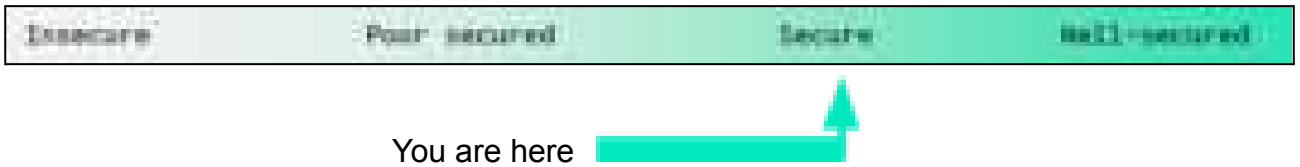
Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 DefoErc20.sol • Name: Defo Token • Symbol: DEFO • Decimal: 18 • Total Supply: 0.2 Million	**YES, This is valid.**
File 2 DefoLimiter.sol • Total Supply a wallet can sell: 10 • Time Frame Expiration: 1 days	**YES, This is valid.**
File 3 Diamond.sol • Find facet for function that is called and execute the function if a facet is found	**YES, This is valid.**
File 4 DiamondInit.sol • It initializes the diamond	**YES, This is valid.**
File 5 LpManager.sol • It manages LP Tokens which includes, adding liquidity, create pair, buffer system, etc.	**YES, This is valid.**
File 6 Redeem.sol • Redeem contract allows to redeem from presale and second presale.l • It has functions like: setNodeAddress, flipActive, etc.	**YES, This is valid.**
File 7 DiamondCutFacet.sol • DiamondCutFacet has functions like: diamondCut.	**YES, This is valid.**
File 8 DiamondLoupeFacet.sol • DiamondLoupeFacet has functions like:	**YES, This is valid.**

facets, facetFunctionSelectors, facetAddress, etc.	
File 9 ERC721Enumerable.sol • ERC721Enumerable has functions like: tokenOfOwnerByIndex, tokenByIndex, etc.	**YES, This is valid.**
File 10 ERC721Facet.sol • ERC721Facet has functions like: balanceOf, tokenURI, ownerOf, getApproved, etc.	**YES, This is valid.**
File 11 GemFacet.sol • GemFacet has functions like: _distributePayment, _sendRewardTokens.	**YES, This is valid.**
File 12 GemGettersFacet.sol • GemGettersFacet has functions like: GemOf, GetGemTypeMetadata, etc.	**YES, This is valid.**
File 13 NodeLimiterFacet.sol • NodeLimiterFacet has functions like: transferLimit, addToWhitelist, etc.	**YES, This is valid.**
File 14 OwnerFacet.sol • OwnerFacet has functions like: setTaperRate, ChangePaymentToken, etc.	**YES, This is valid.**
File 15 OwnershipFacet.sol • OwnershipFacet has functions like: transferOwnership, etc.	**YES, This is valid.**
File 16 VaultStakingFacet.sol • VaultStakingFacet has functions like: showStakedAmount, unstakeTokens,etc.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. 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, 2 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	Moderate
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderate
	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	Moderate
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	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 16 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 Defo Contracts 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 Defo Contracts Protocol.

The Defo Contracts Protocol team has provided unit test scripts, which have helped to determine the integrity of the code in an automated way.

Code parts are **not** well commented on smart contracts.

Documentation

We were given a Defo Contracts Protocol smart contract code in the form of a Github weblink. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. So it is not 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://defo.app which provided rich information about the project architecture and tokenomics.

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

DefoErc20.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	burn	write	Passed	No Issue
21	burnFrom	write	Passed	No Issue
22	owner	read	Passed	No Issue
23	onlyOwner	modifier	Passed	No Issue
24	renounceOwnership	write	access only Owner	No Issue
25	transferOwnership	write	access only Owner	No Issue
26	_transferOwnership	internal	Passed	No Issue
27	onlyLiquidityPoolManager	modifier	Passed	No Issue
28	getLiquidityPoolManager	read	Passed	No Issue
29	setLiquidityPoolManager	write	access only Owner	No Issue
30	recoverLostTokens	external	access only Owner	No Issue

DefoLimiter.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	__Ownable_init	internal	access only Initializing	No Issue
3	__Ownable_init_unchained	internal	access only Initializing	No Issue

4	owner	read	Passed	No Issue
5	onlyOwner	modifier	Passed	No Issue
6	renounceOwnership	write	access only Owner	No Issue
7	transferOwnership	write	access only Owner	No Issue
8	_transferOwnership	internal	Passed	No Issue
9	__AccessControl_init	internal	access only Initializing	No Issue
10	__AccessControl_init_unchained	internal	Passed	No Issue
11	onlyRole	modifier	Passed	No Issue
12	supportsInterface	read	Passed	No Issue
13	hasRole	read	Passed	No Issue
14	_checkRole	internal	Passed	No Issue
15	getRoleAdmin	read	Passed	No Issue
16	grantRole	write	access only Role	No Issue
17	revokeRole	write	access only Role	No Issue
18	renounceRole	write	Passed	No Issue
19	_setupRole	internal	Passed	No Issue
20	_setRoleAdmin	internal	Passed	No Issue
21	_grantRole	internal	Passed	No Issue
22	_revokeRole	internal	Passed	No Issue
23	initialize	write	Passed	No Issue
24	onlyDefoToken	modifier	Passed	No Issue
25	checkTimeframe	modifier	Passed	No Issue
26	notDenied	modifier	Passed	No Issue
27	isPair	read	Passed	No Issue
28	transferLog	external	access only Defo Token	No Issue
29	setMaxPercentage	external	Owner can stop the sell	Refer audit findings
30	setTokenAddress	external	access only Role	No Issue
31	setLPAddress	external	access only Role	No Issue
32	setLPManager	external	access only Role	No Issue
33	setDiamond	write	access only Role	No Issue
34	setTimeframeExpiration	external	access only Role	No Issue
35	editWhitelist	external	access only Role	No Issue
36	editBlocklist	external	access only Role	No Issue

Diamond.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	fallback	external	Passed	No Issue
3	receive	external	Passed	No Issue

DiamondInit.sol

Functions

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

LpManager.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	recoverLostTokens	external	access only Owner	No Issue
8	validAddress	modifier	Passed	No Issue
9	buffer	external	access only Owner	No Issue
10	addLiquidityToken	write	Passed	No Issue
11	bufferDefo	internal	Passed	No Issue
12	createPairWith	write	Passed	No Issue
13	setAllowance	write	access only Owner	No Issue
14	shouldLiquify	write	access only Owner	No Issue
15	setBufferThreshHold	write	access only Owner	No Issue
16	getRouter	external	Passed	No Issue
17	getPair	external	Passed	No Issue
18	getLeftSide	external	Passed	No Issue
19	getRightSide	external	Passed	No Issue
20	isPair	read	Passed	No Issue
21	getLeftBalance	read	Passed	No Issue
22	getRightBalance	read	Passed	No Issue
23	isLiquidityAdded	external	Passed	No Issue
24	isRouter	read	Passed	No Issue
25	getSupply	external	Passed	No Issue
26	setPairAllowance	write	Passed	No Issue
27	getReserver0	external	Passed	No Issue
28	getReserver1	external	Passed	No Issue
29	checkBalance	external	Passed	No Issue
30	token0	external	Passed	No Issue
31	token1	external	Passed	No Issue

Redeem.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	transferOwnership	write	access only Owner	No Issue
5	_transferOwnership	internal	Passed	No Issue
6	renounceOwnership	write	access only Owner	No Issue
7	isActive	modifier	Passed	No Issue
8	nonReentrant	modifier	Passed	No Issue
9	presaleCompliance	modifier	Passed	No Issue
10	secondPresaleCompliance	modifier	Passed	No Issue
11	timeCompliance	modifier	Passed	No Issue
12	setNodeAddress	write	access only Owner	No Issue
13	startTimer	write	access only Owner	No Issue
14	flipActive	write	access only Owner	No Issue
15	redeem	write	For loops can be merged	Refer audit findings
16	secondPresaleRedeem	write	access is Active	No Issue

DiamondCutFacet.sol

Functions

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

DiamondLoupeFacet.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	facets	external	Passed	No Issue
3	facetFunctionSelectors	external	Passed	No Issue
4	facetAddresses	external	Passed	No Issue
5	facetAddress	external	Passed	No Issue
6	supportsInterface	external	Passed	No Issue

ERC721Enumerable.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializeERC721Enumerable	write	Passed	No Issue
3	tokenOfOwnerByIndex	read	Passed	No Issue
4	totalSupply	read	Passed	No Issue
5	tokenByIndex	read	Passed	No Issue

ERC721Facet.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	write	Passed	No Issue
3	balanceOf	read	Passed	No Issue
4	name	read	Passed	No Issue
5	symbol	read	Passed	No Issue
6	tokenURI	read	Passed	No Issue
7	_baseURI	internal	Passed	No Issue
8	ownerOf	read	Passed	No Issue
9	getApproved	read	Passed	No Issue
10	isApprovedForAll	read	Passed	No Issue
11	approve	write	Passed	No Issue
12	setApprovalForAll	write	Passed	No Issue
13	transferFrom	write	Passed	No Issue
14	safeTransferFrom	write	Passed	No Issue
15	safeTransferFrom	write	Passed	No Issue

GemFacet.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	SaleLock	modifier	Passed	No Issue
3	onlyGemOwner	modifier	Passed	No Issue
4	onlyActive	modifier	Passed	No Issue
5	mintTimeLimit	modifier	Passed	No Issue
6	_mintGem	internal	Passed	No Issue
7	_compound	internal	Passed	No Issue
8	_maintenance	internal	Passed	No Issue
9	_maintenanceDiscount	internal	Passed	No Issue
10	RedeemMint	write	Function input parameters lack of check	Refer audit findings
11	RedeemMintBooster	write	Function input parameters lack of check	Refer audit findings
12	BoostGem	write	access only Gem Owner	No Issue
13	MintGem	external	Passed	No Issue
14	Maintenance	external	access only Gem Owner	No Issue
15	BatchMaintenance	external	Passed	No Issue
16	Compound	external	access only Gem Owner	No Issue

17	isActive	read	Passed	No Issue
18	checkRawReward	read	Passed	No Issue
19	checkTaperedReward	read	Passed	No Issue
20	checkTaxedReward	read	Passed	No Issue
21	checkPendingMaintenance	read	Passed	No Issue
22	getGemIdsOf	read	Passed	No Issue
23	getGemIdsOfWithType	read	Passed	No Issue
24	_distributePayment	internal	Passed	No Issue
25	_sendRewardTokens	internal	Passed	No Issue
26	ClaimRewards	external	access only Gem Owner	No Issue
27	BatchClaimRewards	external	Passed	No Issue

GemGettersFacet.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	GemOf	external	Passed	No Issue
3	GetGemTypeMetadata	external	Passed	No Issue
4	getTotalCharity	external	Passed	No Issue
5	getMeta	external	Passed	No Issue

NodeLimiterFacet.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOwner	write	Passed	No Issue
3	addToWhitelist	external	access only Owner	No Issue
4	transferLimit	write	Passed	No Issue

OwnerFacet.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOwner	modifier	Passed	No Issue
3	initialize	external	access only Owner	No Issue
4	setGemSettings	external	access only Owner	No Issue
5	setAddressAndDistTreasury	external	access only Owner	No Issue
6	setAddressAndDistRewardPool	external	access only Owner	No Issue
7	setAddressDonation	external	access only Owner	No Issue

8	setAddressAndDistTeam	external	access only Owner	No Issue
9	setAddressAndDistLiquidity	external	access only Owner	No Issue
10	setAddressVault	external	access only Owner	No Issue
11	setBaseURI	external	access only Owner	No Issue
12	setMinterAddress	external	access only Owner	No Issue
13	setLimiterAddress	external	access only Owner	No Issue
14	setMinReward	external	access only Owner	No Issue
15	setMinRewardTime	external	access only Owner	No Issue
16	setMintLimitHours	external	access only Owner	No Issue
17	setTaperRate	external	access only Owner	No Issue
18	setRewardTax	external	access only Owner	No Issue
19	ToggleSaleLock	external	access only Owner	No Issue
20	ToggleTransferLock	external	access only Owner	No Issue

OwnershipFacet.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	transferOwnership	external	Function input parameters lack of check	Refer audit findings
3	owner	external	Passed	No Issue

VaultStakingFacet.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyGemOwner	modifier	Passed	No Issue
3	onlyActive	modifier	Passed	No Issue
4	batchAddTovault	external	Deploy error	Refer audit findings
5	addToVault	write	access only Gem Owner	No Issue
6	showStakedAmount	read	Passed	No Issue
7	unstakeTokens	write	access only Gem Owner	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.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

(1) Deploy error: **VaultStakingFacet.sol**

```
function batchAddToVault(
    uint256[] calldata _tokenIds,
    uint256[] calldata _amounts
) external {
    LibGem.DiamondStorage storage dsgem = LibGem.diamondStorage();
    LibMeta.DiamondStorage storage metads = LibMeta.diamondStorage();
    LibUser.DiamondStorage storage userds = LibUser.diamondStorage();
    LibUser.UserData storage user = userds.GetUserData[LibMeta.msgSender()];
    LibVaultStaking.DiamondStorage storage ds = LibVaultStaking
        .diamondStorage();
    for (uint256 index = 0; index < _tokenIds.length; index++) {
        require(
            LibERC721._ownerOf(_tokenIds[index]) == LibERC721.msgSender(),
            "You don't own this gem"
        );
        require(LibGem._isActive(_tokenIds[index]), "Gem is deactivated");
        uint256 _pendingRewards = LibGem._taperCalculate(_tokenIds[index]);
        require(
            _amounts[index] >= _pendingRewards,
            "Not enough pending rewards"
        );
        LibGem.Gem storage gem = dsgem.GemOf[_tokenIds[index]];

        uint256 charityAmount = (metads.CharityRate * _amounts[index]) /
            1000;
        _amounts[index] = _amounts[index] - charityAmount;
```

Calldata arrays are read-only. So _amounts variable cannot be updated.

Resolution: We suggest correcting the code by changing calldata to memory for the _amounts.

(2) Owner can stop the sell: **DefoLimiter.sol**

```
function setMaxPercentage(uint256 _newPercentage) external onlyRole(DEFAULT_ADMIN_ROLE) {
    sellLimitVsTotalSupply = _newPercentage;
}
```

The owner can set sellLimitVsTotalSupply to 0 using setMaxPercentage. Hence users cannot sell their tokens.

Resolution: We suggest using some minimum limit to set sellLimitVsTotalSupply.

Low

(1) For loops can be merged: **Redeem.sol**

In the Redeem function, these loops can be merged and use conditions to check for the last element and avoid transferFrom for that.

Resolution: We suggest using only 1 loop for each balance and use conditions to check for the last element and avoid transferFrom for that.

(2) Function input parameters lack of check:

Variable validation is not performed in below functions :

GemFacet.sol

- RedeemMint
- RedeemMintBooster

OwnershipFacet.sol

- transferOwnership

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) Unused variable: **DefoErc20.sol**

The MAXSELLLIMIT is defined but not used anywhere.

Resolution: We suggest removing unused variables.

(2) Critical operation lacks event log: **GemFacet.sol**

Missing event log for:

- Compound
- BatchClaimRewards
- BatchMaintenance
- RedeemMint
- RedeemMintBooster
- BoostGem
- MintGem
- Maintenance
- BatchMaintenance
- BatchClaimRewards

Resolution: We suggest adding logs for the listed events.

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:

- transferLog: DefoLimiter DefoToken owner can call before any token transfer.
- setMaxPercentage: DefoLimiter role owner can set maximum percentage.
- setTokenAddress: DefoLimiter role owner can set token address.
- setLPAddress: DefoLimiter role owner can set LP address.
- setLPManager: DefoLimiter role owner can set token manager address.
- setDiamond: DefoLimiter role owner can set diamond address.
- setTimeframeExpiration: DefoLimiter role owner can set time frame expiration.
- editWhitelist: DefoLimiter role owner can update whitelist addresses and status.
- editBlocklist: DefoLimiter role owner can update blocklist addresses and status.
- setAllowance: LpManager owner can set allowance status.
- shouldLiquify: LpManager owner should set liquify status.
- setBufferThreshHold: LpManager owner can set buffer threshold value.
- setNodeAddress: Redeem owner can set node address.
- startTimer: Redeem owner can set start timer.
- flipActive: Redeem owner can flip active redeem.
- addToWhitelist: NodeLimiterFacet owner can add new address in whitelist.
- initialize: OwnerFacet owner can initialize address.
- setGemSettings: OwnerFacet owner can create a new gem type or change a gem type settings.
- setAddressAndDistTreasury: OwnerFacet owner can set distribution treasury addresses.
- setAddressAndDistRewardPool: OwnerFacet owner can set distribution reward addresses.
- setAddressDonation: OwnerFacet owner can set donation addresses.
- setAddressAndDistTeam: OwnerFacet owner can set address and dist team.
- setAddressAndDistLiquidity: OwnerFacet owner can set address and dist liquidity address.

- setAddressVault: OwnerFacet owner can set new vault address.
- setBaseURI: OwnerFacet owner can set base uri.
- setMinterAddress: OwnerFacet owner can set new minter address.
- setLimiterAddress: OwnerFacet owner can set new limiter address.
- setMinReward: OwnerFacet owner can set minimum reward value.
- setMinRewardTime: OwnerFacet owner can set minimum reward time value.
- setMintLimitHours: OwnerFacet owner can set minimum limit hours time value.
- ChangePaymentToken: OwnerFacet owner can change payment token address.
- setTaperRate: OwnerFacet owner can set taper rate value.
- setRewardTax: OwnerFacet owner can set reward tax value.
- ToggleSaleLock: OwnerFacet owner can toggle sale lock.
- ToggleTransferLock: OwnerFacet owner can toggle transfer lock.
- batchAddTovault: VaultStakingFacet owner can set batch address to vault address.
- addToVault: VaultStakingFacet owner can add new vault address.
- unstakeTokens: VaultStakingFacet owner can unstake tokens.

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

Conclusion

We were given a contract code in the form of github links. And we have used all possible tests based on given objects as files. We have observed some issues in the smart contracts. So, **it's good to go to production by fixing those issues**.

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.

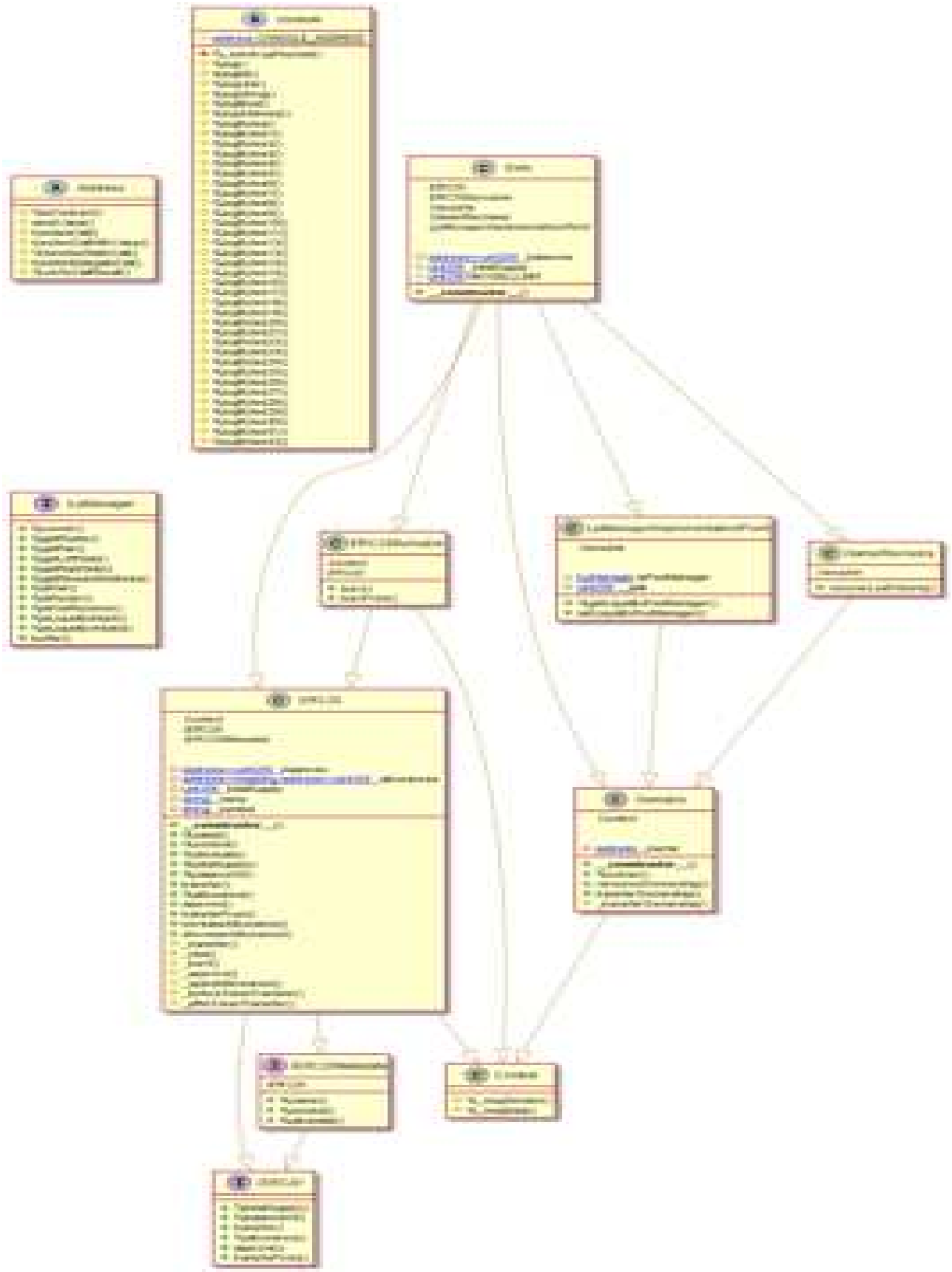
Technical Disclaimer

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.

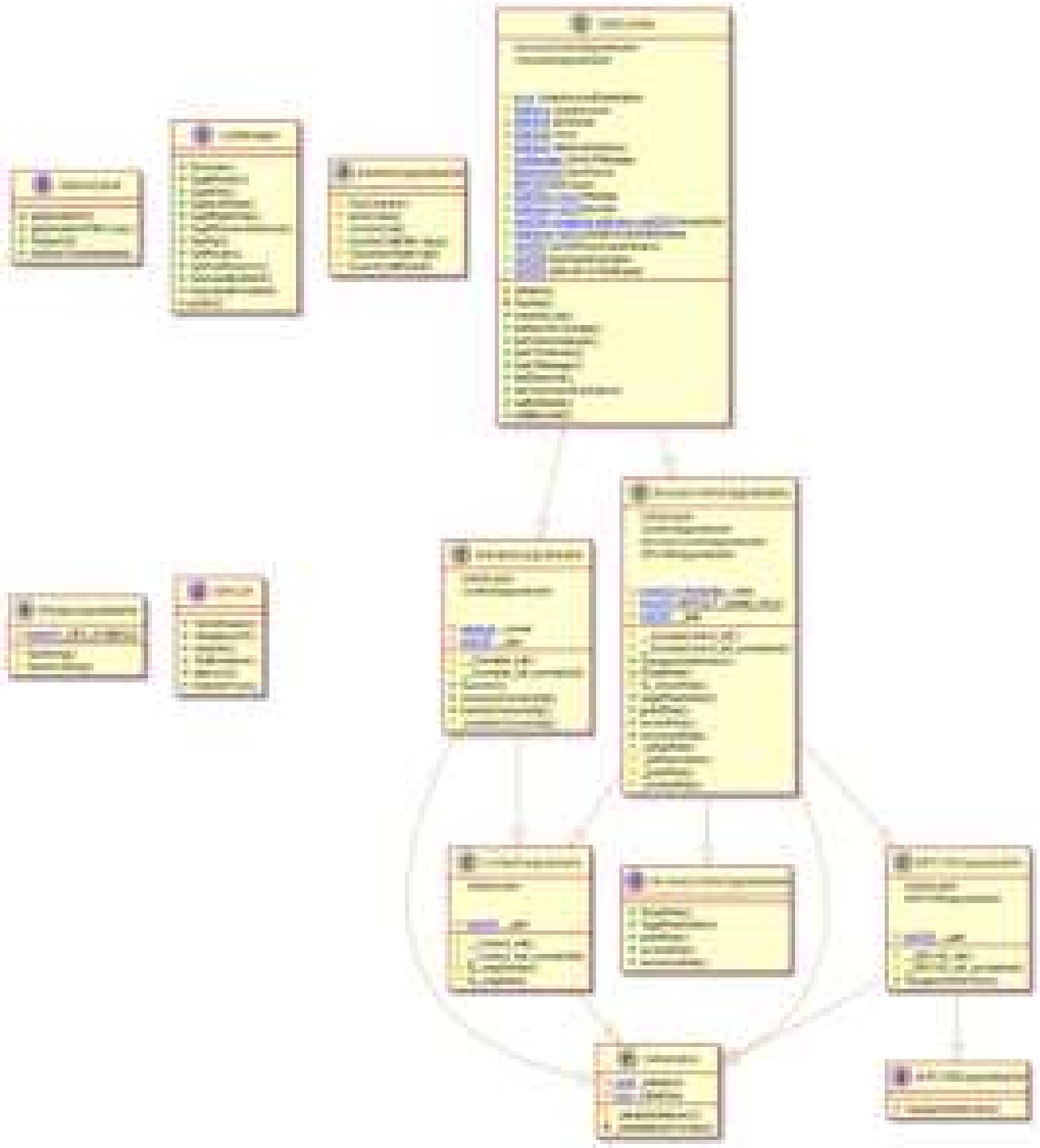
Appendix

Code Flow Diagram - Defo Contracts Protocol

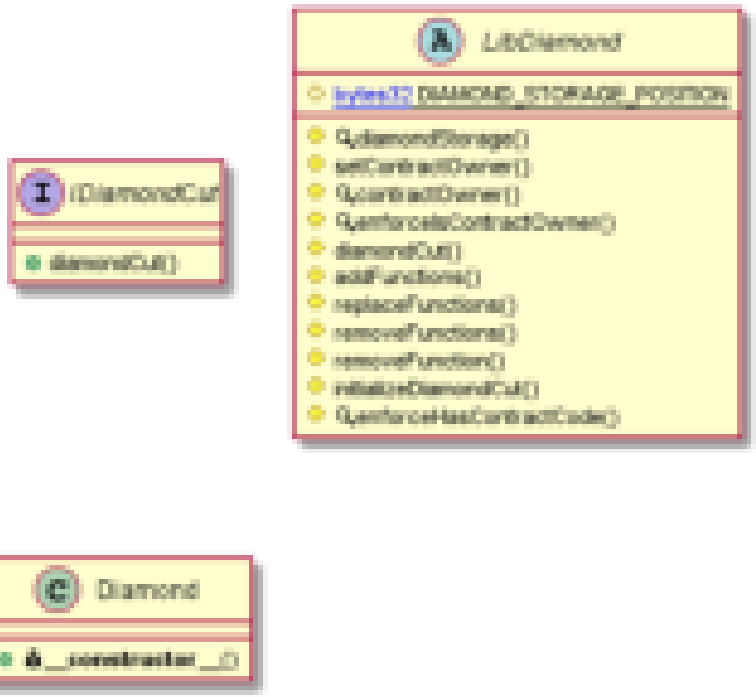
DefoErc20 Diagram



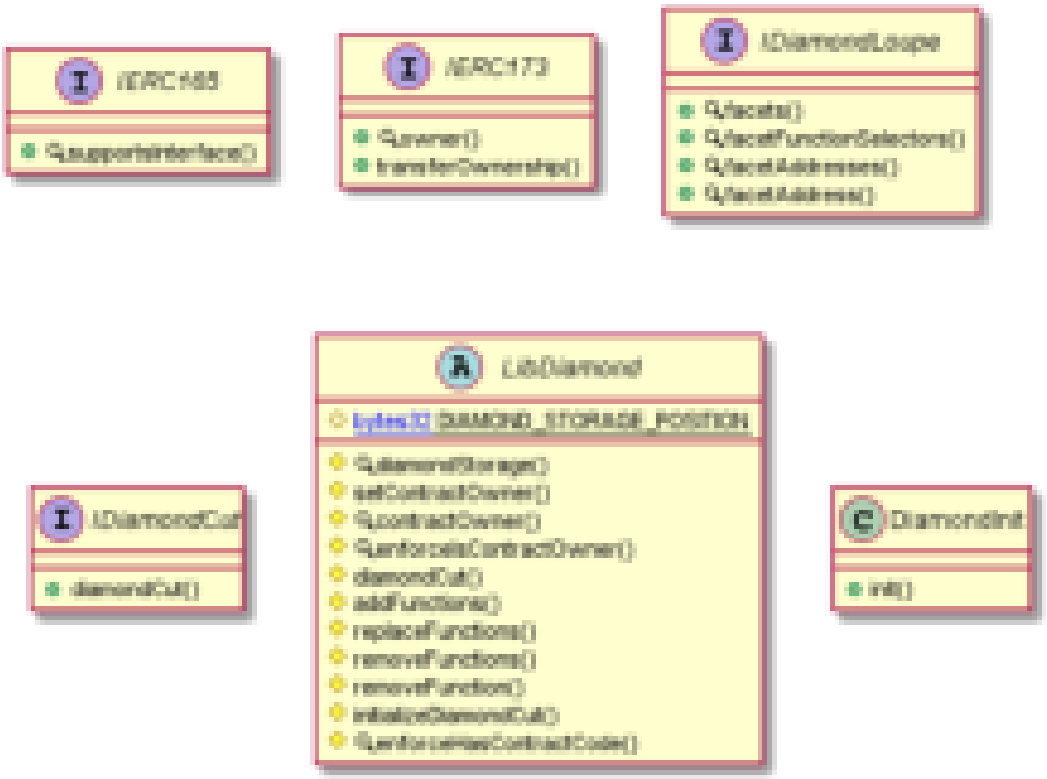
DefoLimiter Diagram



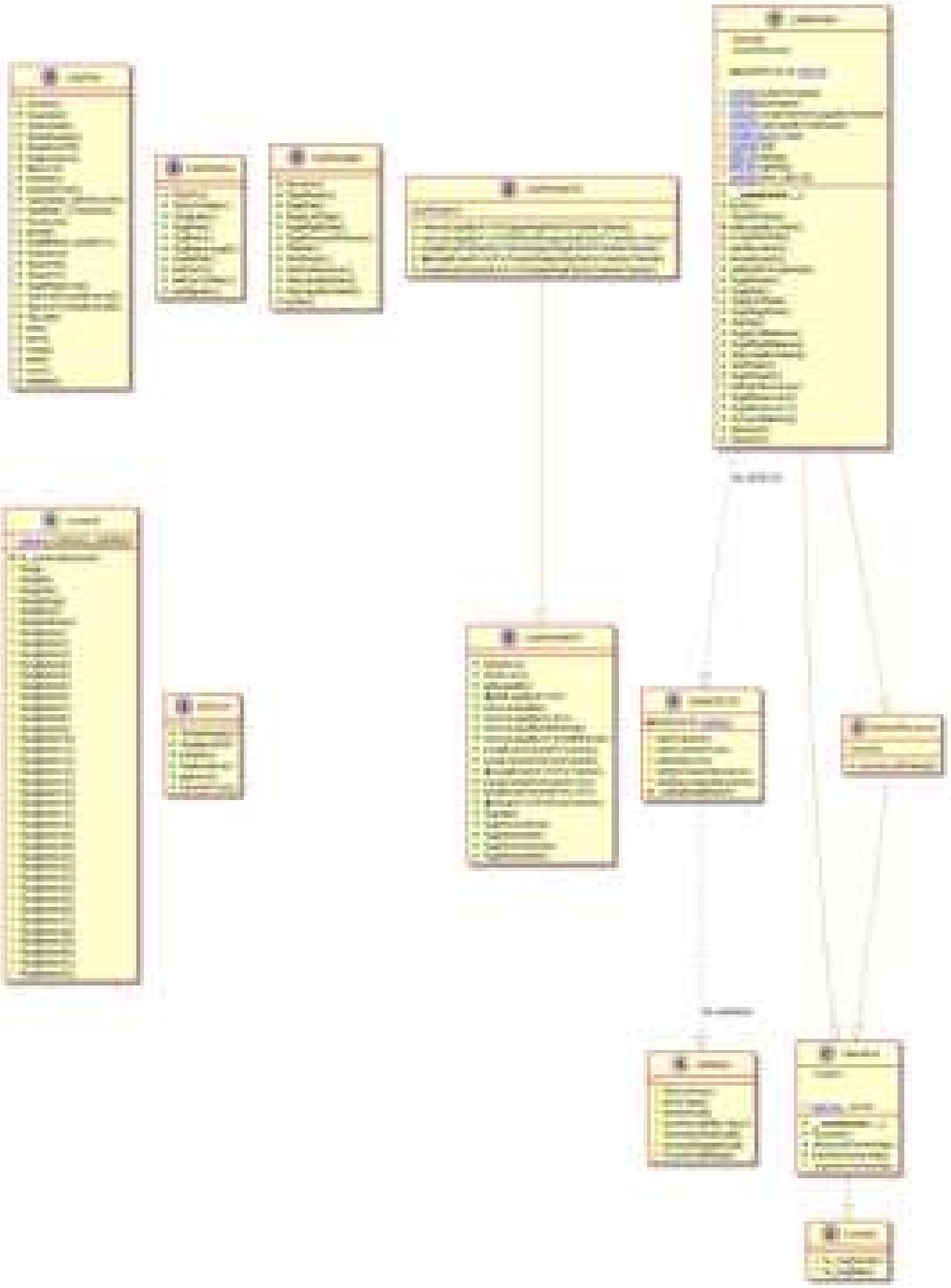
Diamond Diagram



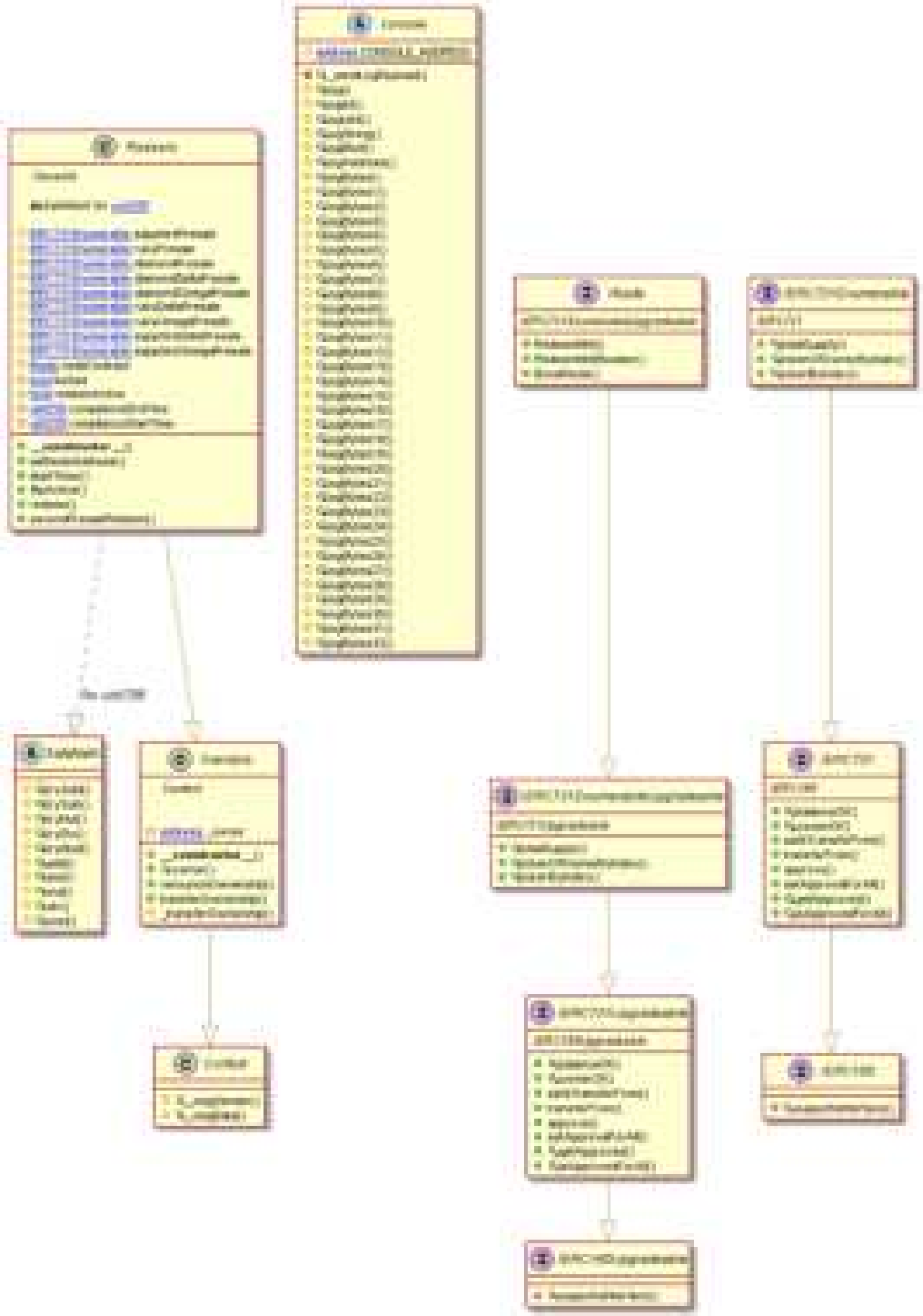
DiamondInit Diagram



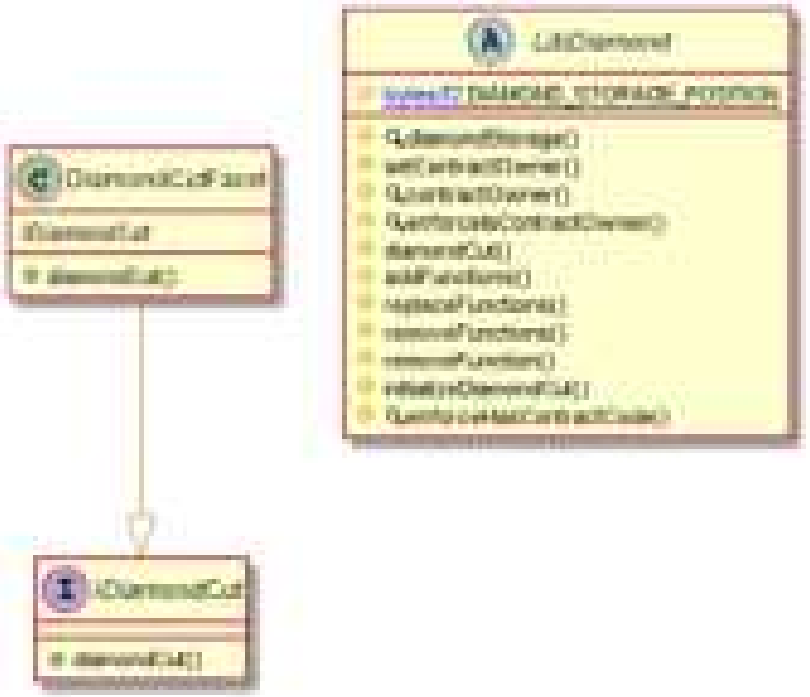
LpManager Diagram



Redeem Diagram

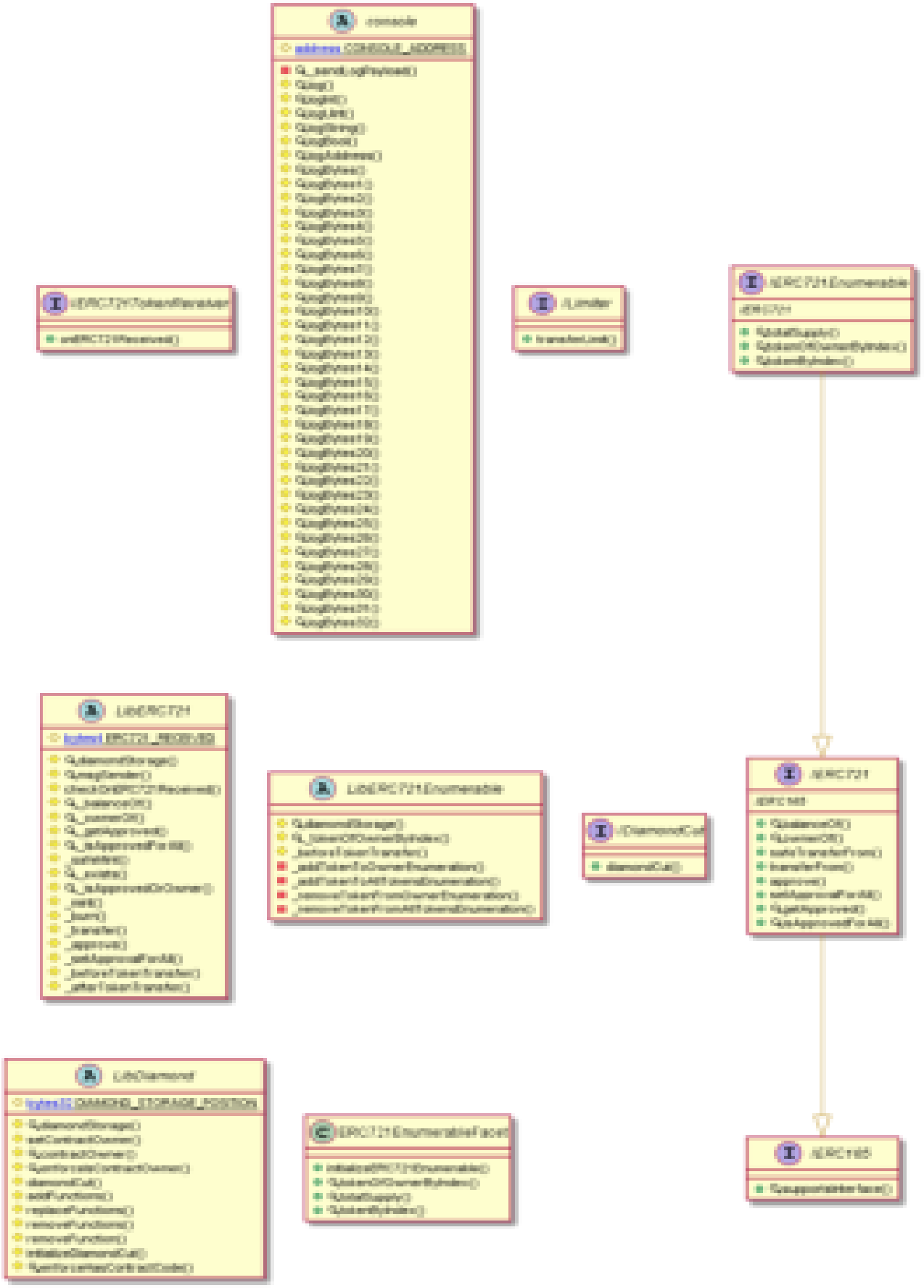


DiamondCutFacet Diagram

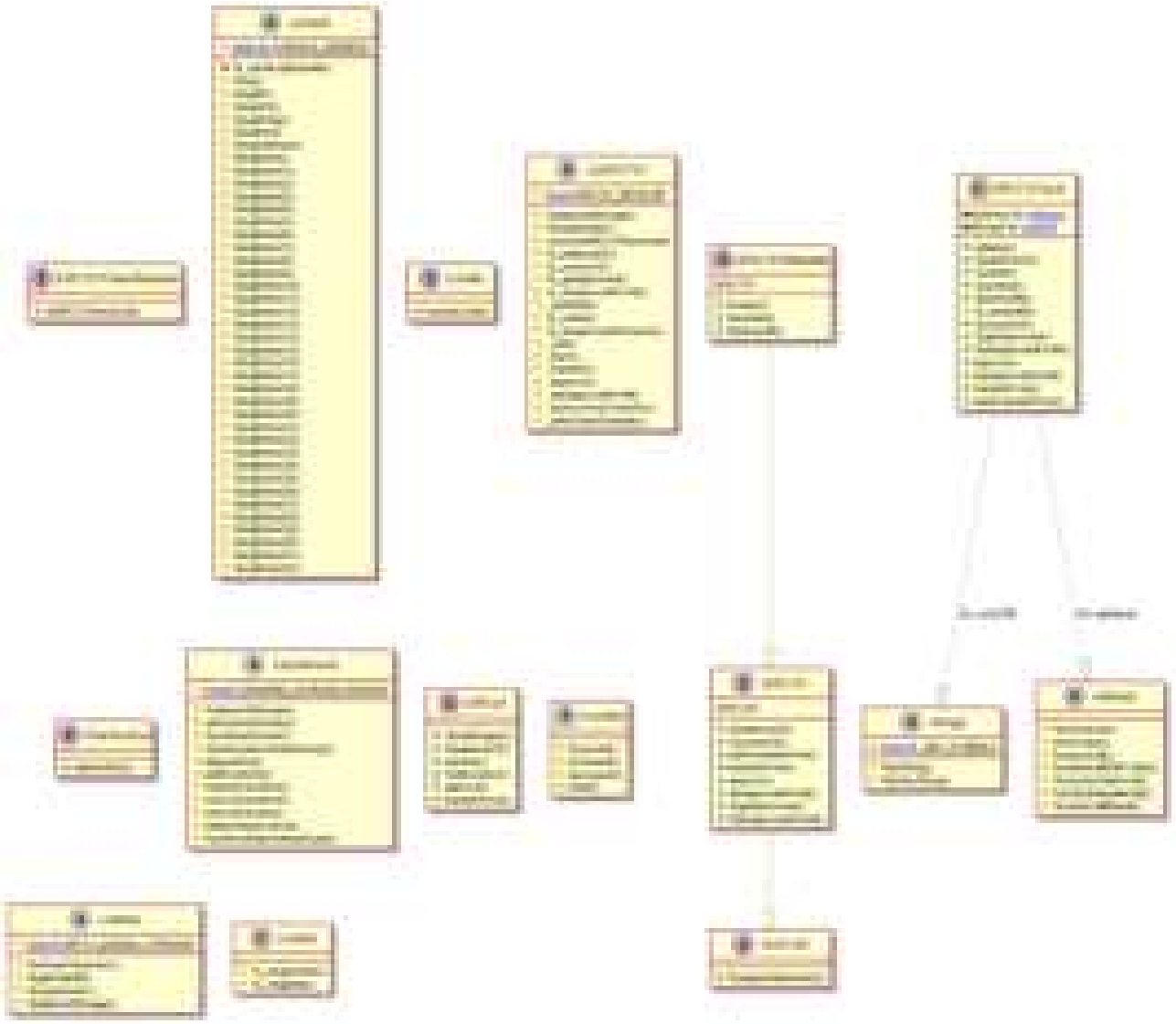


DiamondLoupeFacet Diagram

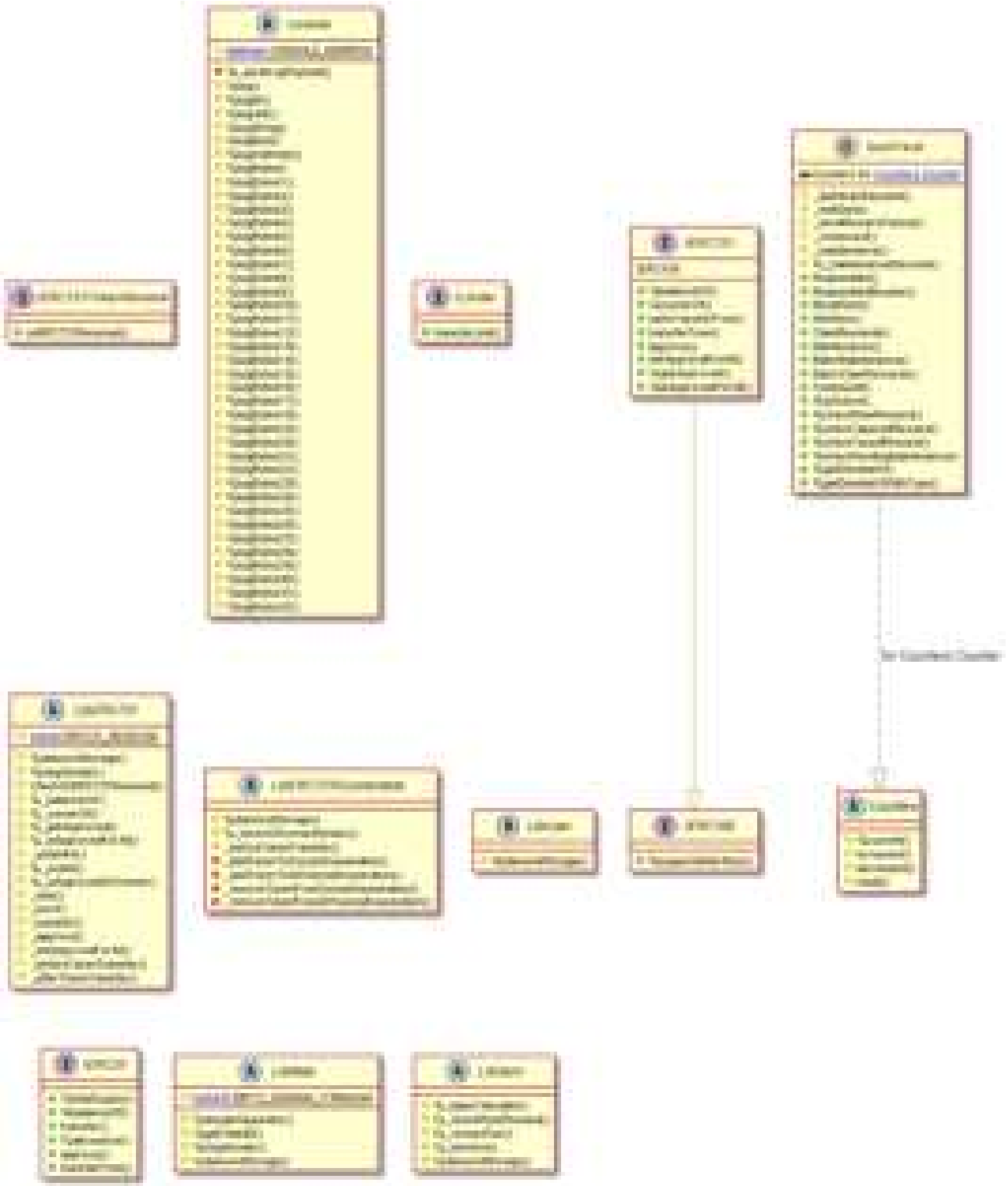
ERC721Enumerable Diagram



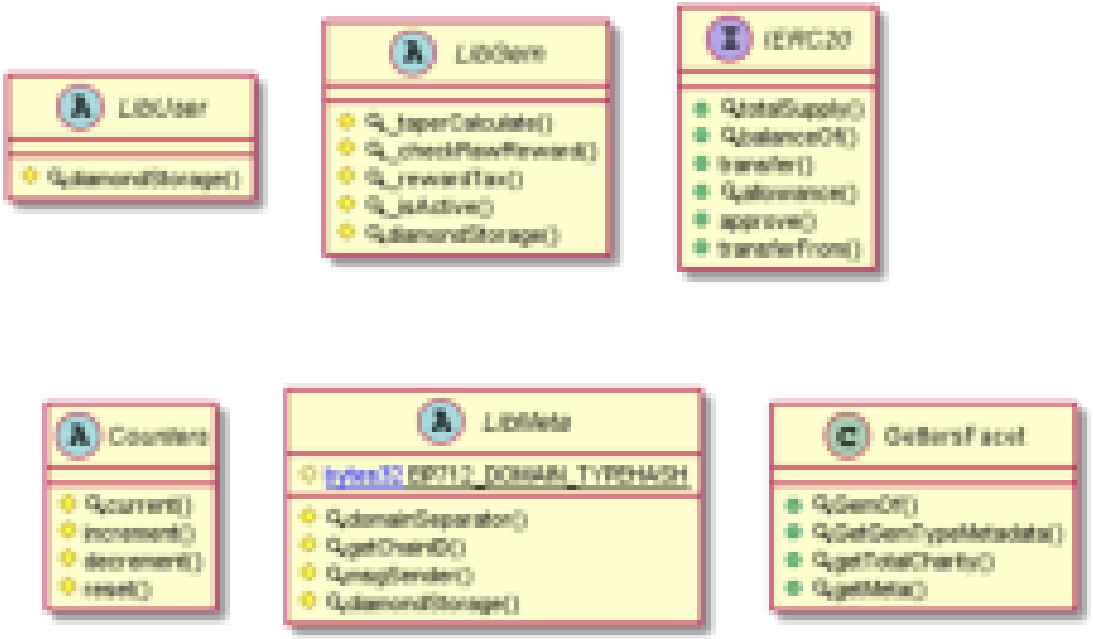
ERC721Facet Diagram



GemFacet Diagram

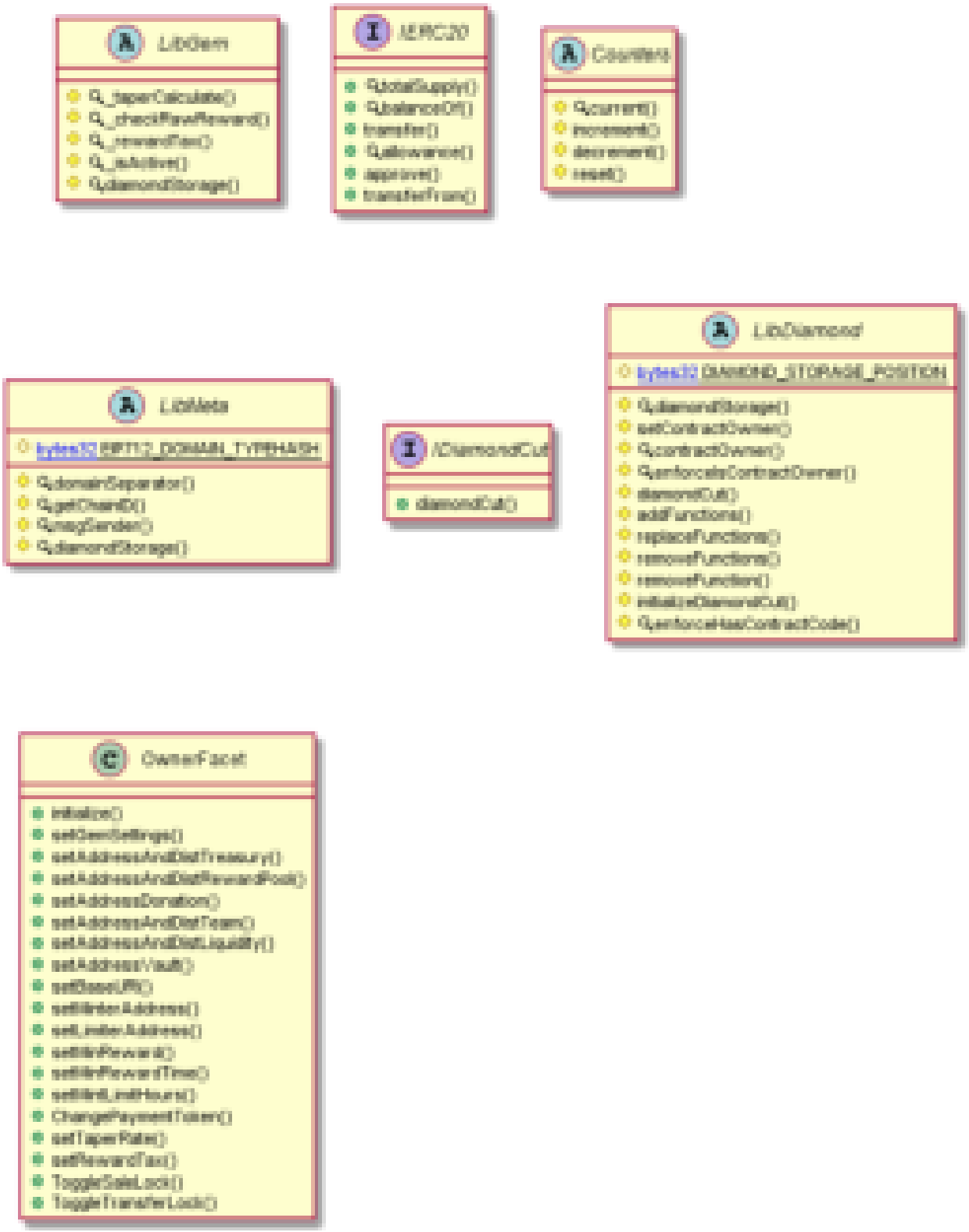


GemGettersFacet Diagram



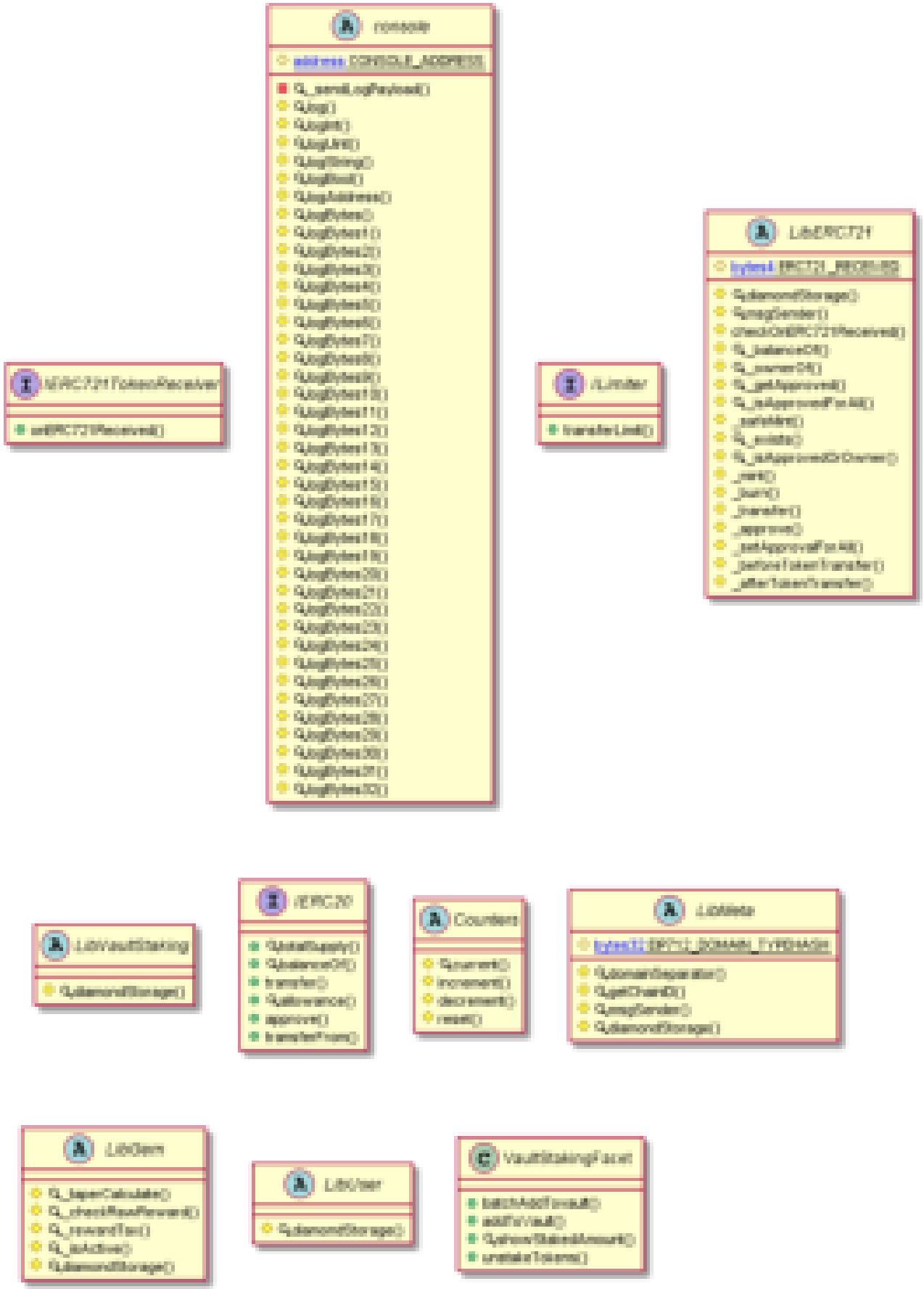
NodeLimiterFacet Diagram

OwnerFacet Diagram



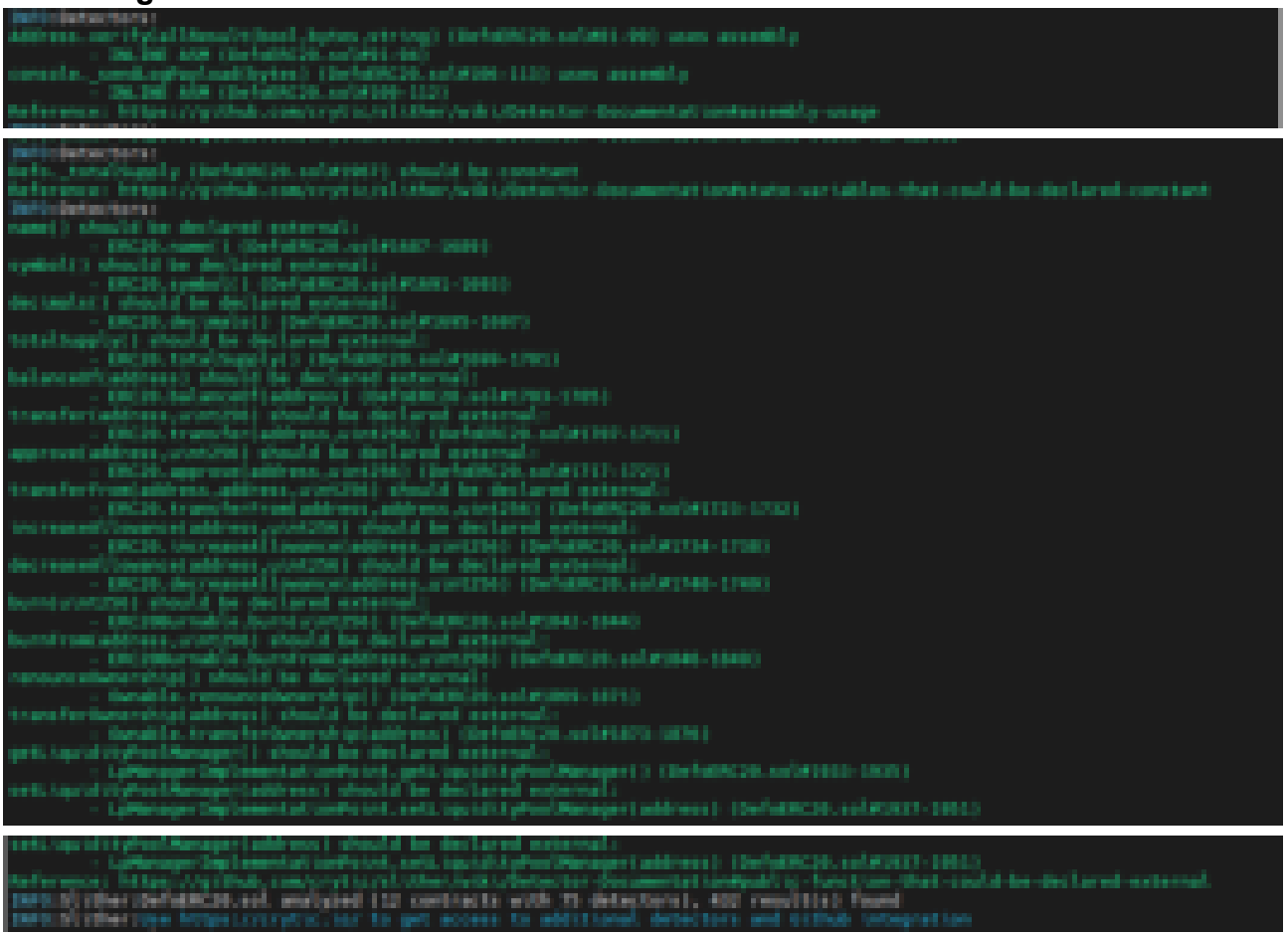
OwnershipFacet Diagram

VaultStakingFacet Diagram

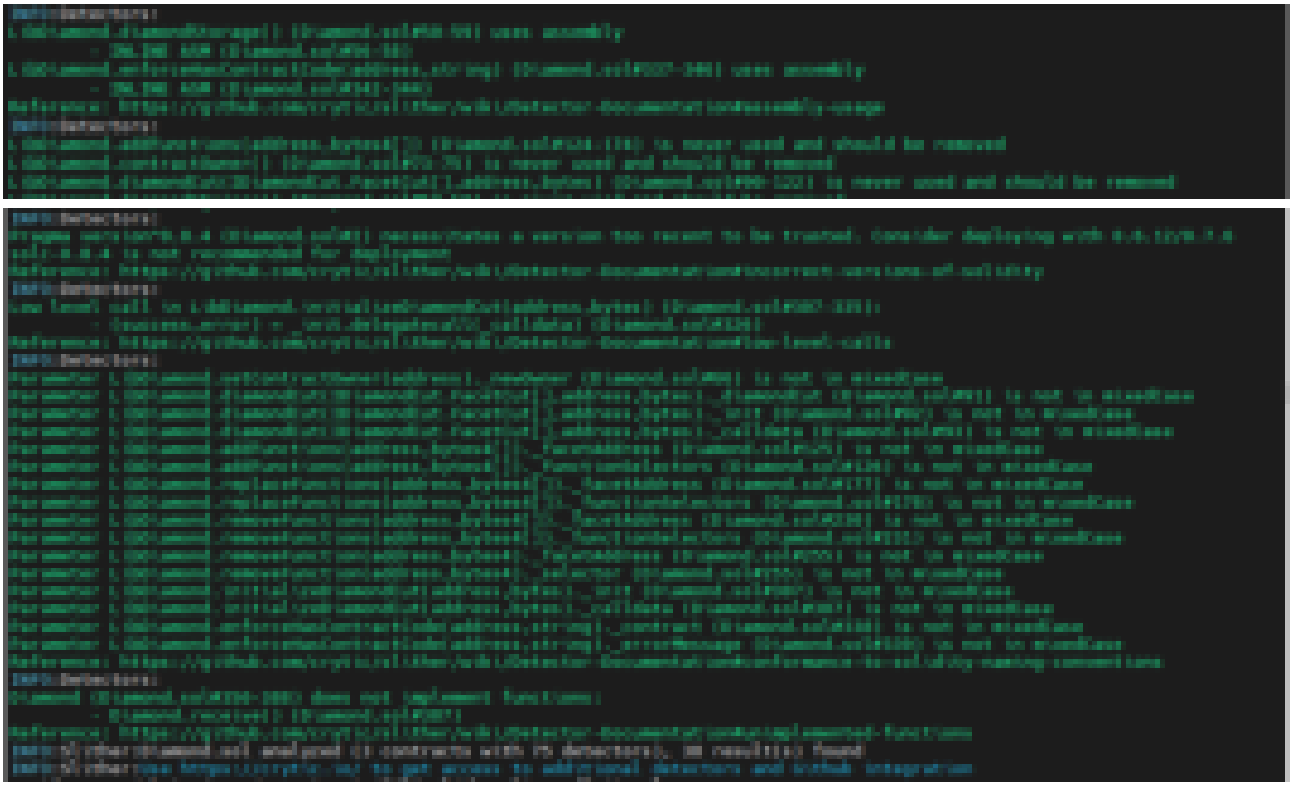


Slither Results Log

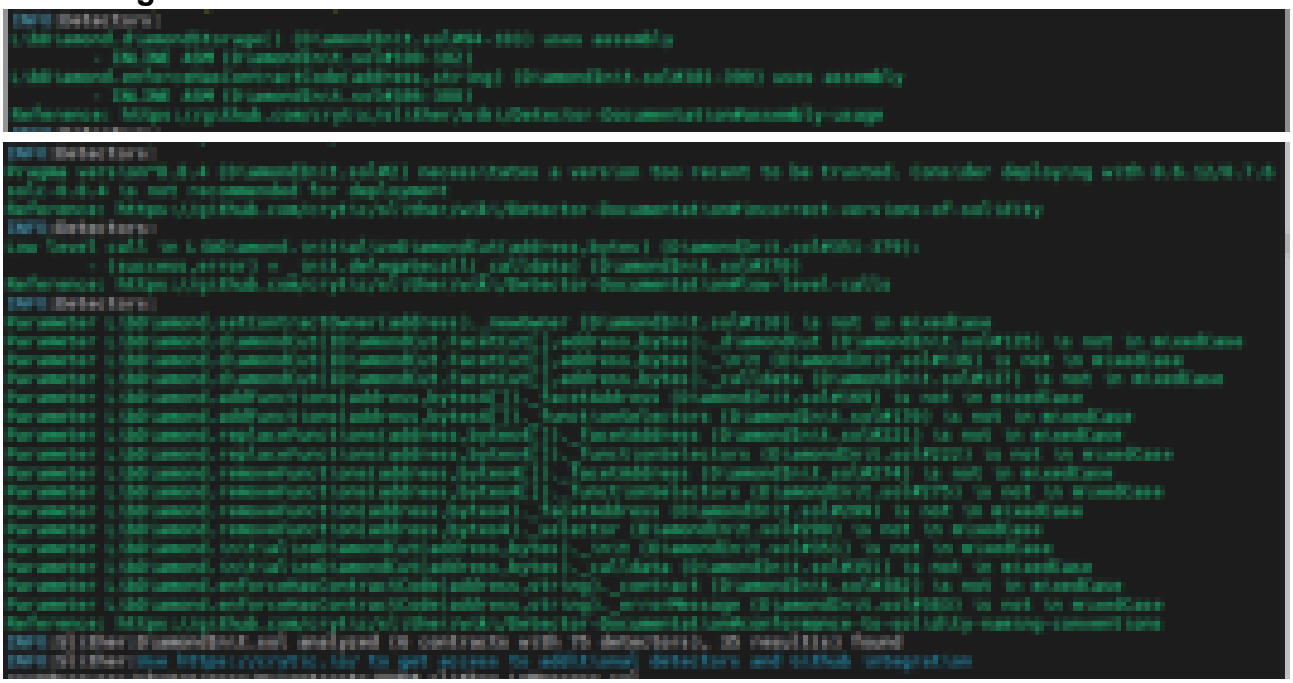
Slither log >> DefoErc20.sol



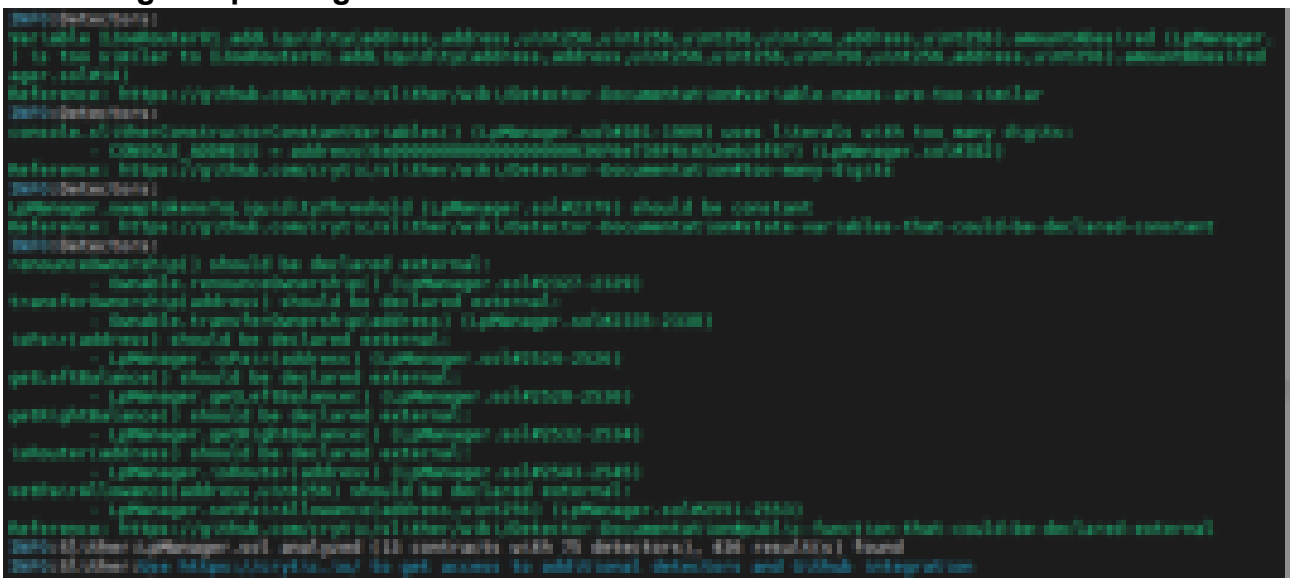
Slither log >> Diamond.sol



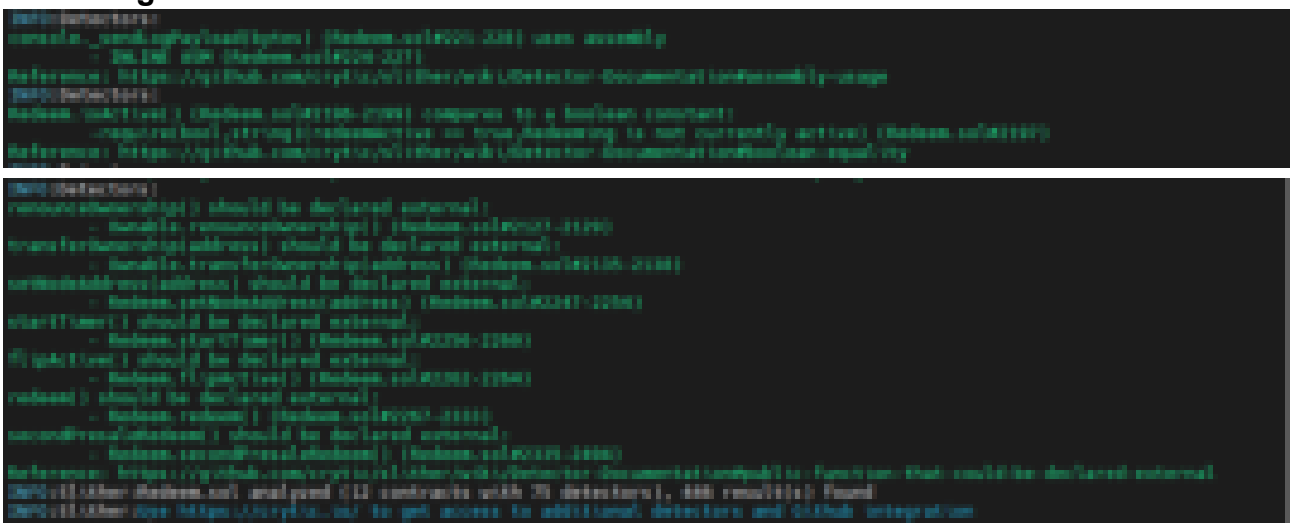
Slither log >> DiamondInit.sol



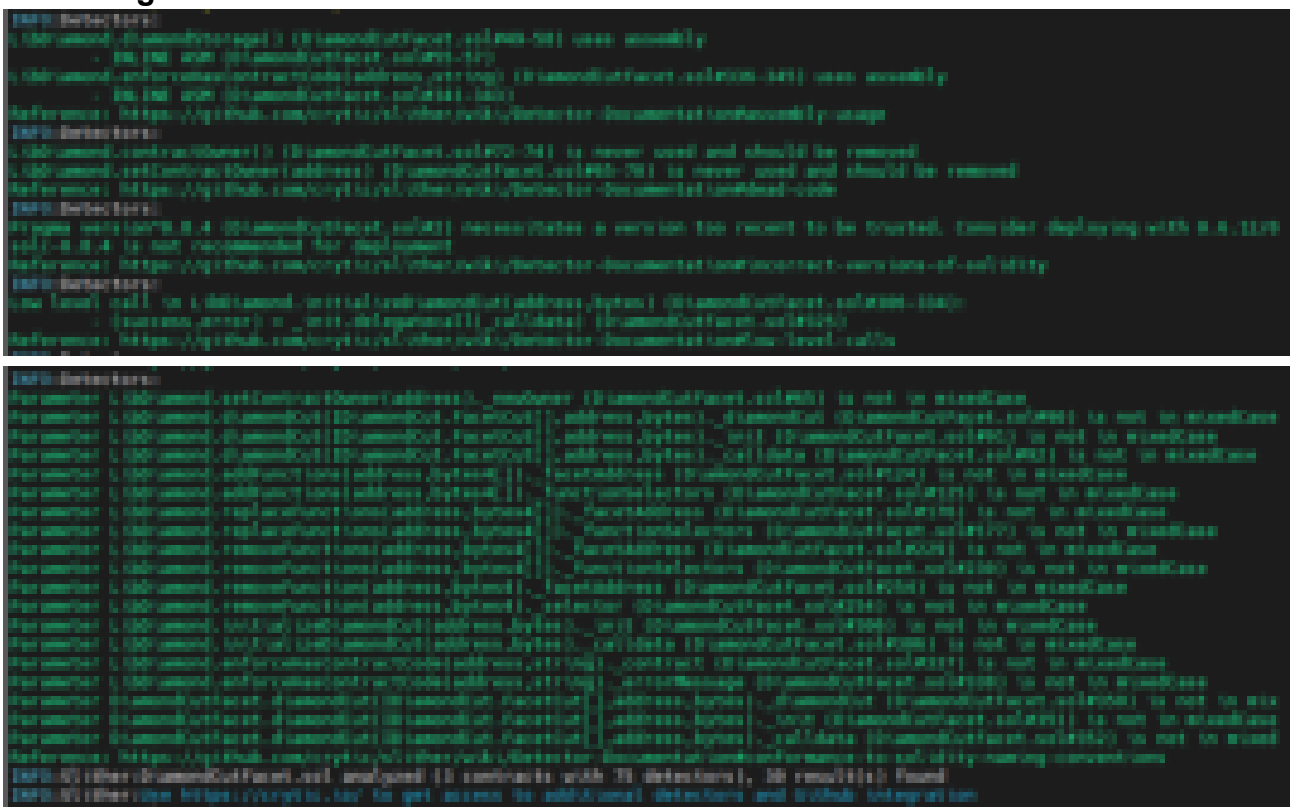
Slither log >> LpManager.sol



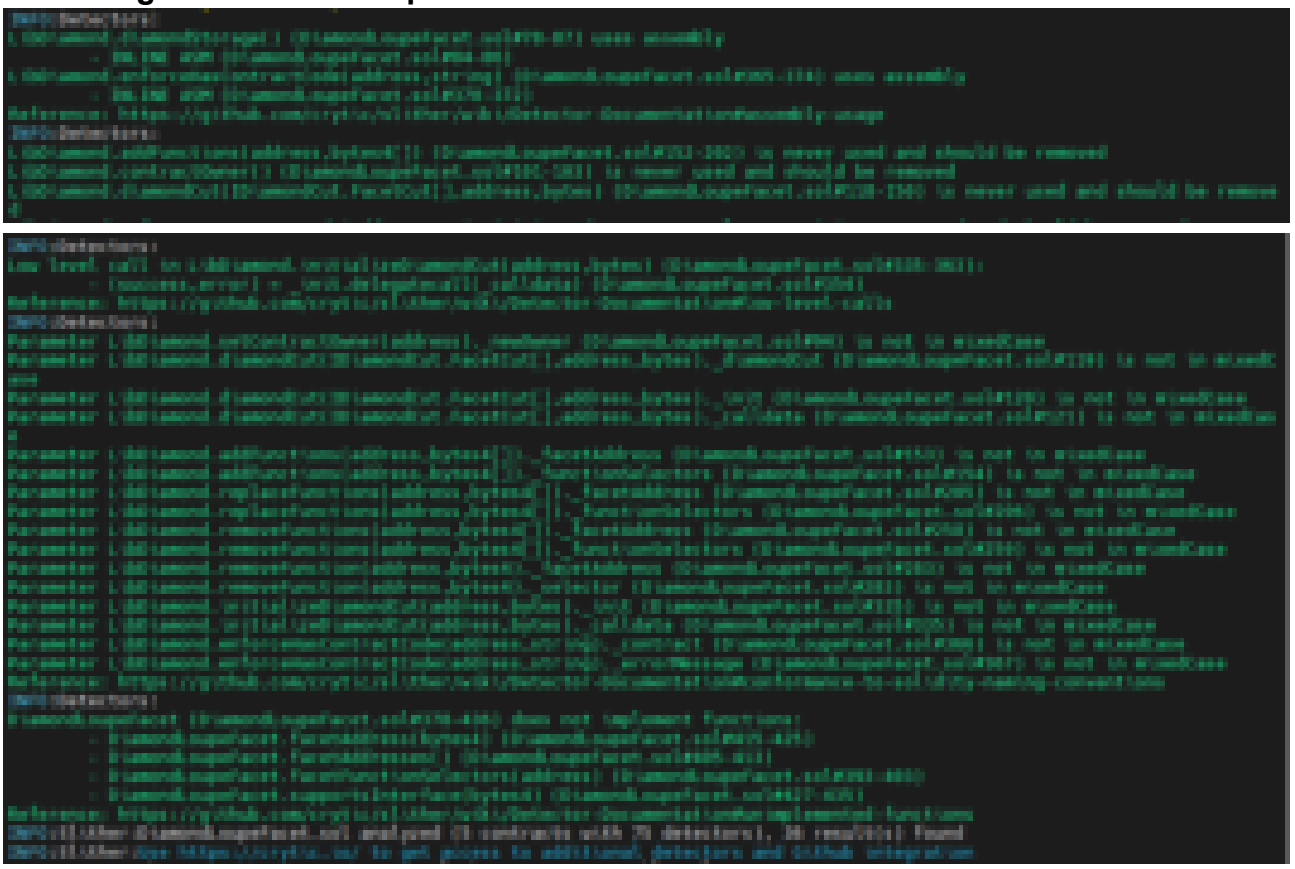
Slither log >> Redeem.sol



Slither log >> DiamondCutFacet.sol

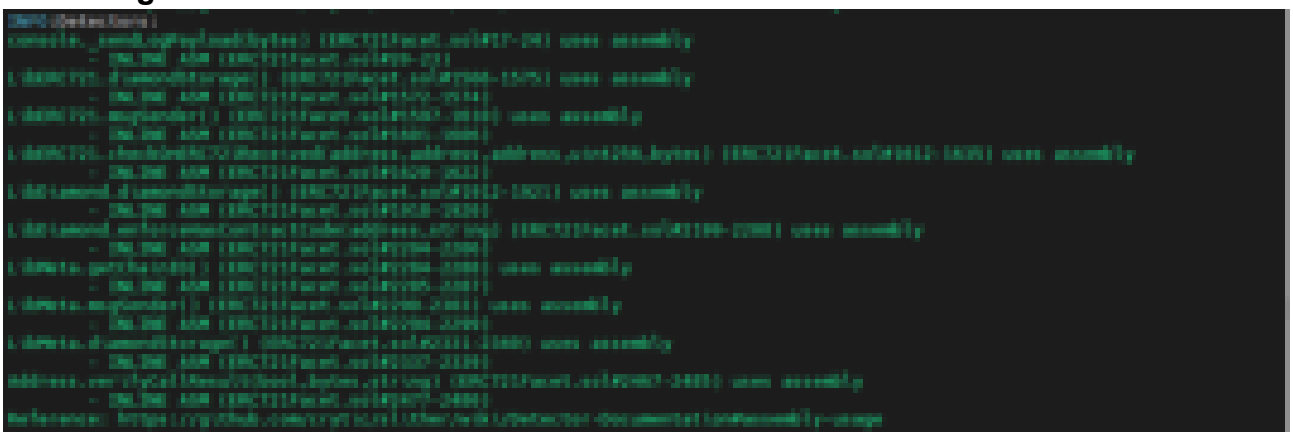


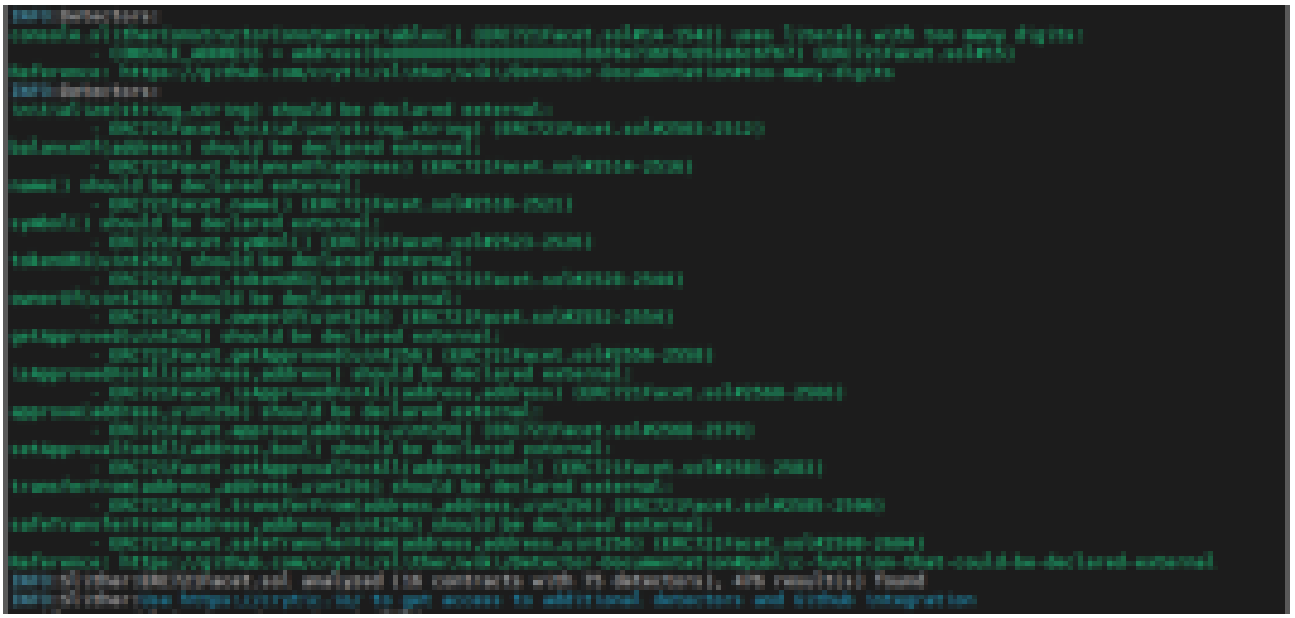
Slither log >> DiamondLoupeFacet.sol



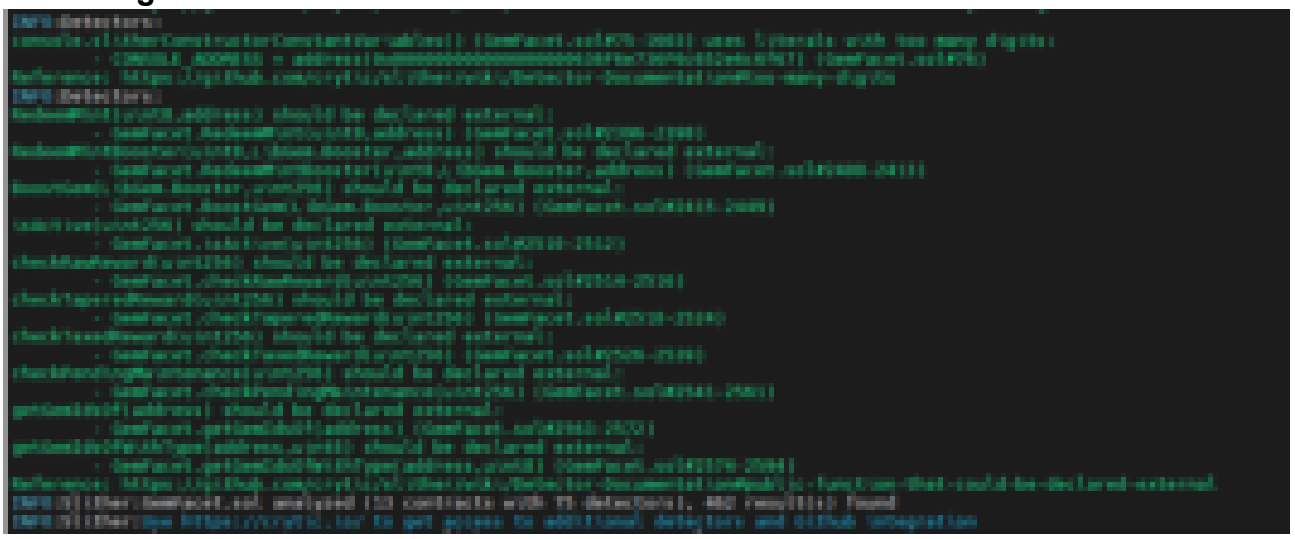
Slither log >> ERC721Enumerable.sol

Slither log >> ERC721Facet.sol

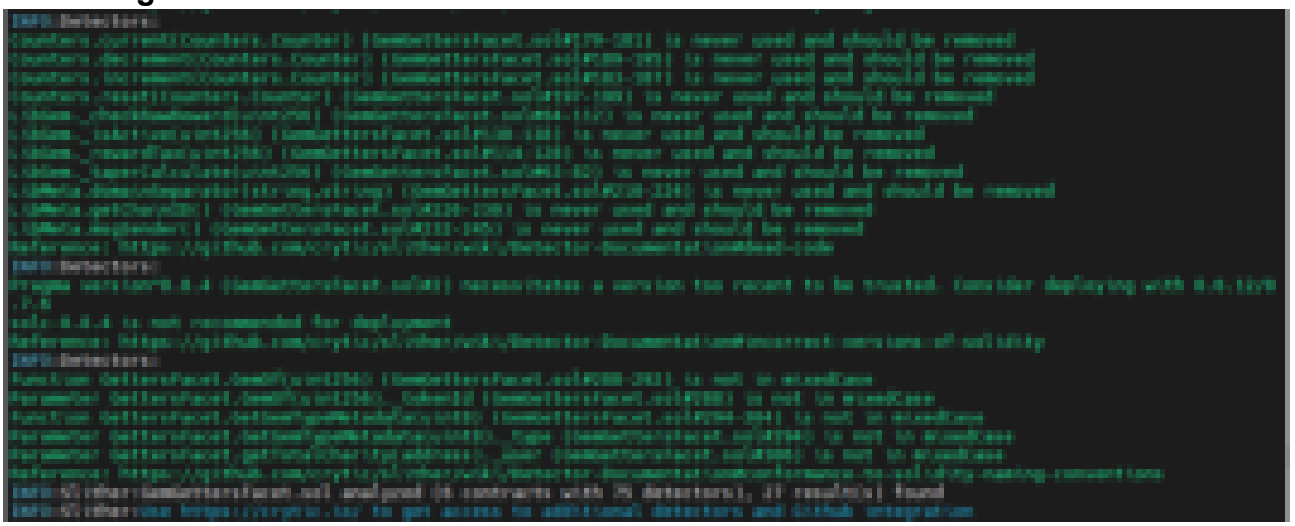




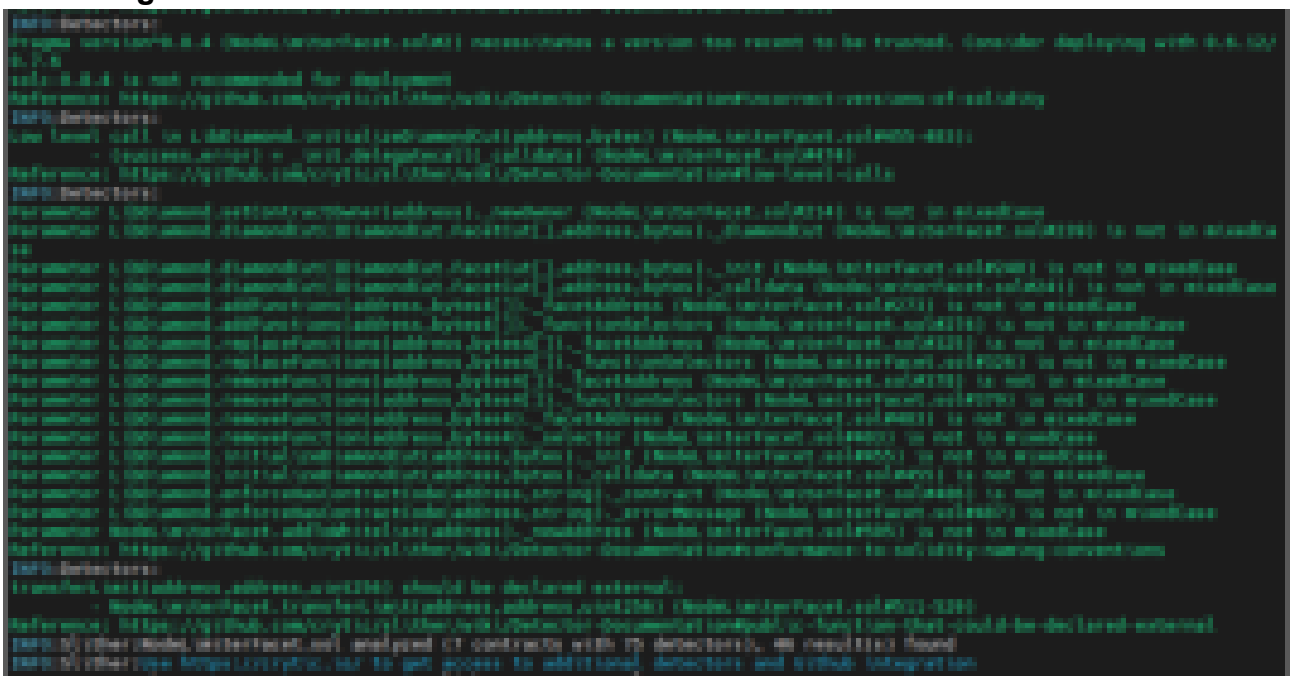
Slither log >> GemFacet.sol



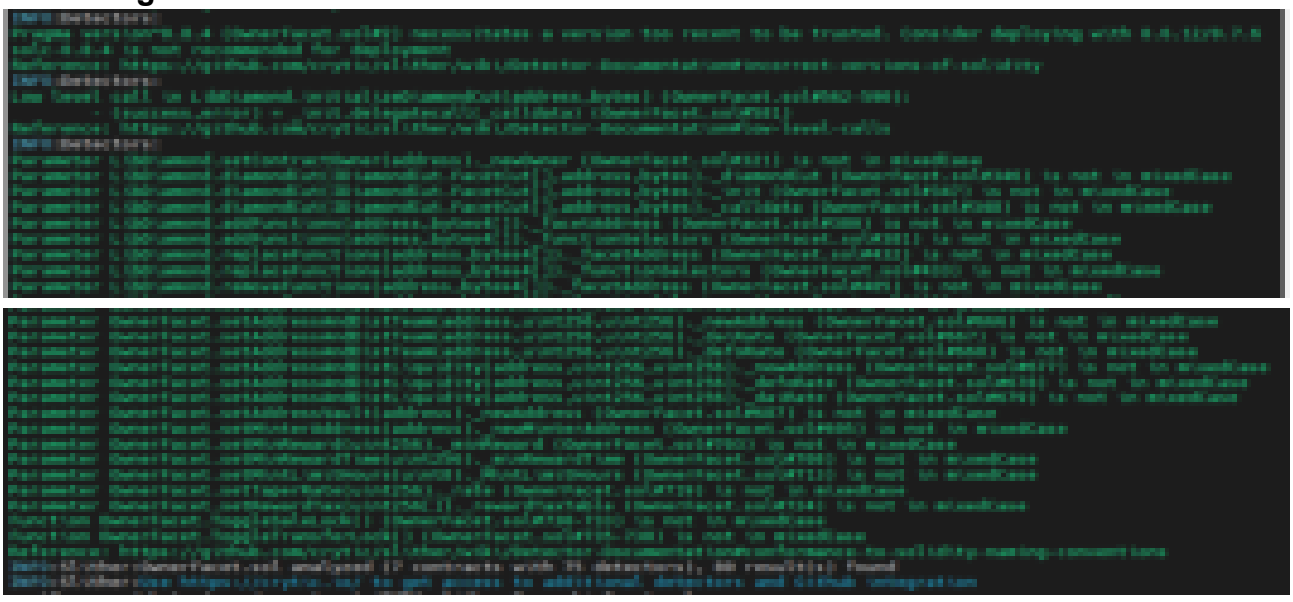
Slither log >> GemGettersFacet.sol



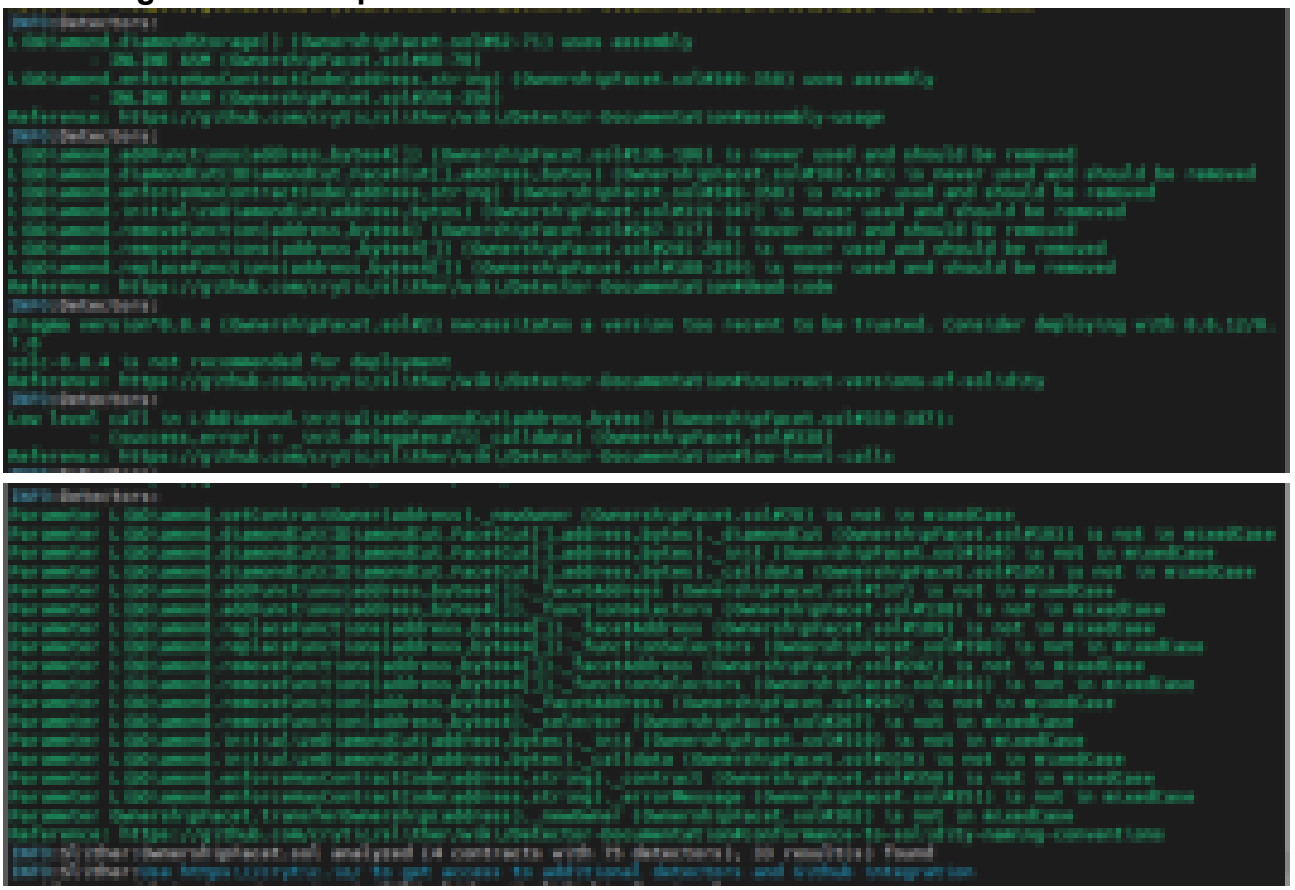
Slither log >> NodeLimiterFacet.sol



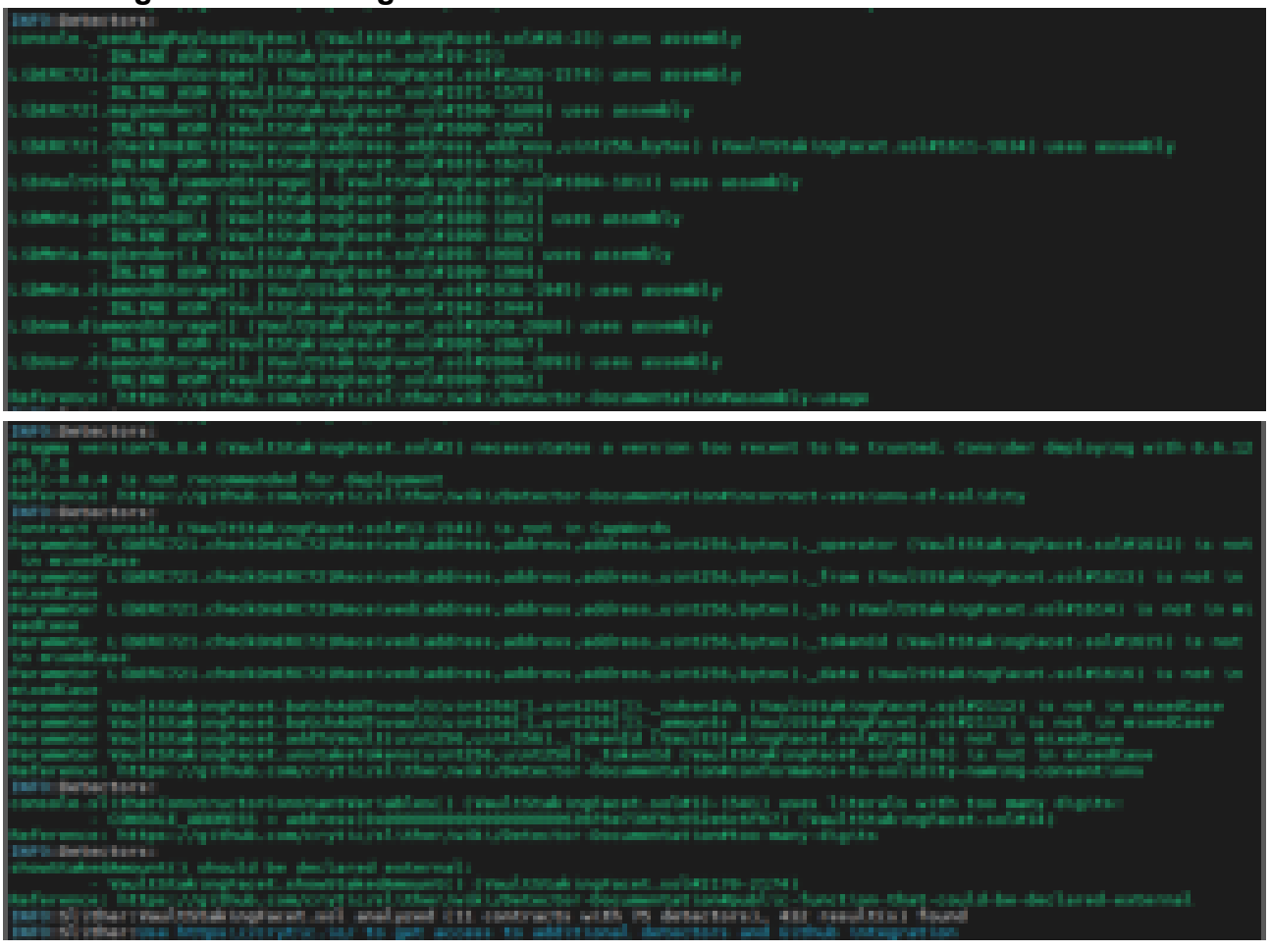
Slither log >> OwnerFacet.sol



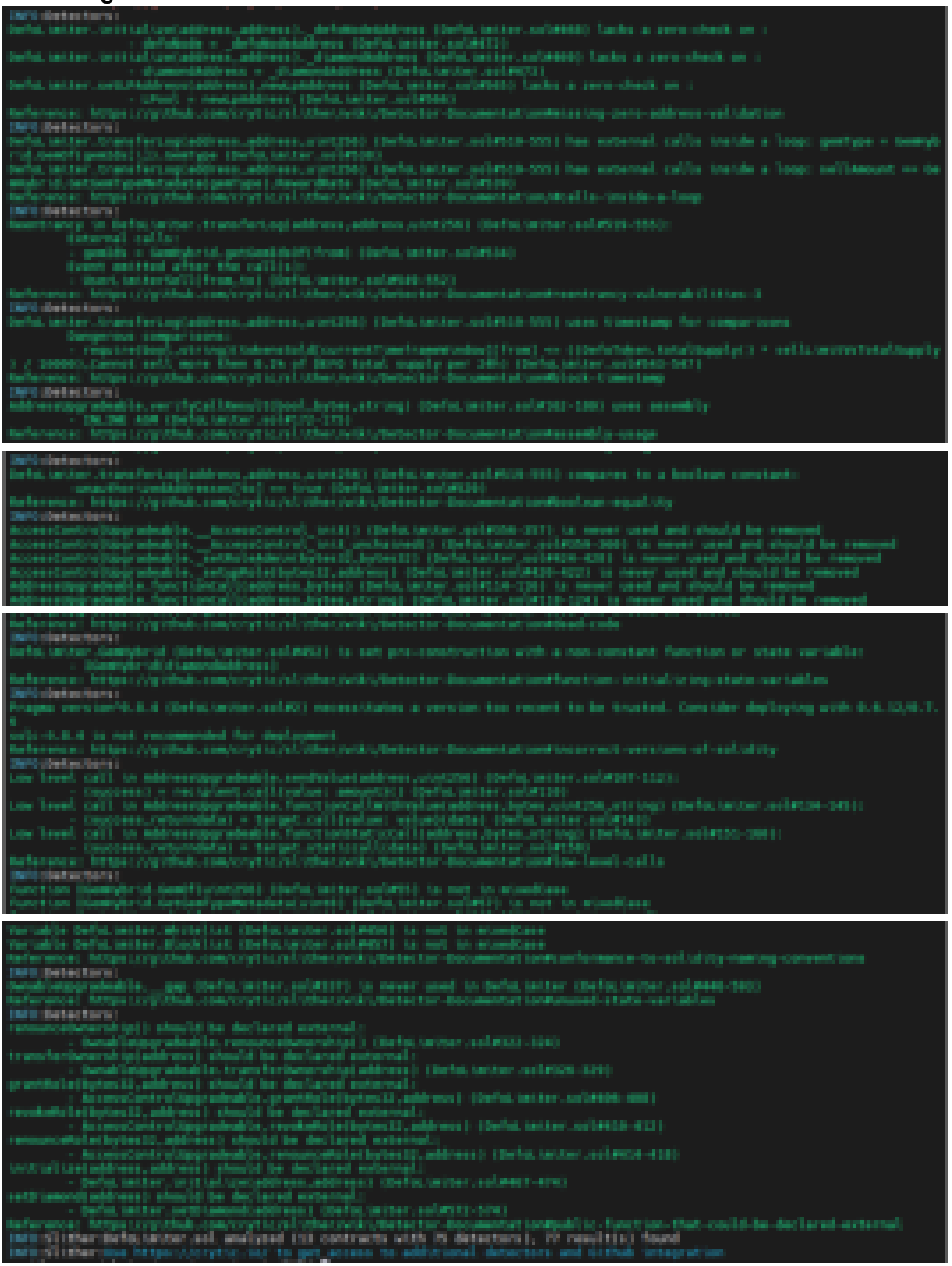
Slither log >> OwnershipFacet.sol



Slither log >> VaultStakingFacet.sol



Slither log >> DefoLimiter.sol



Solidity Static Analysis

DefoErc20.sol

DefoLimiter.sol

Security

Check-effects-interaction:

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

more

Pos: 372:4

Gas & Economy

Gas costs:

Gas requirement of function DefoLimiter.setLPAddress 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: 331:4

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: 381:16

Miscellaneous

Constant/View/Pure functions:

IERC20.transferFrom(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 679:4

Similar variable names:

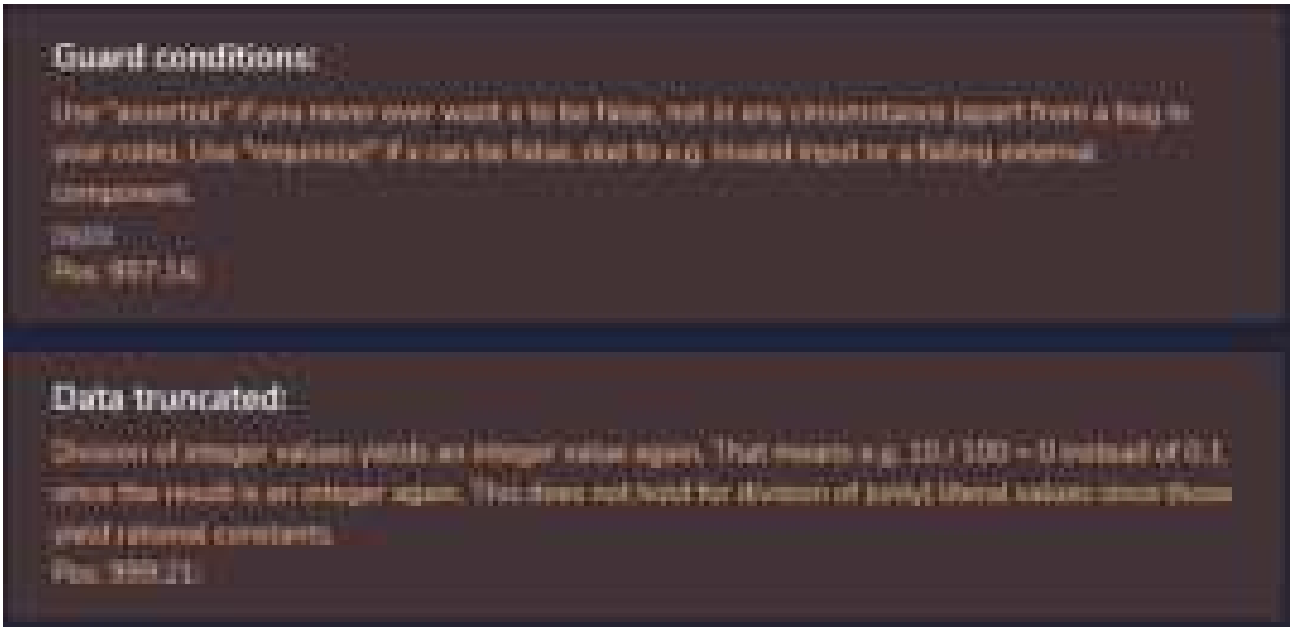
AccessControl.hasRole(bytes32,address) : Variables have very similar names "_msgSender" and "value". Note: Modifiers are currently not considered by this static analysis.

Pos: 799:28

No return:

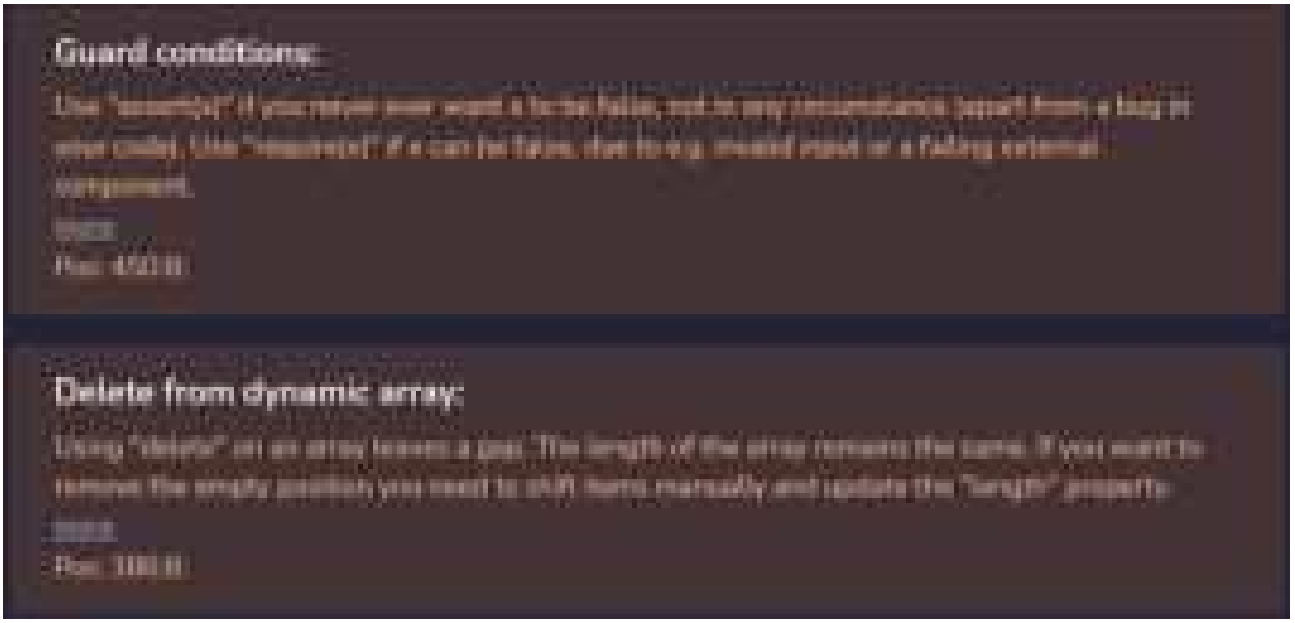
IERC20.transferFrom(address,address,uint256): Defines a return type but never explicitly returns a value.

Pos: 679:4



Diamond.sol

DiamondInit.sol



LpManager.sol

Redeem.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 behavior, use the Solidity library feature if possible.

more

Pos: 333:49

Gas & Economy

Gas costs:

Gas requirement of function DiamondCutFacet.diamondCut 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: 330:4

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: 377:8

Miscellaneous

Constant/View/Pure functions:

LibDiamond.enforceHasContractCode(address,string) is constant but potentially should not be

more

Pos: 267:4

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: 375:8

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: 311:8

DiamondLoupeFacet.sol

ERC721Enumerable.sol

Security

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 2012:8

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 behavior use the Solidity library feature if possible.

more

Pos: 2645:49

Gas & Economy

Gas costs:

Gas requirement of function ERC721Enumerable.safeTransferFrom(address,address,uint256,bytes) 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)

more

Pos: 2609:8

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: 2559:8

Miscellaneous

Constant/View/Pure functions:

ERC721Enumerable.tokenByIndex(uint256) : is constant but potentially should not be.

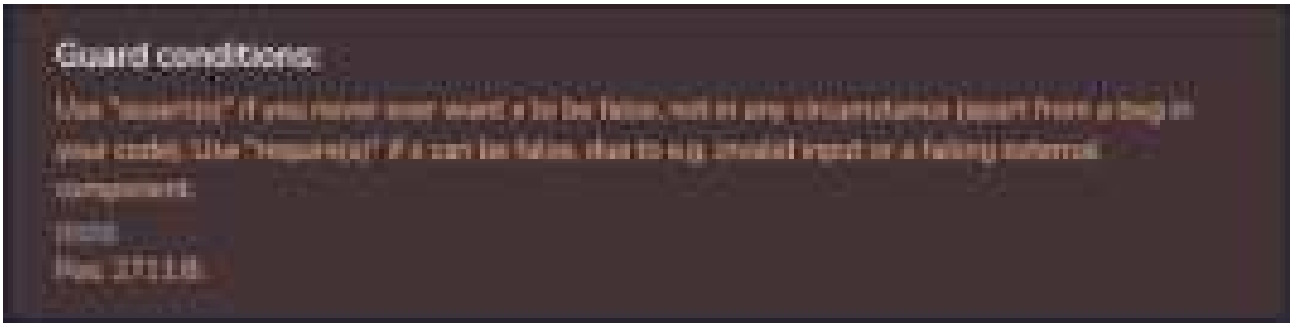
more

Pos: 2710:8

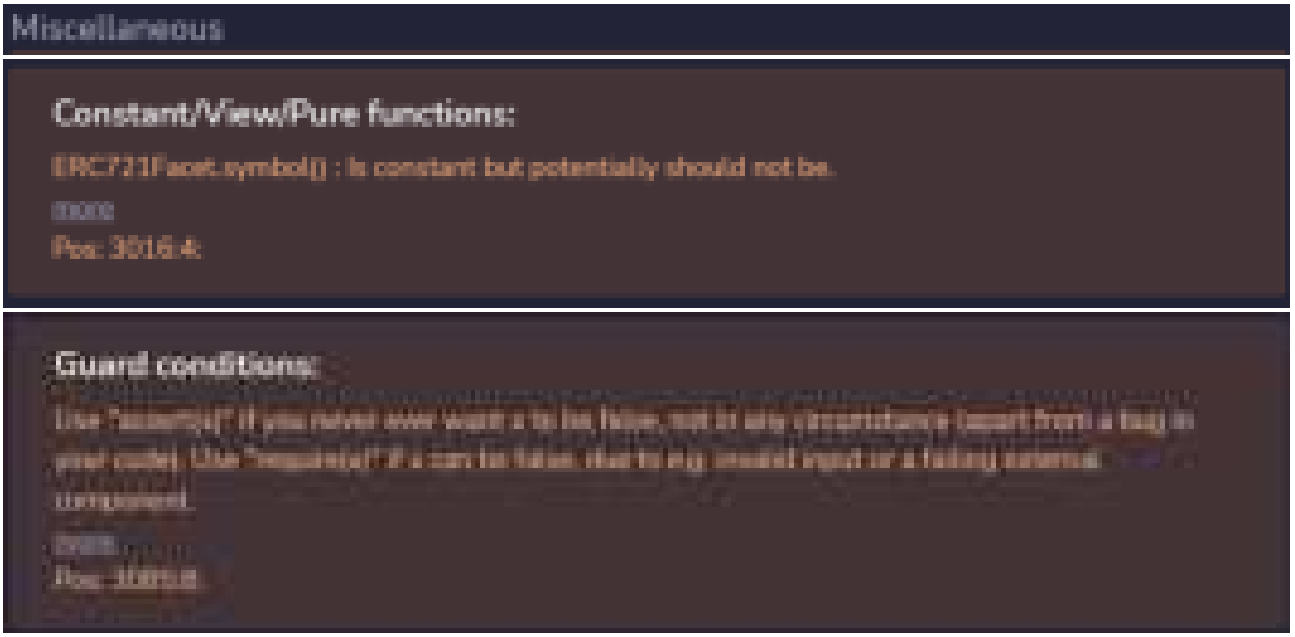
Similar variable names:

ERC721._transfer(address,address,uint256) : Variables have very similar names "to" and "to_".

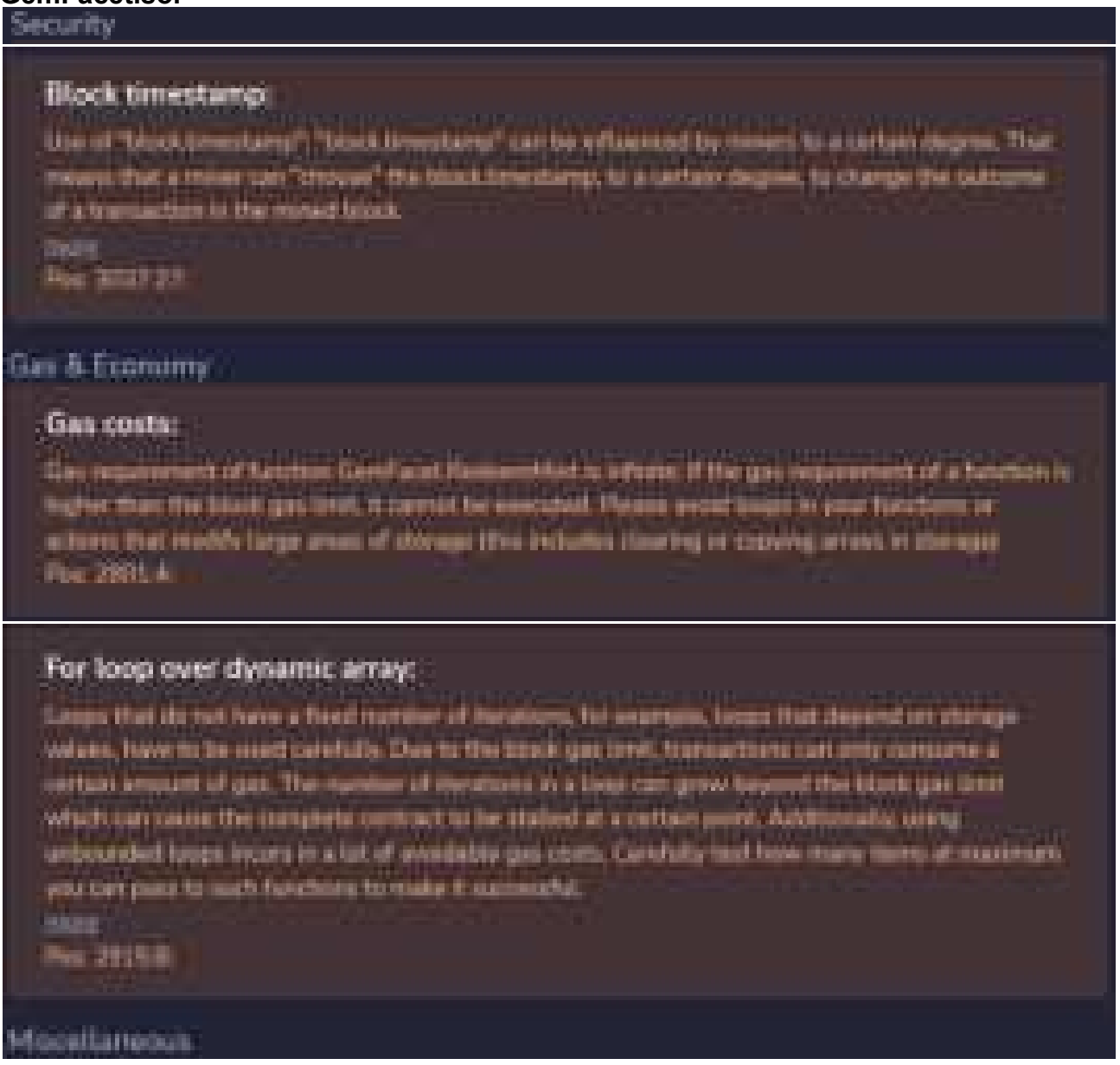
Pos: 3024:34



ERC721Facet.sol



GemFacet.sol



Constant/View/Pure functions:

Similar variable names:

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.

Data truncated:

GemGettersFacet.sol

Security

Inline assembly:

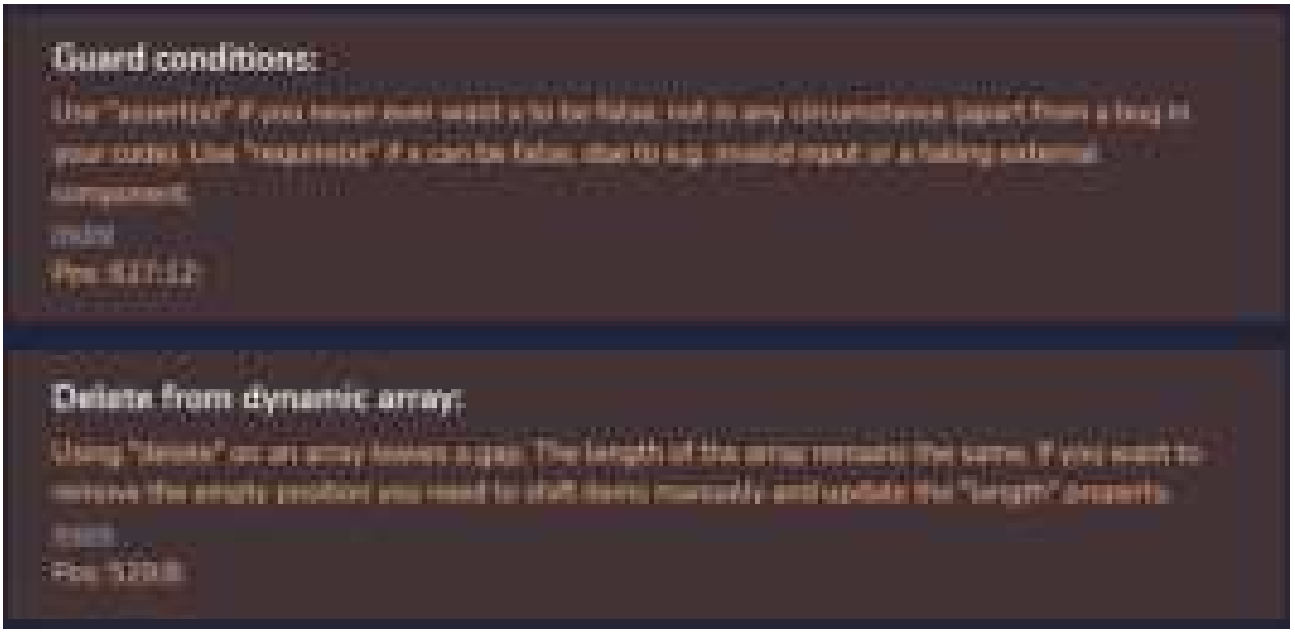
Gas & Economy

Gas costs:

Miscellaneous

Constant/View/Pure functions:

NodeLimiterFacet.sol



OwnerFacet.sol

OwnershipFacet.sol

VaultStakingFacet.sol

Gas & Economy

Gas costs:

Gas requirement of function VaultStaking.forceBatchAddToken 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: 234:4

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: 235:8

Miscellaneous

Constant/View/Pure functions:

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

more

Pos: 240:4

Similar variable names:

VaultStaking.withdrawStakeTokens(uint256,uint256) : Variables have very similar names "unlockTime" and "user". Note: Modifiers are currently not considered by this static analysis.

Pos: 241:8

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: 242:8

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: 243:24

Solhint Linter

DefoErc20.sol

```
DefoErc20.sol:2:1: Error: Compiler version ^0.8.0 does not satisfy
the r semver requirement
DefoErc20.sol:59:28: Error: Avoid using low level calls.
DefoErc20.sol:133:51: Error: Avoid using low level calls.
DefoErc20.sol:187:51: Error: Avoid using low level calls.
DefoErc20.sol:209:17: Error: Avoid using inline assembly. It is
acceptable only in rare cases
DefoErc20.sol:221:1: Error: Contract name must be in CamelCase
DefoErc20.sol:222:2: Error: Explicitly mark visibility of state
DefoErc20.sol:227:3: Error: Avoid using inline assembly. It is
acceptable only in rare cases
DefoErc20.sol:229:8: Error: Variable "r" is unused
DefoErc20.sol:2179:24: Error: Code contains empty blocks
DefoErc20.sol:2199:24: Error: Code contains empty blocks
DefoErc20.sol:2236:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
DefoErc20.sol:2374:5: Error: Explicitly mark visibility of state
DefoErc20.sol:2374:13: Error: Variable name must be in mixedCase
DefoErc20.sol:2377:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
```

DefoLimiter.sol

```
DefoLimiter.sol:889:33: Error: Parse error: mismatched input '('
expecting {';', '='}
DefoLimiter.sol:889:54: Error: Parse error: extraneous input ')'
expecting {';', '='}
DefoLimiter.sol:984:46: Error: Parse error: mismatched input '('
expecting {';', '='}
```

Diamond.sol

```
Diamond.sol:2:1: Error: Compiler version ^0.8.0 does not satisfy the
r semver requirement
Diamond.sol:42:5: Error: Explicitly mark visibility of state
Diamond.sol:76:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
Diamond.sol:358:50: Error: Avoid using low level calls.
Diamond.sol:375:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
Diamond.sol:384:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Diamond.sol:401:5: Error: Fallback function must be simple
Diamond.sol:405:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
```

```
Diamond.sol:412:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
Diamond.sol:430:32: Error: Code contains empty blocks
```

DiamondInit.sol

```
DiamondInit.sol:2:1: Error: Compiler version ^0.8.4 does not satisfy
the r semver requirement
DiamondInit.sol:114:5: Error: Explicitly mark visibility of state
DiamondInit.sol:148:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
DiamondInit.sol:430:50: Error: Avoid using low level calls.
DiamondInit.sol:447:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
```

LpManager.sol

```
LpManager.sol:2418:18: Error: Parse error: missing ';' at '{'
LpManager.sol:2425:18: Error: Parse error: missing ';' at '{'
LpManager.sol:2447:18: Error: Parse error: missing ';' at '{'
```

Redeem.sol

```
Redeem.sol:12:18: Error: Parse error: missing ';' at '{'
Redeem.sol:25:18: Error: Parse error: missing ';' at '{'
Redeem.sol:37:18: Error: Parse error: missing ';' at '{'
Redeem.sol:54:18: Error: Parse error: missing ';' at '{'
Redeem.sol:66:18: Error: Parse error: missing ';' at '{'
Redeem.sol:162:18: Error: Parse error: missing ';' at '{'
Redeem.sol:185:18: Error: Parse error: missing ';' at '{'
Redeem.sol:211:18: Error: Parse error: missing ';' at '{'
```

DiamondCutFacet.sol

```
DiamondCutFacet.sol:2:1: Error: Compiler version ^0.8.4 does not
satisfy the r semver requirement
DiamondCutFacet.sol:39:5: Error: Explicitly mark visibility of state
DiamondCutFacet.sol:73:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
DiamondCutFacet.sol:355:50: Error: Avoid using low level calls.
DiamondCutFacet.sol:372:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
```

DiamondLoupeFacet.sol

```
DiamondLoupeFacet.sol:2:1: Error: Compiler version ^0.8.4 does not
satisfy the r semver requirement
DiamondLoupeFacet.sol:87:5: Error: Explicitly mark visibility of
state
DiamondLoupeFacet.sol:121:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
DiamondLoupeFacet.sol:403:50: Error: Avoid using low level calls.
DiamondLoupeFacet.sol:420:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
```

ERC721Enumerable.sol

```
ERC721Enumerable.sol:3:1: Error: Compiler version ^0.8.4 does not
satisfy the r semver requirement
ERC721Enumerable.sol:191:1: Error: Contract name must be in CamelCase
ERC721Enumerable.sol:192:2: Error: Explicitly mark visibility of
state
ERC721Enumerable.sol:197:3: Error: Avoid using inline assembly. It is
acceptable only in rare cases
ERC721Enumerable.sol:199:8: Error: Variable "r" is unused
ERC721Enumerable.sol:1745:9: Error: Variable name must be in
mixedCase
ERC721Enumerable.sol:1762:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
ERC721Enumerable.sol:1802:13: Error: Avoid using inline assembly. It
is acceptable only in rare cases
ERC721Enumerable.sol:1822:9: Error: Avoid using inline assembly. It
is acceptable only in rare cases
ERC721Enumerable.sol:2106:16: Error: Code contains empty blocks
ERC721Enumerable.sol:2139:9: Error: Avoid using inline assembly. It
is acc
```

ERC721Facet.sol

```
ERC721Facet.sol:2583:18: Error: Parse error: missing ';' at '{'
ERC721Facet.sol:2591:18: Error: Parse error: missing ';' at '{'
```

GemFacet.sol

```
GemFacet.sol:2398:18: Error: Parse error: missing ';' at '{'
GemFacet.sol:2406:18: Error: Parse error: missing ';' at '{'
```

GemGettersFacet.sol

```
GemGettersFacet.sol:257:18: Error: Parse error: missing ';' at '{'
GemGettersFacet.sol:265:18: Error: Parse error: missing ';' at '{'
```

NodeLimiterFacet.sol

```
NodeLimiterFacet.sol:115:18: Error: Parse error: missing ';' at '{'
NodeLimiterFacet.sol:123:18: Error: Parse error: missing ';' at '{'
```

OwnerFacet.sol

```
OwnerFacet.sol:230:18: Error: Parse error: missing ';' at '{'
OwnerFacet.sol:238:18: Error: Parse error: missing ';' at '{'
```

OwnershipFacet.sol

```
OwnershipFacet.sol:2:1: Error: Compiler version ^0.8.4 does not
satisfy the r semver requirement
OwnershipFacet.sol:54:5: Error: Explicitly mark visibility of state
OwnershipFacet.sol:88:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
OwnershipFacet.sol:370:50: Error: Avoid using low level calls.
OwnershipFacet.sol:387:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
```

VaultStakingFacet.sol

```
VaultStakingFacet.sol:2054:18: Error: Parse error: missing ';' at '{'
VaultStakingFacet.sol:2062:18: 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