

Full Audit Report

Darkstar 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	Darkstar Security Assessment
Version	v1.1
Client	Darkstar
Language	Solidity
Confidentiality	Public
Contract File	darkstar.sol SHA-1: 94e2f8ed0fb734d38b8da958dc78e745b51990d0
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 LAB~~ 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.

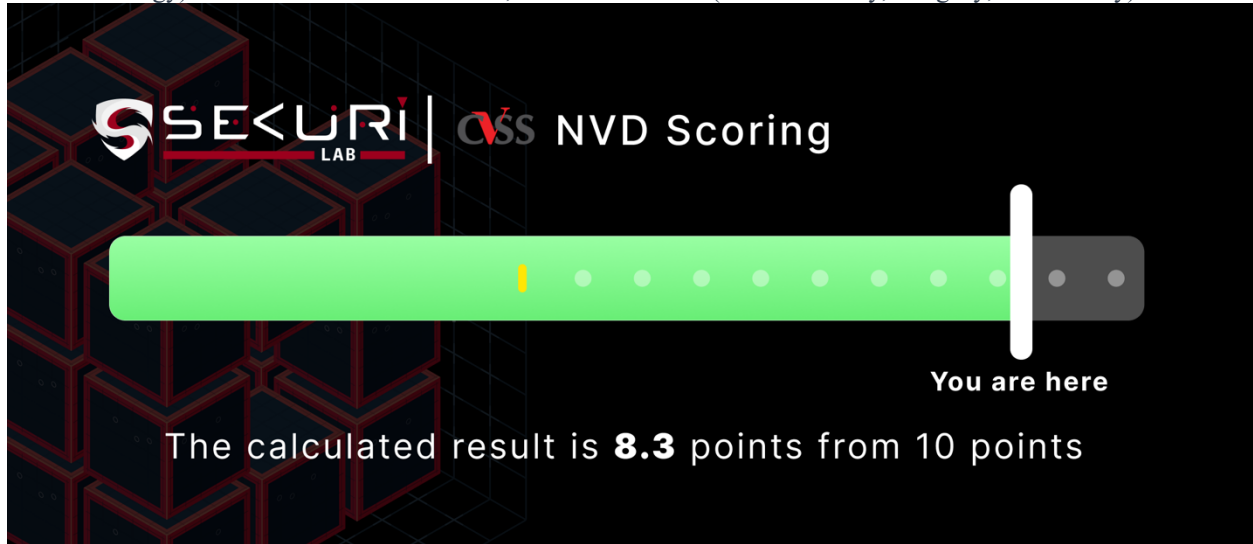
FULL AUDIT REPORT

Executive Summary

For this security assessment, SECURI LAB received a request from **Darkstar on Saturday, March 25, 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	1	1	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 2, 2023



FULL AUDIT REPORT

Project Introduction

Scope Information:

Project Name	Darkstar
Website	https://darkstar.financial/
Chain	BNB Chain
Language	Solidity

Audit Information:

Request Date	Saturday, March 25, 2023
Audit Date	Sunday, March 26, 2023
Re-assessment Date	Sunday, April 2, 2023

Audit Version History:













Version	Date	Description
1.0	Monday, February 27, 2023	Preliminary Report
1.1	Sunday, April 2, 2023	Full Audit Report With Re-Assessment

FULL AUDIT REPORT

Initial Audit Scope:

Smart Contract File	darkstar.sol
Compiler Version	v0.8.17

File Name	SHA-1 Hash
contracts/darkstar.sol	94e2f8ed0fb734d38b8da958dc78e745b51990d0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex Score	Capabilities
  	contracts/darkstar.sol	3	4	780	707	588	2	503	  
  	Totals	3	4	780	707	588	2	503	  

- **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

Re-assessment Audit Scope:

Smart Contract File	darkstar.sol
Compiler Version	v0.8.17

File Name	SHA-1 Hash
contracts/darkstar.sol	e1f524d12413dd58630198b26d8bdef81eabba55

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex Score	Capabilities
	contracts/darkstar.sol	3	4	764	719	611	3	503	
	Totals	3	4	764	719	611	3	503	

- **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,

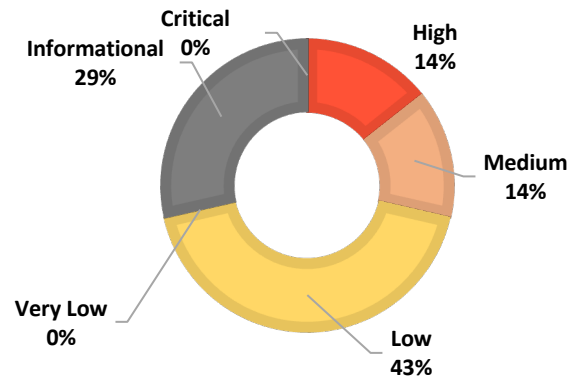
$Risk\ rating = impact * confidence$

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	1
Medium	1
Low	3
Very Low	0
Informational	2
Non-class/Optimization/Best-practices	5

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	Centralization Risk	High	Centralization	Acknowledge
SEC-02	Imprecise arithmetic operations order (divide-before-multiply)	Medium	Mathematical	Resolved
SEC-03	Avoid using block timestamp	Low	Best Practices	Acknowledge
SEC-04	Empty Function Body - Consider commenting why	Low	Best Practices	Resolved
SEC-05	Unsafe ERC20 operation(s)	Low	Best Practices	Mitigate
SEC-06	Conformity to Solidity naming conventions (naming-convention)	Informational	Naming Conventions	Resolved
SEC-07	Conformance to numeric notation best practices	Informational	Best Practices	Resolved
GAS-01	Use `selfbalance()` instead of `address(this).balance`	-	Gas Optimization	Acknowledge
GAS-02	Use assembly to check for `address(0)`	-	Gas Optimization	Resolved
GAS-03	`array[index] += amount` is cheaper than `array[index] = array[index] + amount` (or related variants)	-	Gas Optimization	Resolved
GAS-04	Using bools for storage incurs overhead	-	Gas Optimization	Resolved
GAS-05	State variables should be cached in stack variables rather than re-reading them from storage	-	Gas Optimization	Resolved

FULL AUDIT REPORT

SEC-01: Centralization Risk

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

Finding:

```
21: abstract contract Auth {  
  
38:     function authorize(address adr) public onlyOwner {  
  
42:     function unauthorize(address adr) public onlyOwner {  
  
54:     function transferOwnership(address payable adr) public onlyOwner {  
  
320: contract Darkstar is IBEP20, Auth {  
  
383:     ) Auth(msg.sender) {  
...  

```

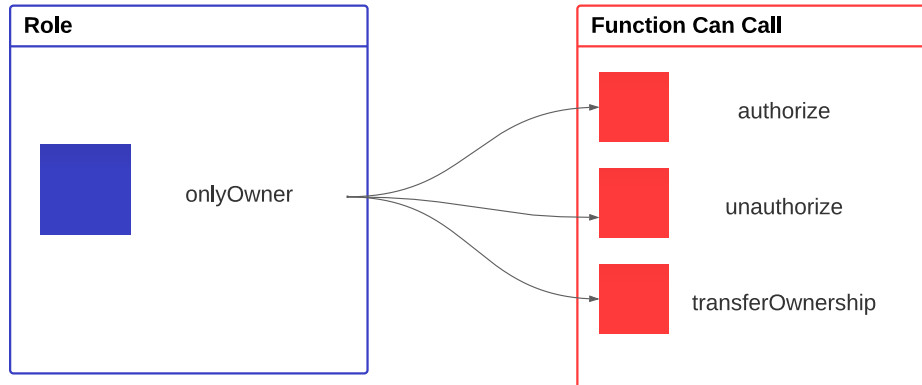
Scenario:

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.

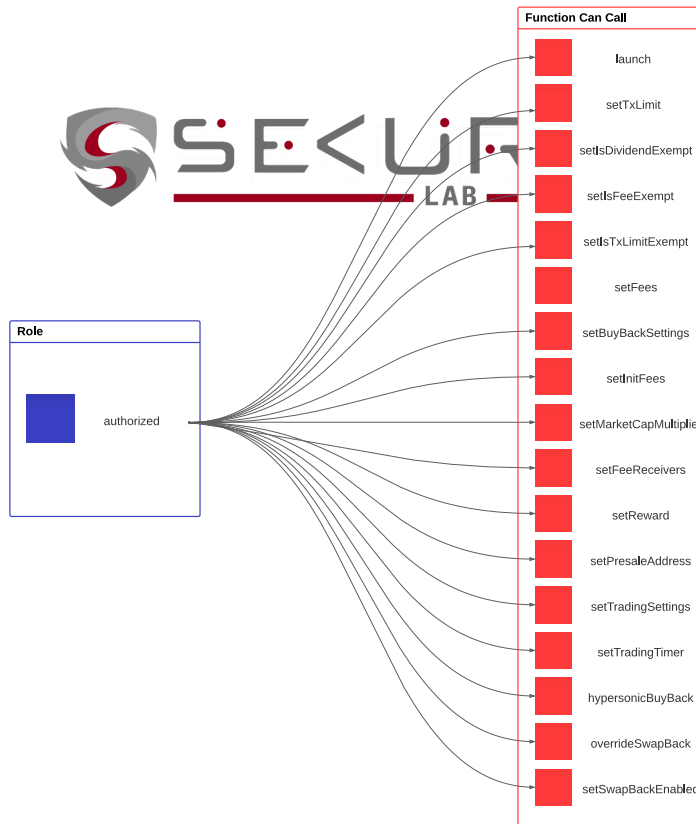
FULL AUDIT REPORT

Contract Auth



The aforementioned function in the authentication contract can only be invoked by the onlyOwner. This contract permits calling of authorized, unauthorized, and transferOwnership functions. Additionally, the implementation of a multi-signature feature adds another layer of security to safeguard the owner's account.

Contract Darkstar



From an authorized role where contract auth can external call to contract darkstar, the risk of The centralization risk is that functions such as 'setTxLimit' 'setFees' 'setFeeReceivers' and so on can be configured as shown in the diagram above.

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:

“if ever we have to use any authorized function after deployment it will decided only with a community vote in our group.” --- From Darkstar Team



FULL AUDIT REPORT

SEC-02: Imprecise arithmetic operations order (divide-before-multiply)

Vulnerability Detail	Severity	Location	Category	Status
Imprecise arithmetic operations order (divide-before-multiply)	Medium	Check on finding	Mathematical	Resolved

Finding:

```
Darkstar.initPresaleInvestor(address,uint256) (darkstar (1).sol:583-598) performs a multiplication on the result of a division:
  • factor = (amount * feeDenominator) / rate (darkstar (1).sol#593)
  • investedAmount = (value * factor) / feeDenominator (darkstar (1).sol#594)
Darkstar.initPresaleInvestor(address,uint256) (darkstar (1).sol:583-598) performs a multiplication on the result of a division:
  • investedAmount = (value * factor) / feeDenominator (darkstar (1).sol#594)
  • taxedInvestedAmount = (investedAmount * multiplier) / denominator (darkstar (1).sol#595)
Darkstar.swapBack() (darkstar (1).sol:712-745) performs a multiplication on the result of a division:
  • amountBNBMarketing = (amountBNB * marketingFee) / totalFee (darkstar (1).sol#734)
  • buyBackAmount = (amountBNBMarketing * buyBackFee) / feeDenominator (darkstar (1).sol#737)
```

Recommendation:

Consider ordering multiplication before division.

To improve the precision of these calculations and reduce the risk of potential rounding errors, consider reordering the operations to perform multiplication before division example:

```
uint256 factor = (amount * feeDenominator) / rate;
uint256 investedAmount = (value * amount * feeDenominator) / (rate * feeDenominator);
```

By reordering the operations, you can help maintain precision and reduce the risk of rounding errors in your calculations.

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply>

FULL AUDIT REPORT

Exploit Scenario:

Solidity's integer division truncates. Thus, performing division before multiplication can lead to precision loss.

```
contract A {  
    function f(uint n) public {  
        coins = (oldSupply / n) * interest;  
    }  
}
```

If n is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5`; $n = 10$, `interest = 2`, `coins` will be zero.

If $(\text{oldSupply} * \text{interest} / n)$ was used, `coins` would have been 1.

In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

Alleviation:

Darkstar Team has already resolved 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:

```
246:         block.timestamp
487:         if(swapBackTime + fullTradingOnTimer < block.timestamp) {
496:             if (fullTradingTime + fullTradingOffTimer < block.timestamp) {
683:                 fullTradingTime = block.timestamp;
687:                 claimTime = block.timestamp;
695:                 && claimTime + claimOffTimer < block.timestamp;
726:                 block.timestamp
729:                 swapBackTime = block.timestamp;
757:                 block.timestamp
```

Recommendation:

Using the block timestamp in a smart contract can introduce potential vulnerabilities related to time manipulation by miners. Miners can slightly manipulate the timestamp of the block they mine, which can affect the outcome of smart contract functions that rely on the block timestamp. In general, the impact of such manipulation is limited, but it's still a good idea to minimize reliance on the block timestamp when possible.

FULL AUDIT REPORT

Exploit Scenario:

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

Alleviation:

“we are not using it for randomness and that with respect to 15 second rule we may be safe from miner attacks as minimum interval that we have is 24 hrs -> so not less than 15 sec”

--- Darkstar Team



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	Best Practices	Resolved

Finding:

```
402:     receive() external payable { }  
    ^
```

Recommendation:

If a function in your smart contract has an empty body, it's essential to provide clear and concise comments explaining the reason for the empty body. This practice helps improve code readability and maintainability, ensuring that other developers can understand your code's intent and behavior.

Exploit Scenario:

-



Alleviation:

Darkstar Team has already resolved this issue.

FULL AUDIT REPORT

SEC-05: Unsafe ERC20 operation(s)

Vulnerability Detail	Severity	Location	Category	Status
Unsafe ERC20 operation(s)	Low	Check on finding	Best Practices	Mitigate

Finding:

```
264:         bool success = reward.transfer(shareholder, rewardAmount);  
  
742:         payable(marketingFeeReceiver).transfer(amountBNBMarketing);  
  
...
```

Recommendation:

Unsafe ERC20 operations can lead to unexpected behavior and potential vulnerabilities in your smart contracts. To mitigate these risks, consider the following recommendations for safer ERC20 operations:

1. Check the return value of ERC20 functions:

Always check the return value of ERC20 functions like **transfer**, **transferFrom**, and **approve**. These functions return a boolean value that indicates whether the operation was successful or not. Make sure to handle potential failures accordingly.

Exploit Scenario:

-

Alleviation:

"we do check for the bool at line 264 and revert reentrancy guard if transaction fails. For line 742 the function swapback will revert if the transfer fails during pre computation and nothing will be executed" --- Darkstar Team

FULL AUDIT REPORT

SEC-06: Conformity to Solidity naming conventions (naming-convention)

Vulnerability Detail	Severity	Location	Category	Status
Conformity to Solidity naming conventions (naming-convention)	Informational	Check on finding	Naming Conventions	Resolved

Finding:

```

✗ Constant Darkstar._decimals (darkstar (1).sol:333) is not in
UPPER_CASE_WITH_UNDERSCORES
✗ Constant Darkstar._name (darkstar (1).sol:331) is not in
UPPER_CASE_WITH_UNDERSCORES
✗ Constant Darkstar._symbol (darkstar (1).sol:332) is not in
UPPER_CASE_WITH_UNDERSCORES
✗ Constant Darkstar._totalSupply (darkstar (1).sol:334) is not in
UPPER_CASE_WITH_UNDERSCORES

```

Recommendation:

Follow the Solidity [naming convention](<https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#naming-conventions>).

Exploit Scenario:

-

Alleviation:

Darkstar Team has already resolved this issue.

FULL AUDIT REPORT

SEC-07: Conformance to numeric notation best practices

Vulnerability Detail	Severity	Location	Category	Status
Conformance to numeric notation best practices (too-many-digits)	Informational	Check on finding	Best Practices	Resolved

Finding:

```
Darkstar.initPresaleInvestor(address,uint256) (darkstar (1).sol:583-598) uses literals with too many digits:  
  • amounts = router.getAmountsOut(1000000000000000000,path) (darkstar (1).sol#587-590)
```

Recommendation:

Description

Literals with many digits are difficult to read and review.

Use:

- [Ether suffix](<https://solidity.readthedocs.io/en/latest/units-and-global-variables.html#ether-units>),
- [Time suffix](<https://solidity.readthedocs.io/en/latest/units-and-global-variables.html#time-units>), or
- [The scientific notation](<https://solidity.readthedocs.io/en/latest/types.html#rational-and-integer-literals>)



Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits>

Exploit Scenario:

```
contract MyContract{  
    uint 1_ether = 1000000000000000000;  
}
```

While 1_ether looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

Alleviation:

Darkstar Team has already resolved 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:

```
732:         uint256 amountBNB = address(this).balance;
    ^
```

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:

Darkstar 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	Resolved

Finding:

```
55:         require(address(adr) != address(0));  
  
651:         require(address(_marketingFeeReceiver) != address(0));  
  
...
```

Recommendation:

Saves 6 gas per instance

Instances (2):

Alleviation:

Darkstar Team has already resolved this issue.



FULL AUDIT REPORT

GAS-03: `array[index] += amount` is cheaper than `array[index] = array[index] + amount` (or related variants)

Vulnerability Detail	Severity	Location	Category	Status
`array[index] += amount` is cheaper than `array[index] = array[index] + amount` (or related variants)	-	Check on finding	Gas Optimization	Resolved

Finding:

```
468:         balances[recipient] = balances[recipient] + amountReceived;
480:         balances[recipient] = balances[recipient] + amount;
529:             totalPurchased[sender] = totalPurchased[sender] - tokenValue;
536:             totalPurchased[recipient] = totalPurchased[recipient] +
taxedValue;
565:         balances[address(this)] = balances[address(this)] + feeAmount;
596:         totalPurchased[shareholder] = totalPurchased[shareholder] +
taxedInvestedAmount;
...

```

Recommendation:

When updating a value in an array with arithmetic, using `array[index] += amount` is cheaper than `array[index] = array[index] + amount`.

This is because you avoid an additional `mload` when the array is stored in memory, and an `sload` when the array is stored in storage.

This can be applied for any arithmetic operation including `+=`, `-=`, `/=`, `*=`, `^=`, `&=`, `%=` , `<<=`, `>>=`, and `>>>=`.

This optimization can be particularly significant if the pattern occurs during a loop.

Saves 28 gas for a storage array, 38 for a memory array

Alleviation:

Darkstar Team has already resolved this issue.

FULL AUDIT REPORT

GAS-04: Using bools for storage incurs overhead

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

Finding:

```
23:      mapping (address => bool) internal authorizations;

142:      mapping (address => bool) public isDiamondHand;

143:      mapping (address => mapping (uint256 => bool)) public isDistributed;

157:      bool public mockPeriod;

342:      mapping (address => bool) isFeeExempt;

343:      mapping (address => bool) isTxLimitExempt;

344:      mapping (address => bool) isDividendExempt;

372:      bool public swapEnabled = true;

373:      bool public buyBackEnabled = true;

374:      bool public fullTradingEnabled;

...

```

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>).

Instances (10):

Alleviation:

Darkstar Team has already resolved this issue.

FULL AUDIT REPORT

GAS-05: State variables should be cached in stack variables rather than re-reading them from storage

Vulnerability Detail	Severity	Location	Category	Status
State variables should be cached in stack variables rather than re-reading them from storage	-	Check on finding	Gas Optimization	Resolved

Finding:

```
231:          rewardPerShare = (rewardPerShareAccuracyFactor * totalRewardAmount) /
totalRewardShares;

472:          if(!isDividendExempt[recipient]){ distributor.setShare(recipient,
balances[recipient], fullTradingEnabled, soldMore); }

472:          if(!isDividendExempt[recipient]){ distributor.setShare(recipient,
balances[recipient], fullTradingEnabled, soldMore); }

528:          breakeven = breakeven - tokenValue;

529:          totalPurchased[sender] = totalPurchased[sender] - tokenValue;

611:          distributor.setShare(holder, balances[holder], fullTradingEnabled,
false);

...

```

Recommendation:

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (100 gas) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

Saves 100 gas per instance

Alleviation:

Darkstar Team has already resolved 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
SWC-116	Block values as a proxy for time	Complete	No risk

FULL AUDIT REPORT

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
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Complete	No risk

FULL AUDIT REPORT

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	0	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	4			
External	Internal	Private	Pure	View
51	73	0	8	26

StateVariables

Total	 Public
72	52




Capabilities

 Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
0.8.17		yes	yes (1 asm blocks)		
 Transfers ETH	 Low-Level Calls	 Delegate Call	 Uses Hash Functions	 ECRECOVER	 New/Create/Create2
yes					yes → NewContract:RewardDistributor



FULL AUDIT REPORT

 TryCatch	Σ Unchecked
<input type="text"/>	<input type="text" value="yes"/>



FULL AUDIT REPORT

Contracts Description Table

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
IBEP20	Interface			
L	totalSupply	External !		NO !
L	decimals	External !		NO !
L	symbol	External !		NO !
L	name	External !		NO !
L	getOwner	External !		NO !
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
L	isNullAddress	Public !		NO !

FULL AUDIT REPORT

Contract	Type	Bases		
IDEXFactory	Interface			
L	createPair	External !		NO !
IDEXRouter	Interface			
L	factory	External !		NO !
L	WETH	External !		NO !
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External !		NO !
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External !		NO !
L	getAmountsOut	External !		NO !
IRewardDistributor	Interface			
L	setShare	External !		NO !
L	deposit	External !		NO !
RewardDistributor	Implementation	IRewardDistributor		
L		Public !		NO !
L	getShareholderCount	Public !		NO !
L	getDiamondHolderCount	Public !		NO !
L	removePresaleAddress	External !		onlyToken
L	setMockPeriod	External !		onlyToken
L	setReward	External !		onlyToken

FULL AUDIT REPORT

Contract	Type	Bases		
L	setShare	External !		onlyTo ken
L	deposit	External !		onlyTo ken
L	claimRewards	External !		NO !
L	distributeReward	Internal		
L	getUnpaidRewardEarnings	Public !		NO !
L	getCumulativeRewards	Internal		
L	addShareholder	Internal		
L	addDiamondHolder	Internal		
L	removeShareholder	Internal		
L	removeDiamondHolder	Internal		
Darkstar	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 !

FULL AUDIT REPORT

Contract	Type	Bases		
L	approveMax	External !		NO !
L	transfer	External !		NO !
L	transferFrom	External !		NO !
L	sub	Internal		
L	_transferFrom	Internal		
L	basicTransfer	Internal		
L	checkFullTrading	Internal		
L	checkTxLimit	Internal		
L	setTotalPurchased	Internal		
L	getTokenValue	Public !		NO !
L	getCurrentMarketCap	Public !		NO !
L	shouldTakeFee	Internal		
L	takeFee	Internal		
L	launched	Internal		
L	launch	External !		author ized
L	initPresaleInvestor	Internal		
L	setTxLimit	External !		author ized
L	setIsRewardExempt	External !		author ized
L	setIsFeeExempt	External !		author ized
L	setIsTxLimitExempt	External !		author ized



FULL AUDIT REPORT

Contract	Type	Bases		
L	setFees	External !		author ized
L	setBuyBackSettings	External !		author ized
L	setInitFees	External !		author ized
L	setMarketCapMultiplier	External !		author ized
L	setReward	External !		author ized
L	setPresaleAddress	External !		author ized
L	setFeeReceivers	External !		author ized
L	overrideFullTrading	External !		author ized
L	setTradingTimer	External !		author ized
L	setFullTrading	Internal		
L	shouldEnableSwap	Internal		
L	shouldSwapBack	Internal		
L	swapBack	Internal		swapp ing
L	buyBackTokens	Internal		swapp ing
L	hypersonicBuyBack	External !		author ized
L	overrideSwapBack	External !		author ized
L	setSwapBack	External !		author ized

FULL AUDIT REPORT

Contract	Type	Bases		
L	getCirculatingSupply	Public !		NO !
L	ownerBalance	Public !		

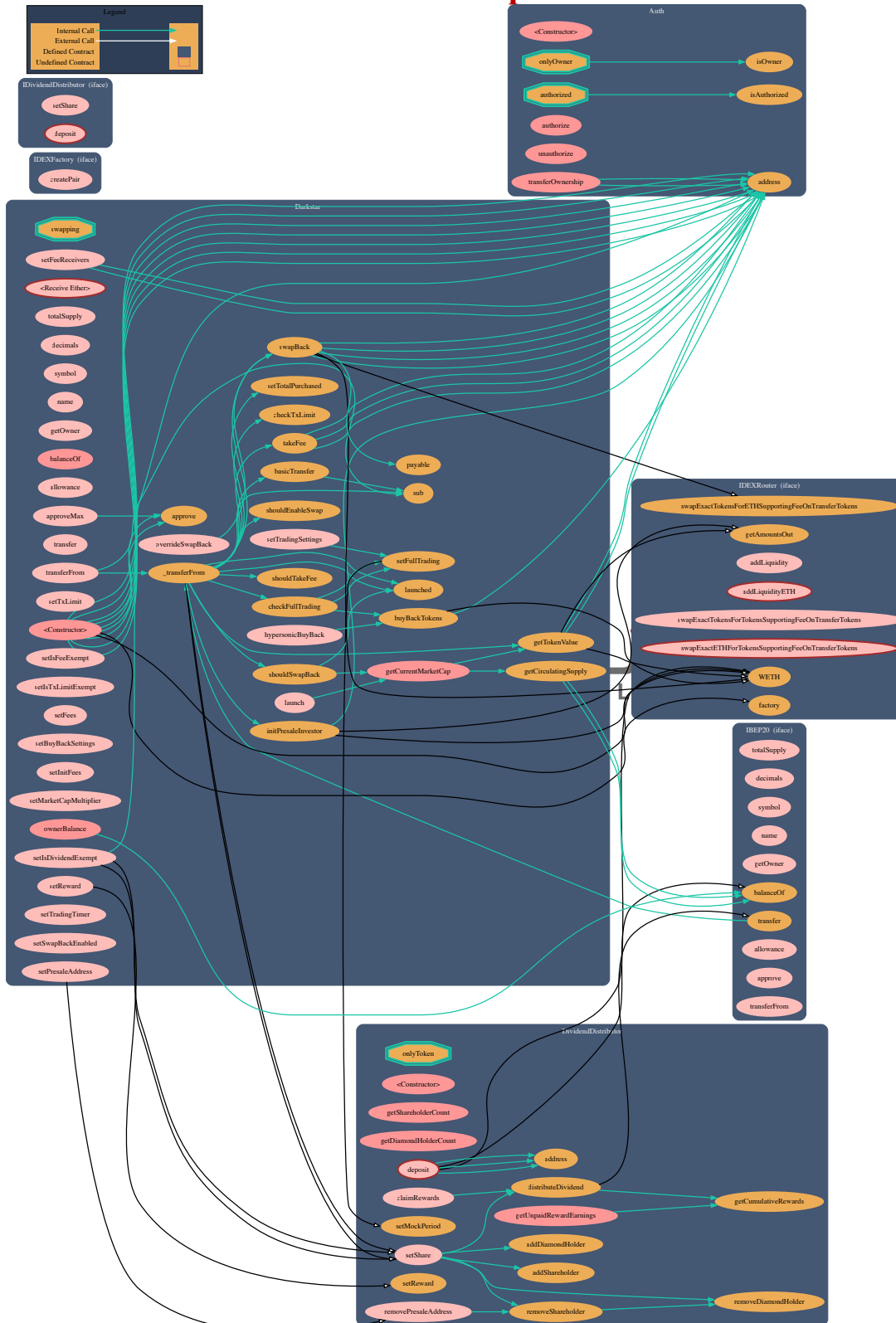
Legend

Symbol	Meaning
	Function can modify state
	Function is payable



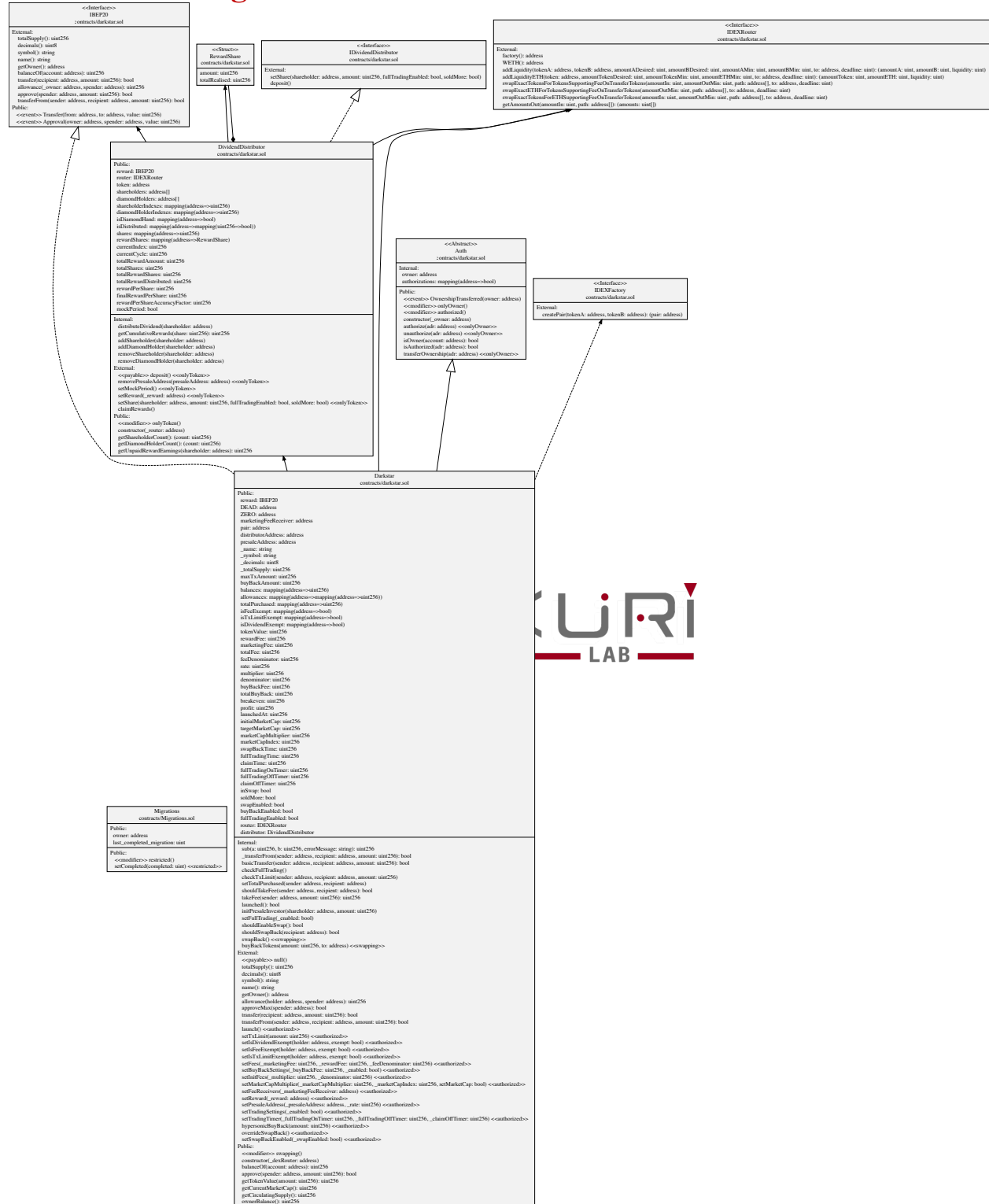
FULL AUDIT REPORT

Inheritate Function Relation Graph



FULL AUDIT REPORT

UML Class Diagram



FULL AUDIT REPORT

About SECURI LAB

Enhance the security and legitimacy of your blockchain project with our professional Audit & KYC services. Our experienced team provides reliable, cost-effective, and secure verification processes.



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