Cyberscope

Audit Report
Nabana

January 2023

Type	BEP20
Network	BSC
Address	0x760fCd32526e24ba9DFdeD84B0FCbE7591d13536
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Nabana
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x760fcd32526e24ba9dfded84b0fcbe7591d13536
Address	0x760fcd32526e24ba9dfded84b0fcbe7591d13536
Network	BSC
Symbol	BANA
Decimals	18
Total Supply	50,000,000

Audit Updates

Initial Audit	02 Jan 2023

Source Files

Filename	SHA256
Nabana.sol	c46f9c6601d5e569cf056a940b25ca99bdaf8a71c642956cf739d2c2cd8e4f89

Introduction

The Nabana ecosystem currently consists a digital, multi-utility token and a vesting contract. This audit investigates security issues, business logic concerns and potential improvements for the **Nabana Token** and **Nabana DividendTracker** contracts.

The **Nabana Vesting** contract audit can be found at https://github.com/cyberscope-io/audits/tree/main/nabana/NabanaVesting.pdf.

Nabana Token

The **Nabana Token** contract implements a BEP20 standard token interface.

Roles

The Nabana Token contract has two roles, the USER, the DIVIDEND_TRACKER and the OWNER role.

USER

The USER role has the authority to

- Execute the BEP20 public methods

DIVIDEND_TRACKER

The DIVIDEND_TRACKER role has the authority to

- Mint the rewarded amount of a private inverstor's unlocked investment to its account's balance.

OWNER

The OWNER role has the authority to

- Transfer/Renounce ownership.
- Update purchase/sell fees up to 3%/5% respectively.
- Exclude/Include accounts from fees.
- Blacklist addresses.

Nabana DividendTracker

The **Nabana DividendTracker** contract implements a variation of a DividendTracker mechanism with some additional business logic to work closely with the **Nabana Token**. It distributes tokens to the holders of the **Nabana Token**.

Roles

The NabanaDividendTracker contract has two roles, the USER and the OWNER role.

USER

The USER role has the authority to

- Retrieve various information regarding an account's investment.

OWNER

The OWNER role has the authority to

- Set a new investment for an account.
- Increase/Decrease an account's invested amount.
- Alter the claim time period to any value without restriction.
- Process an account's investment.

Nabana Token Allocations

The Nabana token total supply is 50,000,000 tokens. These tokens were distributed in the following way:

Allocation	Amount	Total Supply Percentage
Pancake Swap Launch	15,000	0.03
Airdrops	1,709	0.003418
Private Investors	10,000,000	20
Operations	19,983,291	39.966582
Vesting	20,000,000	40

Nabana After Lockup Period Earnings

The Nabana ecosystem locks some of the funds in a vesting contract. All the relevant information regarding the amounts and percentages an address can earn each month can be seen at the table below.

Vesting

Allocation	Lockup Period (months)	Amount Percentage (per month)	Amount (per month)	Time Applicable (months)
Management	36	4	200,000	25
Advisors	12	2	20,000	50
Influencers	6	4	40,000	25
Strategic Partners	6	4	120,000	25
Rewards/Incentives	2	100	5,000,000	1
Reserve	12	4	200,000	25

Private Investors (Dividend Tracker)

The lockup period for the **Private Investors** lasts 6 months. After that, all investors have the authority to sell 10% of their investment each month. Additionally, every investor earns 2% of their initial investment in tokens every month for 6 months. It should be noted, as already mentioned in the **Roles** section, that the owner has the authority to alter the period between every claim.

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
🟠	ST	Stops Transactions	Unresolved
🟠	BC	Blacklists Addresses	Unresolved
●	DTA	Dividend Tracker Architecture	Unresolved
●	SCP	Solidity Code Principles	Unresolved
●	US	Untrusted Source	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved

ST - Stops Transactions

Criticality	Medium
Location	Nabana.sol#L2076,2130
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting the

- `minSellTransactionAmount` to a high value and/or `maxSellTransactionAmount` to zero.
- `handleLimitTime` to a high value.

The contract can limit users from selling more than 20% of the circulating supply for 24 hours (initial `handleLimitTime` value) if their balance is more than 1% of the `totalSupply`. Moreover, during sales the contract limits the transaction amount to a lower and upper bound (initial values are 0.02% and 2% of the `totalSupply` respectively).

```
require(minSellTransactionAmount <= amount && amount <= maxSellTransactionAmount,
"Nabana: Sell transfer amount should be within min and max of sell transaction
limit.");
```

Recommendation

The contract could embody a check for not allowing setting the minSellTransactionAmount more than a reasonable amount and/or maxSellTransactionAmount less than a reasonable amount. A suggested implementation could check that the maximum/minimum amount should be more/less than a fixed percentage of the total supply.

Respectively the `handleLimitTime` variable should be able to be configured no more than a reasonable time period. The team could exploit the error message that says `try again after 24 hours`. Currently, the contract owner can set any value to the `handleLimitTime`.

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

BC - Blacklists Addresses

Criticality	Critical
Location	Nabana.sol#L1958
Status	Unresolved

Description

The contract owner has the authority to massively stop addresses from transactions. The owner may take advantage of it by calling the `blackListMultipleAddresses` function.

```
function blackListMultipleAddresses(address[] calldata accounts, bool blacklisted)
public onlyOwner {
  for(uint256 i = 0; i < accounts.length; i++) {
    _isBlacklisted[accounts[i]] = blacklisted;
  }

  emit BlackListMultipleAddresses(accounts, blacklisted);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

DTA - Dividend Tracker Architecture

Criticality	Minor / Informative
Location	Nabana.sol#L2260
Status	Unresolved

Description

An ordinary DividendTracker usually performs expensive gas calls to sell tokens, get the rewarded token and distribute it progressively to the users. The contract's implementation does not interact with other tokens, either by swapping or transfering. As a result, the gas cost for the NABANA business logic is much smaller. The architectural desicion to extract it as an external contract increases dramatically the gas fee, as all the calls being made, including the minting process, consume gas like an external contract.

```
contract NabanaDividendTracker is Ownable {}
```

Recommendation

The team is advised to reconsider the business logic of using an external contract. A recommended approach could be implementing a simpler, internal solution, which would cost less gas and be less complicated.

SCP - Solidity Code Principles

Criticality	Minor / Informative
Location	Nabana.sol#L2632,L2186
Status	Unresolved

Description

As with any programming task, it is important to keep the Solidity code as simple and straightforward as possible. This can help avoid errors and make it easier for others to understand and review the code.

The `processInvestment, setBalance` functions returns an integer based on some conditions, which makes it unclear of what the function's result really is.

```
function processInvestment(address account, uint256 index, bool manual) public
onlyOwner returns (uint8) {
  ...
  if (investment.currentInterval >= totalIntervals) {
    ...
    uint8 result = setBalance(account,
currentInvestment.safeSub(investment.amount));
    ...
    return result;
  }

  return 1;
}
```

Additionally, there are some code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

During a transaction, if the sender is an investor and the amount is smaller or equal to the `unlockedInvestments`, then the contract performs the same action twice. Once for the calculated `totalFee` and once for the remaining `amount`, which is redundant since it can only be performed only once with the whole amount.

```
if (dividendTracker.isPrivateInvestor(from)) {
  require(totalFee <= dividendTracker.unlockedInvestments(from), "Nabana: Amount
exceeds unlocked amount");
  dividendTracker.decreaseUnlockedInvestments(from, totalFee);
}
...
if (dividendTracker.isPrivateInvestor(from)) {
  require(amount <= dividendTracker.unlockedInvestments(from), "Nabana: Amount
exceeds unlocked amount");
  dividendTracker.decreaseUnlockedInvestments(from, amount);
}
```

Recommendation

The team is advised to follow Solidity's Code Principles in the best way possible. Some recommendations are to use descriptive names for variables, avoid unnecessary complexity and aim for clarity in the code. The `processInvestment` and `setBalance` functions count return a descriptive value of what the result actually is.

The team is advised to take into consideration code segments like double `decreaseUnlockedInvestments` and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

US - Untrusted Source

Criticality	Minor / Informative
Location	Nabana.sol#L1857
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. If the owner’s account is compromised, then the external contract could be set to an untrusted source. This behavior can be combined with the fact that the contract wraps the functions of the external contract in a try-catch block, while others do not. As a result, it may produce inconsistency in the transactions.

```
function updateDividendTracker(address newAddress) public onlyOwner {}
...
dividendTracker.setInvestment(to, amount);
```

Recommendation

The pointing addresses of the dividend tracker variable could be immutable, so it cannot be compromised. If a mutation is required, an extra security measure is to wrap in a try-catch block all calls to the external contract. Hence, the external call errors will not propagate to the main contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Nabana.sol#L649
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases unnecessarily the gas consumption.

```
library SafeMath {}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Nabana.sol#L392,559,560,577,878,883,1318,1366,1703,1704,1705,1706,1707,1708,1709,1710,1711,1712,2413
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
int128 private constant MIN_64x64 = -0x80000000000000000000000000000000
int128 private constant MAX_64x64 = 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Nabana.sol#L806
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Nabana.sol#L1892,1897,1907,1913,1918,1923
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
purchaseFee = amount
sellFee = amount
minSellTransactionAmount = newMinSellTransactionAmount
handleLimitCriteria = newHandleLimitCriteria
handleLimitMax = newHandleLimitMax
handleLimitTime = newHandleLimitTime
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Nabana.sol#L333,852,858,866,892,906,933,947,962,1021,1102,1149,1162,1176,1192,1206,1305,1318,1351,1366,1516,1585
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount
exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	Nabana.sol#L2113,2114
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(_isBlacklisted[from] == false, "Nabana: Cannot send token FROM a blacklisted address")
require(_isBlacklisted[to] == false, "Nabana: Cannot send token TO a blacklisted address")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	Nabana.sol#L2141,2142,2143,2144,2157,2158,2159,2160,2220,2221,2222,2223
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint256 iterations
uint256 claims
uint256 lastProcIndex
uint256 lastProcInvestment
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Nabana.sol#L1553,1556,1557,2667,2671,2680
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 numberOfIntervals =
block.timestamp.sub(investment.lastClaimTime).div(intervalTime)
uint256 amountToUnlock =
investment.amountToUnlock.mul(percentageToUnlock).div(1000).mul(numberOfIntervals)
numberOfIntervals = numberOfIntervals > leftoverIntervals ? leftoverIntervals :
numberOfIntervals
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Nabana.sol#L2101,2141,2142,2143,2144,2157,2158,2159,2160,2174,2220,2221,2222,2223,2333
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
Limiter memory emptyLimiter
uint256 iterations
uint256 claims
uint256 lastProcIndex
uint256 lastProcInvestment
uint256 totalFee
Investment memory investment
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Nabana.sol#L606,1780
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
_owner = msgSender
nabanaVesting = _nabanaVesting
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-

	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-

	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-

	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-

	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
	safeSub	Internal		
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
ABDKMath64x64	Library			
	fromInt	Internal		
	toInt	Internal		
	fromUInt	Internal		

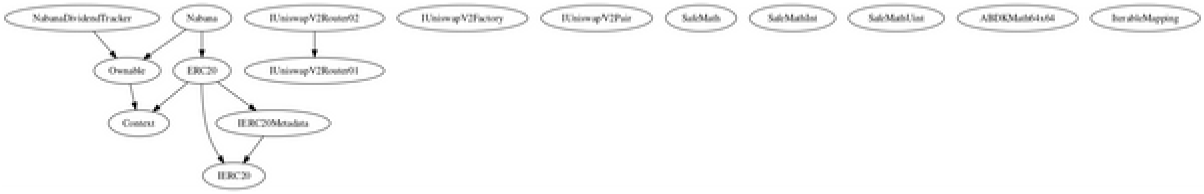
	toUInt	Internal		
	from128x128	Internal		
	to128x128	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	muli	Internal		
	mulu	Internal		
	div	Internal		
	divi	Internal		
	divu	Internal		
	neg	Internal		
	abs	Internal		
	inv	Internal		
	avg	Internal		
	gavg	Internal		
	pow	Internal		
	sqrt	Internal		
	log_2	Internal		
	ln	Internal		
	exp_2	Internal		
	exp	Internal		
	divuu	Private		
	sqrtu	Private		
IterableMapping	Library			
	get	Public		-
	getIndexOfKey	Public		-
	getKeyAtIndex	Public		-

	size	Public		-
	set	Public	✓	-
	remove	Public	✓	-
Nabana	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	updateUniswapV2Router	Public	✓	onlyOwner
	updateFeeWallet	External	✓	onlyOwner
	updateNabanaVesting	External	✓	onlyOwner
	updateDividendTracker	Public	✓	onlyOwner
	updatePurchaseFee	External	✓	onlyOwner
	updateSellFee	External	✓	onlyOwner
	updateMinMaxSellTransactionAmounts	External	✓	onlyOwner
	updateHandleLimitCriteria	External	✓	onlyOwner
	updateHandleLimitMax	External	✓	onlyOwner
	updateHandleLimitTime	External	✓	onlyOwner
	updateGasForProcessing	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner
	blackListAddress	Public	✓	onlyOwner
	blackListMultipleAddresses	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	circulatingSupply	Public		-
	isExcludedFromFees	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-

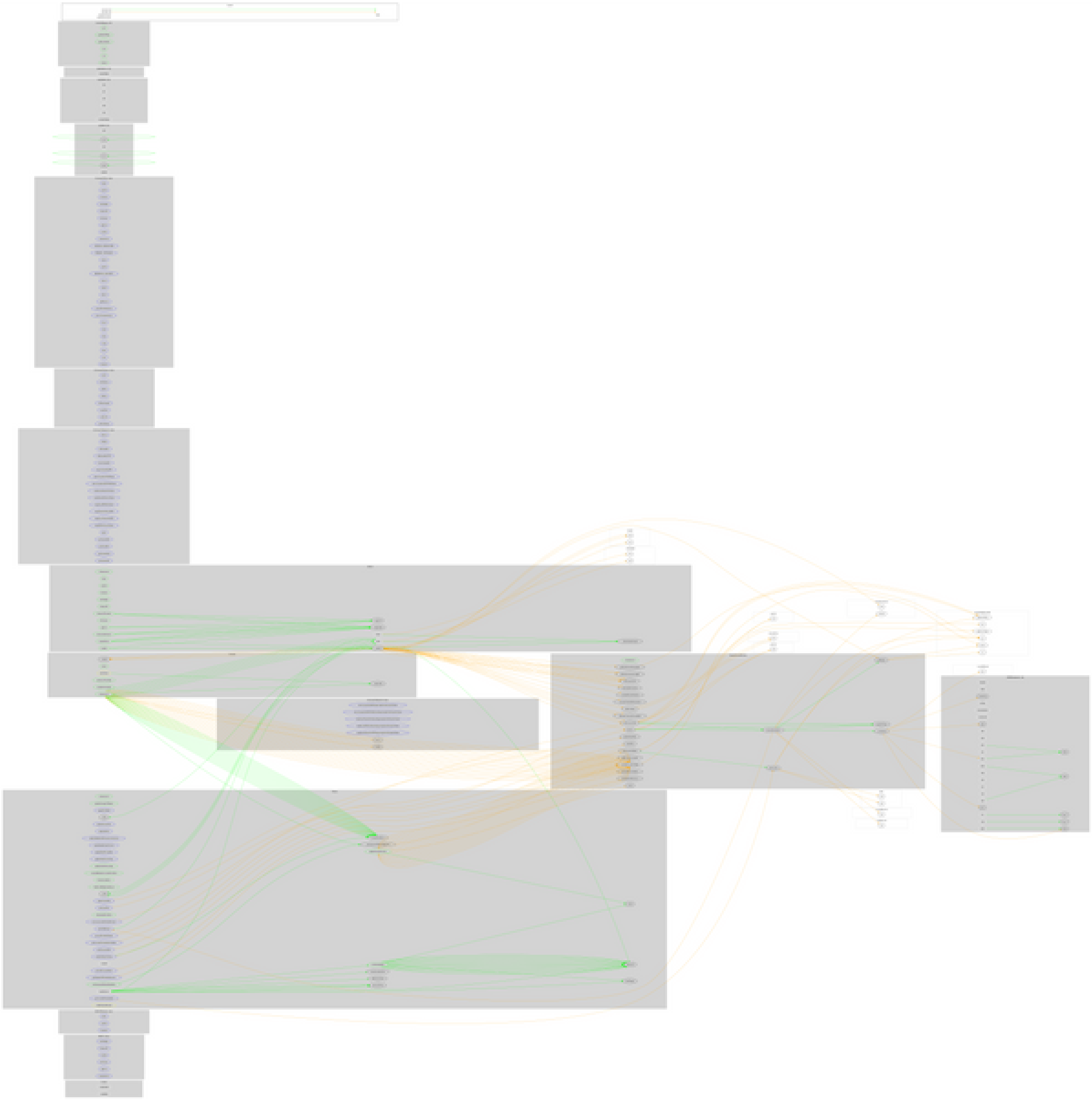
	getInvestmentInfo	External		-
	processDividendTracker	External	✓	-
	processAccountDividendTracker	External	✓	-
	handleLimit	Internal	✓	
	isLimitedUser	Internal		
	addLimitedUser	Internal	✓	
	removeLimitedUser	Internal	✓	
	_transfer	Internal	✓	
	_isBuy	Internal		
	_isSell	Internal		
	getLastProcessedIndex	External		-
	getNumberOfDividendInvestors	External		-
	mintWithPermit	External	✓	onlyDividendTracker
NabanaDividendTracker	Implementation	Ownable		
		Public	✓	-
	setInvestment	External	✓	onlyOwner
	increaseUnlockedInvestments	External	✓	onlyOwner
	decreaseUnlockedInvestments	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfInvestors	External		-
	privateInvestmentsLength	External		-
	getPrivateInvestments	External		-
	getTotalCurrentInvestment	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	canAutoClaim	Private		

	setBalance	Public	✓	onlyOwner
	process	Public	✓	onlyOwner
	processAccount	Public	✓	onlyOwner
	processInvestment	Public	✓	onlyOwner
	_compound	Internal		

Inheritance Graph



Flow Graph



Summary

Nabana Token implements a stands ERC20 interface enriched with a progressively mint mechanism that is called dividend tracker. There are some functions that could be compromised and produce vulnerabilities. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. Additionally, the audit reports some architectural concerns and potential improvements regarding the implementation of the Nabana DividendTracker contract.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io