



Full Audit Report

EthShares Security Assessment

Real Cybersecurity
Protecting digital assets



Made in Thailand

SECURI LAB
(THAILAND) contact@securi-lab.com



FULL AUDIT REPORT

Table of Contents

	1
▪ Report Information	2
▪ Disclaimer	3
▪ Executive Summary	4
NVD CVSS Scoring	
Audit Result	
▪ Project Introduction	5
Scope Information	
Audit Information	
Audit Version History	
▪ Initial Audit Scope	6-7
▪ Security Assessment Procedure	8
▪ Risk Rating	9
▪ Vulnerability Severity Summary	10
▪ Vulnerability Findings	11-27
SWC & SEC & Non-severity level	
▪ SWC Findings	28-30
▪ Visibility, Mutability, Modifier function testing	31-38
Component, Exposed Function	
StateVariables, Capabilities, Contract Descripton Table	
▪ Inheritate Function Relation Graph	39
▪ UML Diagram	40
▪ About Securi	41

FULL AUDIT REPORT

Report Information

About Report	EthShares Security Assessment
Version	v1.0
Client	EthShares
Language	Solidity
Confidentiality	Public
Contract Address	0x5f12D4012185e044B5FE1B3dBD9B8B1e7Ffb27f https://bscscan.com/token/0x5f12D4012185e044B5FE1B3dBD9B8B1e7Ffb27f#code
Audit Method	Whitebox
Security Assessment Author	Auditor  Mark K. [Security Researcher Redteam] Kevin N. [Security Researcher Web3 Dev] Yusheng T. [Security Researcher Incident Response] Approve Document Ronny C. CTO & Head of Security Researcher Chinnakit J. CEO & Founder

*Audit Method

Whitebox: SECURI LAB Team receives all source code from the client to provide the assessment.
Blackbox: SECURI LAB Team receives only bytecode from the client to provide the assessment.

Digital Sign (Only Full Audit Report)

FULL AUDIT REPORT

Disclaimer

Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as **"Source code"**.

And **SECURI Lab** hereinafter referred to as **"Service Provider"**, the **Service Provider** will not be held liable for any legal liability arising from errors in the security assessment. The responsibility will be the responsibility of the **Client**, hereinafter referred to as **"Service User"** and the **Service User** agrees not to be held liable to the **service provider** in any case. By contract **Service Provider** to conduct security assessments with integrity with professional ethics, and transparency to deliver security assessments to users The **Service Provider** has the right to postpone the delivery of the security assessment. If the security assessment is delayed whether caused by any reason and is not responsible for any delayed security assessments.

If the **service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security assessment. Or happened later All responsibility shall be sole with the **service user**.

Security Assessment Not Financial/Investment Advice Any loss arising from any investment in any project is the responsibility of the investor.

SECURI LAB disclaims any liability incurred. Whether it's Rugpull, Abandonment, Soft Rugpull

The SECURI LAB team has conducted a comprehensive security assessment of the vulnerabilities. This assessment is tested with an expert assessment. Using the following test requirements

1. Smart Contract Testing with Expert Analysis By testing the most common and uncommon vulnerabilities.
2. Automated program testing It includes a sample vulnerability test and a sample of the potential vulnerabilities being used for the most frequent attacks.
3. Manual Testing with AST/WAS/ASE/SMT and reviewed code line by line
4. Visibility, Mutability, Modifier function testing, such as whether a function can be seen in general, or whether a function can be changed and if so, who can change it.
5. Function association test It will be displayed through the association graph.
6. This safety assessment is cross-checked prior to the delivery of the assessment results.

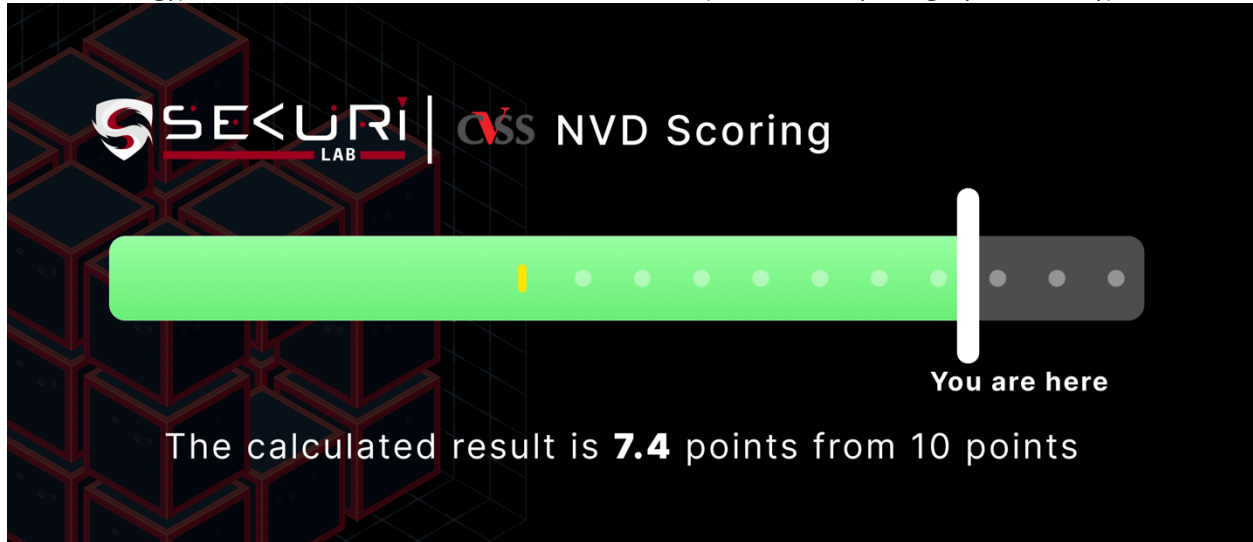
FULL AUDIT REPORT

Executive Summary

For this security assessment, SECURI LAB received a request from **EthShares** on Wednesday, April 26, 2023.

NVD CVSS Scoring

The score was calculated using the NVD (National Vulnerability Database) of NIST (National Institute of Standards and Technology) under the CVSS 3.1 standard, based on the CIA (Confidentiality, Integrity, Availability).



Audit Result

SECURI LAB evaluated the smart contract security of the project and found: **[Total : 7]**

Critical	High	Medium	Low	Very Low	Informational
0	2	0	3	0	2



SECURI LAB has assessed the security of this smart contract.

The results of the security assessment revealed

No Critical Vulnerabilities.

Full Audit Report by SECURI LAB on Apr 29, 2023



FULL AUDIT REPORT

Project Introduction

Scope Information:

Project Name	EthShares
Website	https://ethshares.io/
Chain	BNB Chain (Previously name Binance Smart Chain)
Language	Solidity

Audit Information:

Request Date	Wednesday, April 26, 2023
Audit Date	Wednesday, April 26, 2023
Re-assessment Date	-

Audit Version History:

Version	Date	Description
1.0	Thursday, April 27, 2023	Preliminary Report
1.1	Saturday, April 29, 2023	Full Audit Report

FULL AUDIT REPORT

Initial Audit Scope:

Smart Contract	0x5f12D4012185e044B5FE1B3dBD9B8B1e7Ffb27f https://bscscan.com/token/0x5f12D4012185e044B5FE1B3dBD9B8B1e7Ffb27f#code
Compiler Version	^0.8.0

Source Units Analyzed: 1

Source Units in Scope: 1 (100%)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/ETS.sol	4	4	790	719	544	34	507	
	Totals	4	4	790	719	544	34	507	

Legend: []

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines:** lines containing single or block comments
- **Complexity Score:** a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

FULL AUDIT REPORT

Security Assessment Procedure

Securi has the following procedures and regulations for conducting security assessments:

1.Request Audit Client submits a form request through the Securi channel. After receiving the request, Securi will discuss a security assessment. And drafting a contract and agreeing to sign a contract together with the Client

2.Auditing Securi performs security assessments of smart contracts obtained through automated analysis and expert manual audits.

3.Preliminary Report At this stage, Securi will deliver an initial security assessment. To report on vulnerabilities and errors found under Audit Scope will not publish preliminary reports for safety.

4.Reassessment After Securi has delivered the Preliminary Report to the Client, Securi will track the status of the vulnerability or error, which will be published to the Final Report at a later date with the following statuses:

a.Acknowledge The client has been informed about errors or vulnerabilities from the security assessment.

b.Resolved The client has resolved the error or vulnerability. Resolved is probably just a commit, and Securi is unable to verify that the resolved has been implemented or not.

c.Decline Client has rejected the results of the security assessment on the issue.

5.Final Report Securi providing full security assessment report and public



FULL AUDIT REPORT

Risk Rating

Risk rating using this commonly defined: $Risk\ rating = impact * confidence$

Impact The severity and potential impact of an attacker attack

Confidence Ensuring that attackers expose and use this vulnerability

Both have a total of 3 levels: **High, Medium, Low**. By *Informational* will not be classified as a level

Confidence Impact [Likelihood]	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical



FULL AUDIT REPORT

Vulnerability Severity Summary

Severity is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods,

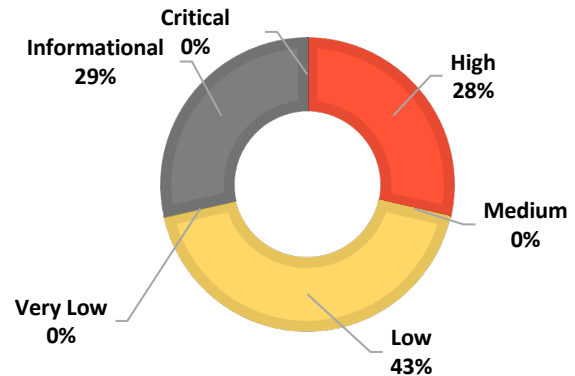
$$\text{Risk rating} = \text{impact} * \text{confidence}$$

It is categorized into

5 categories based on the lowest severity:

Very Low, Low, Medium, High, Critical.

For **Informational** & will **Non-class/Optimization/Best-practices** will not be counted as severity



Vulnerability Severity Level	Total
Critical	0
High	2
Medium	0
Low	3
Very Low	0
Informational	2
Non-class/Optimization/Best-practices	6

Category information:

Centralization Centralization Risk is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission	Economics Risk Economics Risk is Risks that may affect the economic mechanism system, such as the ability to increase Mint token	Logical Issue Logical Issue is that can cause errors to core processing, such as any prior operations that cause background processes to crash.	Authorization Authorization is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.	Mathematical Mathematical Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.	Naming Conventions Naming Conventions naming variables that may affect code understanding or naming inconsistencies
Security Risk Security Risk of loss or damage if it's no mitigate	Coding Style Coding Style is Tips coding for efficiency performance	Best Practices Best Practices is suggestions for improvement	Optimization Optimization is performance improvement	Gas Optimization Gas Optimization is increase performance to avoid expensive gas	Dead Code Dead Code having unused code This may result in wasted resources and gas fees.

FULL AUDIT REPORT

Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
SEC-01	Unchecked tokens transfer (unchecked-transfer)	High	Logical Issue	Acknowledge
SEC-02	Centralization Risk	High	Centralization	Acknowledge
SEC-03	Avoid using block timestamp	Low	Best Practices	Acknowledge
SEC-04	Empty Function Body - Consider commenting why	Low	Coding Style	Acknowledge
SEC-05	Multiple calls in a loop (calls-loop)	Low	Best Practices	Acknowledge
SEC-06	unlocked-compiler-version	Informational	Best Practices	Acknowledge
SEC-07	inconsistent-comments-and-code	Informational	Coding Style	Acknowledge
NC-01	Functions not used internally could be marked external	-	Gas Optimization	Acknowledge
GAS-01	Use `selfbalance()` instead of `address(this).balance`	-	Gas Optimization	Acknowledge
GAS-02	Use assembly to check for `address(0)`	-	Gas Optimization	Acknowledge
GAS-03	Using bools for storage incurs overhead	-	Gas Optimization	Acknowledge
GAS-04	Use Custom Errors	-	Gas Optimization	Acknowledge
GAS-05	Use != 0 instead of > 0 for unsigned integer comparison	-	Gas Optimization	Acknowledge

FULL AUDIT REPORT

SEC-01: Unchecked tokens transfer (unchecked-transfer)

Vulnerability Detail	Severity	Location	Category	Status
Unchecked tokens transfer (unchecked-transfer)	High	Check on finding	Logical Issue	Acknowledge

Finding:

`DividendDistributor.distributeDividend(address)` (ETS.sol:354-365) ignores return value by `ETH.transfer(shareholder,amount)` (ETS.sol#360)

Recommendation:

Use `SafeERC20`, or ensure that the transfer/transferFrom return value is checked.

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer>

Exploit Scenario:

there's a potential risk associated with the `ETH.transfer(shareholder, amount)` line. This line transfers Ether without checking for the success of the transfer operation. If the transfer fails, the contract will continue executing, potentially leading to unexpected behavior.

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

SEC-02: Centralization Risk

Vulnerability Detail	Severity	Location	Category	Status
Centralization Risk	High	Check on finding	Centralization	Acknowledge

Finding:

```

109: abstract contract Auth {
135:     function authorize(address adr) public onlyOwner {
142:     function unauthorize(address adr) public onlyOwner {
163:     function transferOwnership(address payable adr) public onlyOwner {
398: contract ETS is IBEP20, Auth {
468:     ) Auth(msg.sender) {
...

```

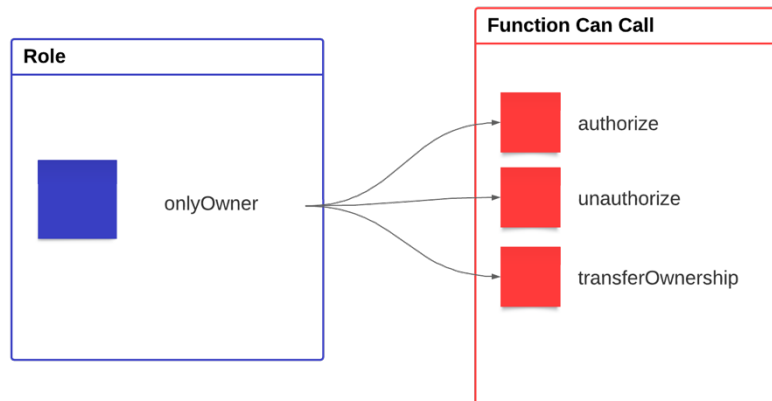
Scenario:

Centralized risk refers to the potential security risks that arise when a smart contract is controlled by a central entity or a single point of failure. If the contract is controlled by a central authority, then the contract may be vulnerable to attacks that target the centralized entity.

Centralized risk that can lead to rug pulls typically arises from the centralization of control or ownership of a project's assets, particularly in decentralized finance (DeFi) projects built on blockchain platforms like Ethereum.

FULL AUDIT REPORT

Contract Auth (File: ETS.sol)

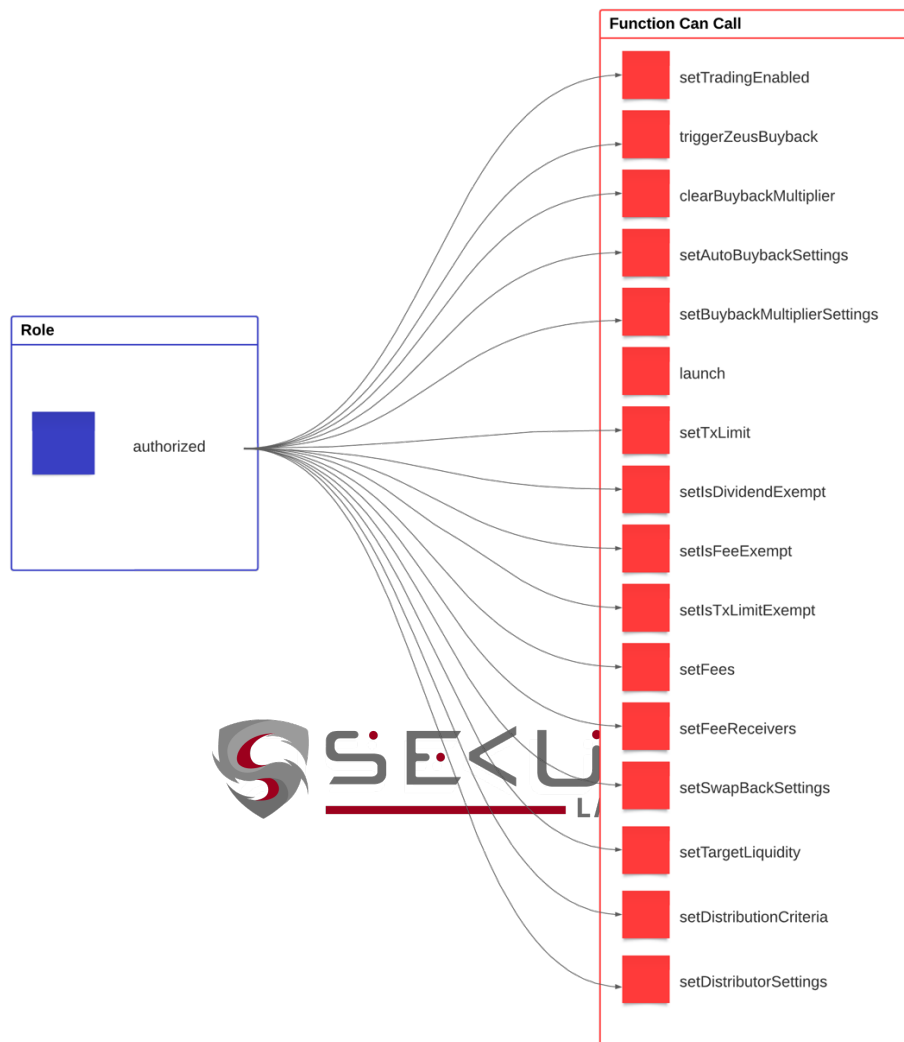


In the Auth (File: ETS.sol) contract, Owner can call functions **authorize**, **unauthorize**, **transferOwnership**. We've found that some functions work in an anti-whale manner and allow the owner to pause trading. Assigning a blacklist address and also another function we recommend that for transparency use Timelock to increase the delay for users. Function calls are visible before they are fully executed. Additionally, the implementation of a multi-signature feature adds another layer of security to safeguard the owner's account.



FULL AUDIT REPORT

Contract ETS (File: ETS.sol)



In the ETC (File: ETS.sol) contract can external call, for authorized role can call functions **setTradingEnabled**, **triggerZeusBuyback**, **clearBuybackMultiplier**, **setAutoBuybackSettings**, **setBuybackMultiplierSettings**, **launch**, **setTxLimit**, **setIsDividendExempt**, **setIsFeeExempt**, **setIsTxLimitExempt**, **setFees**, **setFeeReceivers**, **setSwapBackSettings**, **setTargetLiquidity**, **setDistributionCriteria**, **setDistributorSettings**. We've found that some functions work in an anti-whale manner and allow the owner to pause trading. Assigning a blacklist address and also another function we recommend that for transparency use Timelock to increase the delay for users. Function calls are visible before they are fully executed. Additionally, the implementation of a multi-signature feature adds another layer of security to safeguard the owner's account.

FULL AUDIT REPORT

Recommendation:

In terms of timeframes, there are three categories: short-term, long-term, and permanent.

For short-term solutions, a combination of timelock and multi-signature (2/3 or 3/5) can be used to mitigate risk by delaying sensitive operations and avoiding a single point of failure in key management. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; assigning privileged roles to multi-signature wallets to prevent private key compromise; and sharing the timelock contract and multi-signer addresses with the public via a medium/blog link.

For long-term solutions, a combination of timelock and DAO can be used to apply decentralization and transparency to the system. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; introducing a DAO/governance/voting module to increase transparency and user involvement; and sharing the timelock contract, multi-signer addresses, and DAO information with the public via a medium/blog link.

Finally, permanent solutions should be implemented to ensure the ongoing security and protection of the system.

Alleviation:

EthShares team has Acknowledge this issue.



FULL AUDIT REPORT

SEC-03: Avoid using block timestamp

Vulnerability Detail	Severity	Location	Category	Status
Avoid using block timestamp	Low	Check on finding	Best Practices	Acknowledge

Finding:

```

314:         block.timestamp

350:         return shareholderClaims[shareholder] + minPeriod < block.timestamp

361:         shareholderClaims[shareholder] = block.timestamp;

583:         if (launchedAtTimestamp + 1 days > block.timestamp) {

585:         } else if (buybackMultiplierTriggeredAt.add(buybackMultiplierLength) >
block.timestamp) {

586:         uint256 remainingTime =
buybackMultiplierTriggeredAt.add(buybackMultiplierLength).sub(block.timestamp);

624:         block.timestamp

647:         block.timestamp

664:         buybackMultiplierTriggeredAt = block.timestamp;

689:         block.timestamp

716:         launchedAtTimestamp = block.timestamp;

```

Recommendation:

Using block timestamp in smart contracts can lead to security vulnerabilities and should be avoided.

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp>

Exploit Scenario:

-

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

SEC-04: Empty Function Body - Consider commenting why

Vulnerability Detail	Severity	Location	Category	Status
Empty Function Body - Consider commenting why	Low	Check on finding	Coding Style	Acknowledge

Finding:

```
493:     receive() external payable { }

550:         if(!isDividendExempt[sender]){ try distributor.setShare(sender,
_balances[sender]) {} catch {} }

550:         if(!isDividendExempt[sender]){ try distributor.setShare(sender,
_balances[sender]) {} catch {} }

551:         if(!isDividendExempt[recipient]){ try distributor.setShare(recipient,
_balances[recipient]) {} catch {} }

551:         if(!isDividendExempt[recipient]){ try distributor.setShare(recipient,
_balances[recipient]) {} catch {} }

553:         try distributor.process(distributorGas) {} catch {}

553:         try distributor.process(distributorGas) {} catch {}

635:         try distributor.deposit{value: amountBNBReflection}() {} catch {}

635:         try distributor.deposit{value: amountBNBReflection}() {} catch {}
```

Recommendation:

While this line of code might work correctly in most cases, it is considered potentially unsafe because it assumes the ERC20 token contract implements the transfer function correctly and consistently with the ERC20 standard.

A safer way to interact with ERC20 tokens is to use a widely-adopted and audited library, such as OpenZeppelin's ERC20 library. This helps you avoid potential issues with non-standard or malicious token implementations.

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

SEC-05: Multiple calls in a loop (calls-loop)

Vulnerability Detail	Severity	Location	Category	Status
Multiple calls in a loop (calls-loop)	Low	Check on finding	Best Practices	Acknowledge

Finding:

```
354:     function distributeDividend(address shareholder) internal {
        if(shares[shareholder].amount == 0){ return; }

        uint256 amount = getUnpaidEarnings(shareholder);
        if(amount > 0){
            totalDistributed = totalDistributed.add(amount);
            ETH.transfer(shareholder, amount);
            shareholderClaims[shareholder] = block.timestamp;
            shares[shareholder].totalRealised =
shares[shareholder].totalRealised.add(amount);
            shares[shareholder].totalExcluded =
getCumulativeDividends(shares[shareholder].amount);
        }
365:     }
```

Recommendation:

distributeDividend(address shareholder) function, which involves making an external call using ETH.transfer(shareholder, amount). Although the provided code snippet does not contain a loop, the warning suggests that you should favor the "pull over push" strategy for external calls, which can help reduce the risk of reentrancy attacks and make your contract more gas-efficient in certain situations.

Favor [pull over push](<https://github.com/ethereum/wiki/wiki/Safety#favor-pull-over-push-for-external-calls>) strategy for external calls.

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop>

Exploit Scenario:

-

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

SEC-06: unlocked-compiler-version

Vulnerability Detail	Severity	Location	Category	Status
unlocked-compiler-version	Informational	Check on finding	Best Practices	Acknowledge

Finding:

```
2: pragma solidity ^0.8.0;
```

Recommendation:

Unlocked pragma disables all source code analysis, making it vulnerable to attacks

Exploit Scenario:

-

Alleviation:

EthShares team has Acknowledge this issue.



FULL AUDIT REPORT

SEC-07: inconsistent-comments-and-code

Vulnerability Detail	Severity	Location	Category	Status
inconsistent-comments-and-code	Informational	Check on finding	Coding Style	Acknowledge

Finding:

538: //

Recommendation:

Inconsistent comments and code can lead to confusion and misunderstandings when maintaining and auditing a smart contract. It's crucial to keep comments up-to-date and in sync with the actual code implementation

Exploit Scenario:

-

Alleviation:

EthShares team has Acknowledge this issue.



FULL AUDIT REPORT

NC-01: Functions not used internally could be marked external

Vulnerability Detail	Severity	Location	Category	Status
Functions not used internally could be marked external	-	Check on finding	Best Practices	Acknowledge

Finding:

```

135:    function authorize(address adr) public onlyOwner {
142:    function unauthorize(address adr) public onlyOwner {
163:    function transferOwnership(address payable adr) public onlyOwner {
713:    function launch() public authorized {

```

Recommendation:

is marked **public**, which means it can be called both internally (from within the contract) and externally (from outside the contract). However, if this function is not intended to be used internally, it can be marked **external** to optimize gas usage and restrict its usage to external calls only.



Exploit Scenario:

-

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

GAS-01: Use `selfbalance()` instead of `address(this).balance`

Vulnerability Detail	Severity	Location	Category	Status
Use `selfbalance()` instead of `address(this).balance`	-	Check on finding	Gas Optimization	Acknowledge

Finding:

```

617:         uint256 balanceBefore = address(this).balance;
627:         uint256 amountBNB = address(this).balance.sub(balanceBefore);
658:         && address(this).balance >= autoBuybackAmount;

```

Recommendation:

Use assembly when getting a contract's balance of ETH.

You can use `selfbalance()` instead of `address(this).balance` when getting your contract's balance of ETH to save gas. Additionally, you can use `balance(address)` instead of `address.balance()` when getting an external contract's balance of ETH.

Saves 15 gas when checking internal balance, 6 for external

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

GAS-02: Use assembly to check for `address(0)

Vulnerability Detail	Severity	Location	Category	Status
Use assembly to check for `address(0)	-	Check on finding	Gas Optimization	Acknowledge

Finding:

```
276:      router = _router != address(0)
```

Recommendation:

Saves 6 gas per instance

Instances (2):

Alleviation:

EthShares team has Acknowledge this issue.



FULL AUDIT REPORT

GAS-03: Using bools for storage incurs overhead

Vulnerability Detail	Severity	Location	Category	Status
Using bools for storage incurs overhead	-	Check on finding	Gas Optimization	Acknowledge

Finding:

```

111:    mapping (address => bool) internal authorizations;

418:    mapping (address => bool) isFeeExempt;

419:    mapping (address => bool) isTxLimitExempt;

420:    mapping (address => bool) isDividendExempt;

449:    bool public autoBuybackEnabled = false;

450:    mapping (address => bool) buyBacker;

462:    bool public swapEnabled = true;

```

Recommendation:

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past. See [source] (<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27>).

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

GAS-04: Use Custom Errors

Vulnerability Detail	Severity	Location	Category	Status
Use Custom Errors	-	Check on finding	Gas Optimization	Acknowledge

Finding:

```
122:         require(isOwner(msg.sender), "!OWNER"); _;  
129:         require(isAuthorized(msg.sender), "!AUTHORIZED"); _;  
500:     modifier onlyBuybacker() { require(buyBacker[msg.sender] == true, ""); _; }  
535:         require(tradingEnabled, "Trading disabled");  
569:         require(amount <= _maxTxAmount || isTxLimitExempt[sender], "TX Limit  
Exceeded");  
714:         require(launchedAt == 0, "Already launched boi");
```

Recommendation:

[Source](<https://blog.soliditylang.org/2021/04/21/custom-errors/>)

Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

GAS-05: Use != 0 instead of > 0 for unsigned integer comparison

Vulnerability Detail	Severity	Location	Category	Status
Use != 0 instead of > 0 for unsigned integer comparison	-	Check on finding	Gas Optimization	Acknowledge

Finding:

```

81:         require(b > 0, errorMessage);
88:         require(b > 0, errorMessage);
288:         if(shares[shareholder].amount > 0){
292:         if(amount > 0 && shares[shareholder].amount == 0){
294:         }else if(amount == 0 && shares[shareholder].amount > 0){
358:         if(amount > 0){
640:         if(amountToLiquify > 0){

```

Recommendation:

[Source](<https://blog.soliditylang.org/2021/04/21/custom-errors/>)

Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

Alleviation:

EthShares team has Acknowledge this issue.

FULL AUDIT REPORT

SWC Findings

ID	Title	Scanning	Result
SWC-100	Function Default Visibility	Complete	No risk
SWC-101	Integer Overflow and Underflow	Complete	No risk
SWC-102	Outdated Compiler Version	Complete	No risk
SWC-103	Floating Pragma	Complete	No risk
SWC-104	Unchecked Call Return Value	Complete	No risk
SWC-105	Unprotected Ether Withdrawal	Complete	No risk
SWC-106	Unprotected SELFDESTRUCT Instruction	Complete	No risk
SWC-107	Reentrancy	Complete	No risk
SWC-108	State Variable Default Visibility	Complete	No risk
SWC-109	Uninitialized Storage Pointer	Complete	No risk
SWC-110	Assert Violation	Complete	No risk
SWC-111	Use of Deprecated Solidity Functions	Complete	No risk
SWC-112	Delegatecall to Untrusted Callee	Complete	No risk
SWC-113	DoS with Failed Call	Complete	No risk
SWC-114	Transaction Order Dependence	Complete	No risk
SWC-115	Authorization through tx.origin	Complete	No risk

FULL AUDIT REPORT

SWC-116	Block values as a proxy for time	Complete	No risk
SWC-117	Signature Malleability	Complete	No risk
SWC-118	Incorrect Constructor Name	Complete	No risk
SWC-119	Shadowing State Variables	Complete	No risk
SWC-120	Weak Sources of Randomness from Chain Attributes	Complete	No risk
SWC-121	Missing Protection against Signature Replay Attacks	Complete	No risk
SWC-122	Lack of Proper Signature Verification	Complete	No risk
SWC-123	Requirement Violation	Complete	No risk
SWC-124	Write to Arbitrary Storage Location	Complete	No risk
SWC-125	Incorrect Inheritance Order	Complete	No risk
SWC-126	Insufficient Gas Griefing	Complete	No risk
SWC-127	Arbitrary Jump with Function Type Variable	Complete	No risk
SWC-128	DoS With Block Gas Limit	Complete	No risk
SWC-129	Typographical Error	Complete	No risk
SWC-130	Right-To-Left-Override control character (U+202E)	Complete	No risk
SWC-131	Presence of unused variables	Complete	No risk
SWC-132	Unexpected Ether balance	Complete	No risk

FULL AUDIT REPORT




SWC-133	Hash Collisions With Multiple Variable Length Arguments	Complete	No risk
SWC-134	Message call with hardcoded gas amount	Complete	No risk
SWC-135	Code With No Effects	Complete	No risk
SWC-136	Unencrypted Private Data On-Chain	Complete	No risk



FULL AUDIT REPORT



Visibility, Mutability, Modifier function testing

Components


 Contracts	 Libraries	 Interfaces	 Abstract
2	1	4	1

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.











 Public	 Payable			
66	5			
External	Internal	Private	Pure	View
52	78	0	18	26

StateVariables

Total	 Public
68	21



Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
<input type="text" value="^0.8.0"/>		<input type="text" value="yes"/>	<input type="text"/>	<input type="text"/>	
 Transfers ETH	 Low-Level Calls	 Delegate Call	 Uses Hash Functions	 ECREcover	 New/Create/Create2
<input type="text" value="yes"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="yes"/> <input type="text" value="→ NewContract:Divide"/> <input type="text" value="ndDistributor"/>



FULL AUDIT REPORT

 TryCatch	Σ Unchecked
yes	yes













FULL AUDIT REPORT

Contracts Description Table

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
L	tryAdd	Internal 🔒		
L	trySub	Internal 🔒		
L	tryMul	Internal 🔒		
L	tryDiv	Internal 🔒		
L	tryMod	Internal 🔒		
L	add	Internal 🔒		
L	sub	Internal 🔒		
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
L	sub	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
IBEP20	Interface			
L	totalSupply	External !		NO !
L	decimals	External !		NO !
L	symbol	External !		NO !
L	name	External !		NO !
L	getOwner	External !		NO !

FULL AUDIT REPORT

Contract	Type	Bases		
L	balanceOf	External !		NO !
L	transfer	External !		NO !
L	allowance	External !		NO !
L	approve	External !		NO !
L	transferFrom	External !		NO !
Auth	Implementation			
L		Public !		NO !
L	authorize	Public !		onlyOwner
L	unauthorize	Public !		onlyOwner
L	isOwner	Public !		NO !
L	isAuthorized	Public !		NO !
L	transferOwnership	Public !		onlyOwner
IDEXFactory	Interface			
L	createPair	External !		NO !
IDEXRouter	Interface			
L	factory	External !		NO !
L	WETH	External !		NO !
L	addLiquidity	External !		NO !
L	addLiquidityETH	External !		NO !

FULL AUDIT REPORT

Contract	Type	Bases		
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External !		NO !
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External !		NO !
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External !		NO !
IDividendDistributor	Interface			
L	setDistributionCriteria	External !		NO !
L	setShare	External !		NO !
L	deposit	External !		NO !
L	process	External !		NO !
DividendDistributor	Implementation	IDividendDistributor		
L		Public !		NO !
L	setDistributionCriteria	External !		onlyToken
L	setShare	External !		onlyToken
L	deposit	External !		onlyToken
L	process	External !		onlyToken
L	shouldDistribute	Internal		
L	distributeDividend	Internal		
L	claimDividend	External !		NO !
L	getUnpaidEarnings	Public !		NO !







FULL AUDIT REPORT

Contract	Type	Bases		
L	getCumulativeDividends	Internal		
L	addShareholder	Internal		
L	removeShareholder	Internal		
ETS	Implementation	IBEP20, Auth		
L		Public !		Auth
L		External !		NO !
L	totalSupply	External !		NO !
L	decimals	External !		NO !
L	symbol	External !		NO !
L	name	External !		NO !
L	getOwner	External !		NO !
L	balanceOf	Public !		NO !
L	allowance	External !		NO !
L	approve	Public !		NO !
L	approveMax	External !		NO !
L	transfer	External !		NO !
L	transferFrom	External !		NO !
L	setTradingEnabled	External !		author ized
L	_transferFrom	Internal		
L	_basicTransfer	Internal		
L	checkTxLimit	Internal		
L	shouldTakeFee	Internal		

FULL AUDIT REPORT



Contract	Type	Bases		
L	getTotalFee	Public !		NO !
L	getMultipliedFee	Public !		NO !
L	takeFee	Internal 🔒	🛑	
L	shouldSwapBack	Internal 🔒		
L	swapBack	Internal 🔒	🛑	swapping
L	shouldAutoBuyback	Internal 🔒		
L	triggerZeusBuyback	External !	🛑	authorized
L	clearBuybackMultiplier	External !	🛑	authorized
L	triggerAutoBuyback	Internal 🔒	🛑	
L	buyTokens	Internal 🔒	🛑	swapping
L	setAutoBuybackSettings	External !	🛑	authorized
L	setBuybackMultiplierSettings	External !	🛑	authorized
L	launched	Internal 🔒		
L	launch	Public !	🛑	authorized
L	setTxLimit	External !	🛑	authorized
L	setIsDividendExempt	External !	🛑	authorized
L	setIsFeeExempt	External !	🛑	authorized
L	setIsTxLimitExempt	External !	🛑	authorized

FULL AUDIT REPORT

Contract	Type	Bases		
L	setFees	External !		author ized
L	setFeeReceivers	External !		author ized
L	setSwapBackSettings	External !		author ized
L	setTargetLiquidity	External !		author ized
L	setDistributionCriteria	External !		author ized
L	setDistributorSettings	External !		author ized
L	getCirculatingSupply	Public !		NO !
L	getLiquidityBacking	Public !		NO !
L	isOverLiquified	Public !		NO !

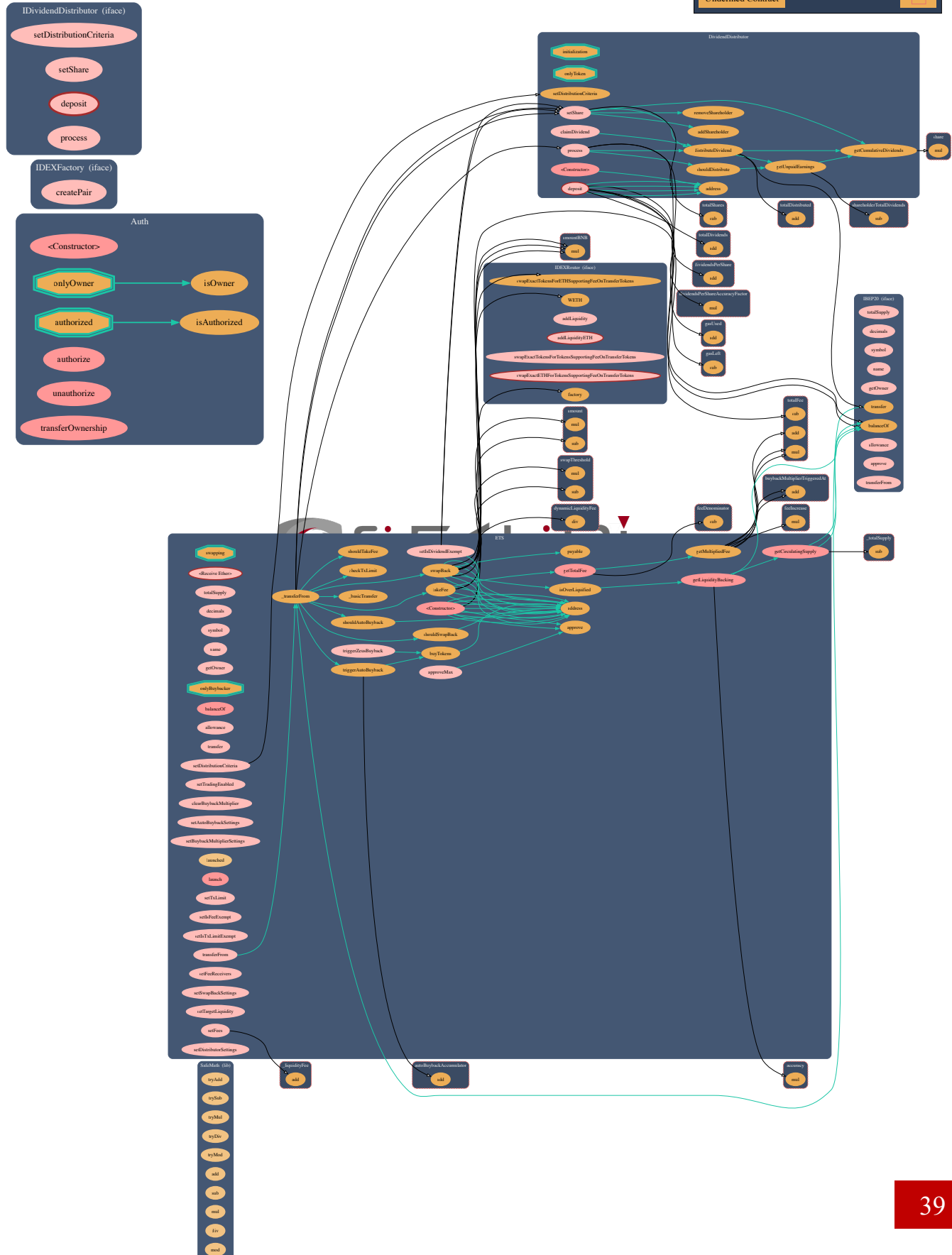
LAB

Legend

Symbol	Meaning
	Function can modify state
	Function is payable

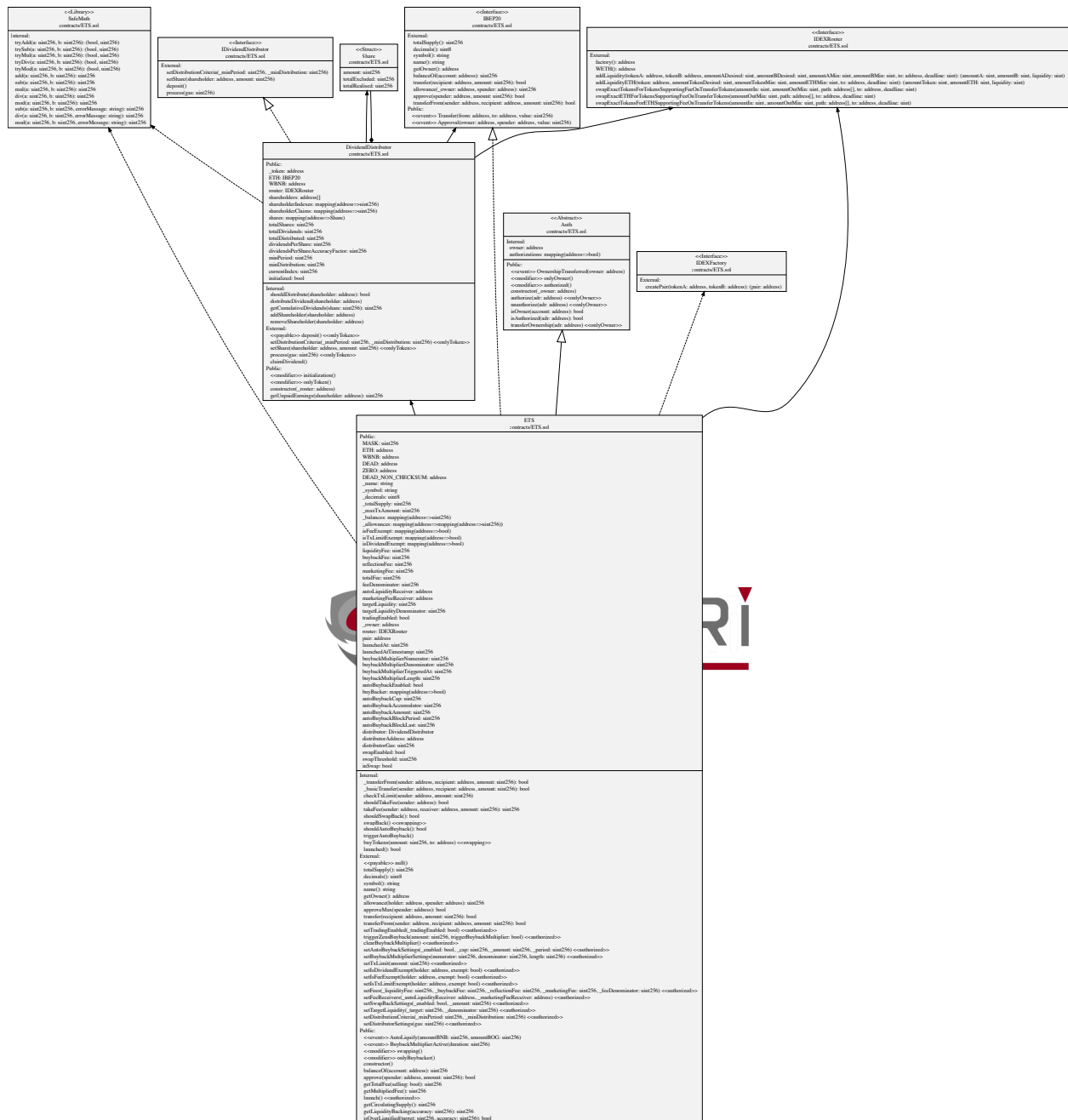
FULL AUDIT REPORT

Inheritate Function Relation Graph



FULL AUDIT REPORT

UML Class Diagram



FULL AUDIT REPORT

About SECURI LAB

SECURI LAB is a group of cyber security experts providing cyber security consulting, smart contract security audits, and KYC services.



SECURI LAB

**Why US? — High Reliability
Intense Inspection
Affordable Price**

Cybersecurity Audit | KYC | Consultant

Follow Us On:

Website	https://securi-lab.com/
Twitter	https://twitter.com/SECURI_LAB
Telegram	https://t.me/securi_lab
Medium	https://medium.com/@securi