Cyberscope

Audit Report

Bogdanoff

March 2024

Network	BSC
Address	0x6e8eac45d77cb8bc1528ccb98025e161b86e7460
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	DDP	Decimal Division Precision	Unresolved
●	FSS	Fee Structure Simplification	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	IIL	Inefficient If Logic	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	PAIS	Potential Arbitrary Index Set	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RFF	Redundant Fee Functionality	Unresolved
●	RIC	Redundant If Checks	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	TUU	Time Units Usage	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved

●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Bogdanoff
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x6e8eac45d77cb8bc1528ccb98025e161b86e7460
Address	0x6e8eac45d77cb8bc1528ccb98025e161b86e7460
Network	BSC
Symbol	BOG
Decimals	9
Total Supply	420,000,000,000,000,000
Badge Eligibility	Yes

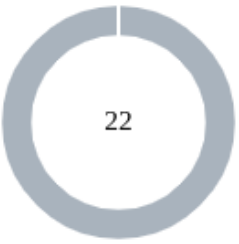
Audit Updates

Initial Audit	07 Mar 2024

Source Files

Filename	SHA256
Bogdanoff.sol	77dda21ba7b78229608da280890a75bf5580bc837da6469f894a95304e9ac9b1

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	22

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	22	0	0	0

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Bogdanoff.sol#L1133
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
if((marketingFeeOnBuy + marketingFeeOnSell) > 0) {
    uint256 marketingBNB = newBalance * (marketingFeeOnBuy +
marketingFeeOnSell) / bnbShare;
    sendBNB(payable(marketingWallet), marketingBNB);
    emit SendMarketing(marketingBNB);

if((rewardsFeeOnBuy + rewardsFeeOnSell) > 0) {
    uint256 rewardBNB = newBalance * (rewardsFeeOnBuy +
rewardsFeeOnSell) / bnbShare;
    swapAndSendDividends(rewardBNB);
}
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

FSS - Fee Structure Simplification

Criticality	Minor / Informative
Location	Bogdanoff.sol#L986
Status	Unresolved

Description

The contract is currently managing its fee logic through individual variables for each type of fee (burn, marketing, rewards) across different transaction types (buy, sell). This approach, while functional, leads to fragmented code that can be challenging to maintain and update, especially as the complexity of the contract evolves. The use of separate variables for each fee type and transaction mode necessitates repetitive calculations and updates, increasing the potential for errors and making the codebase less intuitive. Moreover, this method complicates the addition of new fee types or adjustments to the fee logic, as changes need to be meticulously mirrored across multiple sections of the code.

```
burnFeeOnBuy         = 0;
marketingFeeOnBuy     = 3;
rewardsFeeOnBuy      = 2
totalBuyFee          = burnFeeOnBuy + marketingFeeOnBuy +
rewardsFeeOnBuy
burnFeeOnSell        = 0;
marketingFeeOnSell    = 3;
rewardsFeeOnSell     = 2
totalSellFee         = burnFeeOnSell + marketingFeeOnSell +
rewardsFeeOnSell;
```

Recommendation

It is recommended to adopt a structured approach to managing fees by encapsulating fee-related variables within a struct. This would not only consolidate fee information, making the code more organized and readable but also facilitate easier maintenance and updates to the fee logic. Specifically, implementing a Fees struct for both buy and sell transactions would streamline the process of calculating total fees, updating fee rates, and handling fee distributions. Additionally, this approach would reduce the risk of inconsistencies and errors in fee handling, as modifications would be centralized within the

struct definitions. Adopting this structured approach enhances the contract's scalability, allowing for more straightforward implementation of future enhancements or new fee types.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Bogdanoff.sol#L982,983,984,986,988,989,990,992,994,1003
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
burnFeeOnBuy
marketingFeeOnBuy
rewardsFeeOnBuy
totalBuyFee
burnFeeOnSell
marketingFeeOnSell
rewardsFeeOnSell
totalSellFee
marketingWallet
uniswapV2Pair
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

IIL - Inefficient If Logic

Criticality	Minor / Informative
Location	Bogdanoff.sol#L1143
Status	Unresolved

Description

The contract is currently implementing its fee logic through two separate if conditions, which determine whether a transaction should be subjected to fees. This approach, while functional, introduces unnecessary complexity and redundancy into the fee determination logic. The sequential and disjointed use of conditions to alter the `takeFee` flag could lead to maintenance challenges and increases the risk of logical errors in future modifications. Moreover, it impacts the contract's efficiency, as executing multiple conditions sequentially consumes more gas than necessary.

```
if(_isExcludedFromFees[from] || _isExcludedFromFees[to]) {
    takeFee = false;

// w2w & not excluded from fees
if(from != uniswapV2Pair && to != uniswapV2Pair && takeFee) {
    takeFee = false;
}
```

Recommendation

It is recommended to consolidate the fee determination logic into a single if condition that comprehensively evaluates all scenarios in which fees should not be taken. This can be achieved by combining the conditions using logical OR (||) operators, thus checking for exclusion from fees or a wallet-to-wallet transfer in one step. Simplifying the logic in this manner will not only make the code cleaner and easier to understand but also optimize gas usage for transactions. Furthermore, consolidating the conditions enhances the contract's maintainability by reducing the complexity involved in modifying or extending the fee logic in the future.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	Bogdanoff.sol#L631,682,754
Status	Unresolved

Description

The contract is missing error messages. These are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(totalSupply() > 0)
require(false)
require(!excludedFromDividends[account])
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

PAIS - Potential Arbitrary Index Set

Criticality	Minor / Informative
Location	Bogdanoff.sol#L774
Status	Unresolved

Description

The contract is currently allowing the owner to arbitrarily set the `lastProcessedIndex`, a critical variable used in the management of dividend distributions to token holders. This variable plays a pivotal role in determining which token holders are due for dividend payments and their respective order in the distribution process. The ability for the owner to unilaterally adjust this index at will introduces several risks, including potential manipulation of dividend distributions, unfair treatment of token holders, inefficient gas usage, and susceptibility to both accidental and malicious disruptions. Given the importance of trust and fairness in dividend distribution mechanisms, this feature undermines the perceived integrity and security of the contract, potentially affecting token holder confidence and the overall value proposition of the token.

```
function setLastProcessedIndex(uint256 index) external
onlyOwner {
    lastProcessedIndex = index;
}
```

Recommendation

It is recommended to remove or significantly restrict the ability of the contract owner to arbitrarily set the `lastProcessedIndex`. If this implementation is needed, implementing a more controlled and transparent approach is crucial. Additionally, considering an automated approach to updating the `lastProcessedIndex` can minimize the need for manual adjustments, ensuring a fair, efficient, and tamper-resistant dividend distribution process. These measures will collectively enhance the security, fairness, and trustworthiness of the contract's dividend distribution mechanism.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Bogdanoff.sol#L1090
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
uint256 contractTokenBalance = balanceOf(address(this));

bool canSwap = contractTokenBalance >= swapTokensAtAmount;
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	Bogdanoff.sol#L969
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event UpdateUniswapV2Router(address indexed newAddress, address
indexed oldAddress);
event UpdateDividendTracker(address indexed newAddress, address
indexed oldAddress);
event GasForProcessingUpdated(uint256 indexed newValue, uint256
indexed oldValue);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RFF - Redundant Fee Functionality

Criticality	Minor / Informative
Location	Bogdanoff.sol#L986,992,1099
Status	Unresolved

Description

The contract includes a block of code dedicated to handling burning fees upon buy and sell transactions through variables `burnFeeOnBuy` and `burnFeeOnSell`. These variables are initially set to zero within the constructor, and there are no mechanisms within the contract to alter these values post-deployment. This implementation choice results in the burning fee functionality being effectively redundant, as the condition `burnFeeOnBuy + burnFeeOnSell > 0` will perpetually evaluate to false under the current setup. Consequently, the associated code block designed to calculate and execute token burns will never be triggered, rendering this ostensibly important feature of the contract inactive and superfluous. This redundancy not only increases the contract's complexity without any functional benefit but also potentially misle

```
constructor() payable ERC20("Bogdanoff", "BOG") {
...
burnFeeOnBuy        = 0;
...
burnFeeOnSell       = 0;
...
}
...

uint256 burnToken
if(burnFeeOnBuy + burnFeeOnSell > 0) {
    burnTokens = contractTokenBalance * (burnFeeOnBuy +
burnFeeOnSell) / 100;
    super._transfer(address(this), DEAD, burnTokens);
}
```

Recommendation

It is recommended to re-evaluate the necessity of the burning fee functionality within the contract's economic model. If the intention is to activate this feature under certain conditions or in the future, appropriate mechanisms should be implemented to allow for the dynamic adjustment of the `burnFeeOnBuy` and `burnFeeOnSell` values by authorized parties, such as through a governance process or by the contract owner, while ensuring transparency and security. This could involve adding setter functions with appropriate access controls. Conversely, if the burning fee feature is deemed unnecessary or undesirable, it would be prudent to remove the related code to simplify the contract, and reduce the gas costs for transactions.

RIC - Redundant If Checks

Criticality	Minor / Informative
Location	Bogdanoff.sol#L987,993,
Status	Unresolved

Description

The contract declares the fees for marketing and rewards, which are set during the contract's deployment via the constructor. These fees are defined for both buying and selling transactions, with preset values that remain constant throughout the contract's lifecycle due to the absence of functions capable of modifying these values post-deployment. Consequently, the fee variables, including `marketingFeeOnBuy`, `rewardsFeeOnBuy`, `marketingFeeOnSell`, and `rewardsFeeOnSell`, are effectively immutable. This immutable nature of the fee structure renders the if conditions checking for the presence of fees redundant. Given that the fees are permanently set to be greater than zero, these checks will always evaluate to true, leading to unnecessary computational overhead and gas usage during transactions.

```
constructor() payable ERC20("Bogdanoff", "BOG") {
    rewardToken = 0x7130d2A12B9BCbFAe4f2634d864A1Ee1Ce3Ead9c; // BTCB

    marketingFeeOnBuy    = 3;
    rewardsFeeOnBuy      = 2;

    marketingFeeOnSell   = 3;
    rewardsFeeOnSell     = 2;
...
}

uint256 bnbShare = (marketingFeeOnBuy + marketingFeeOnSell) +
(rewardsFeeOnBuy + rewardsFeeOnSell);

if(contractTokenBalance > 0 && bnbShare > 0) {
    .
    if((marketingFeeOnBuy + marketingFeeOnSell) > 0) {
        ....

    if((rewardsFeeOnBuy + rewardsFeeOnSell) > 0) {
         ...
    }

...
```

Recommendation

It is recommended to streamline the contract's fee handling logic by reassessing the necessity of these redundant if conditions. Since the fee rates are hardcoded and unchangeable, the contract could be optimized by removing these checks, thereby reducing gas costs and simplifying the codebase. If there is a potential need for flexible fee structures in the future, consider implementing functions to allow for dynamic fee rate adjustments by authorized parties, ensuring such changes are transparent and secure. This approach would not only enhance the contract's functionality and adaptability but also maintain clarity in the contract's operational logic, ultimately benefiting all stakeholders involved.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	Bogdanoff.sol#L108
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Bogdanoff.sol
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 to 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 gas consumption unnecessarily.

```
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.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	,Bogdanoff.sol#L768
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
claimWait = 3600;
...
require(newClaimWait >= 3_600 && newClaimWait <= 86_400,
"claimWait must be updated to between 1 and 24 hours");
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Bogdanoff.sol#L959
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public gasForProcessing = 300_000
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Bogdanoff.sol#L286,287,304,324,617,664,668,672,676,748,782
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
uint256 constant internal magnitude = 2**128
address _owner
uint256 _newMinimumBalance
address _account
```

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	Bogdanoff.sol#L155
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	Bogdanoff.sol#L771,1202
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.

```
lastProcessedIndex = index
swapTokensAtAmount = newAmount
```

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	Bogdanoff.sol#L182,681,1038
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function _transfer(address from, address to, uint256 value) internal
virtual override {
...
        magnifiedDividendCorrections[from] =
magnifiedDividendCorrections[from].add(_magCorrection);
        magnifiedDividendCorrections[to] =
magnifiedDividendCorrections[to].sub(_magCorrection);
    }

function isContract(address account) internal view returns (bool) {
        return account.code.length > 0;
    }
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Bogdanoff.sol#L1095,1170
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.

```
uint256 burnTokens
uint256 claims
uint256 lastProcessedIndex
uint256 iterations
```

Recommendation

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

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Bogdanoff.sol#L626,664,668,672,676
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
address _owner
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Bogdanoff.sol#L627
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.

```
rewardToken = _rewardToken
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Bogdanoff.sol#L1035
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
ERC20token.transfer(msg.sender, balance)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		

	mod	Internal		
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IterableMapping	Library			
	get	Public		-
	getIndexOfKey	Public		-
	getKeyAtIndex	Public		-
	size	Public		-
	set	Public	✓	-
	remove	Public	✓	-
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	✓	-
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	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-

	allowance	External		-
	transfer	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	✓	-

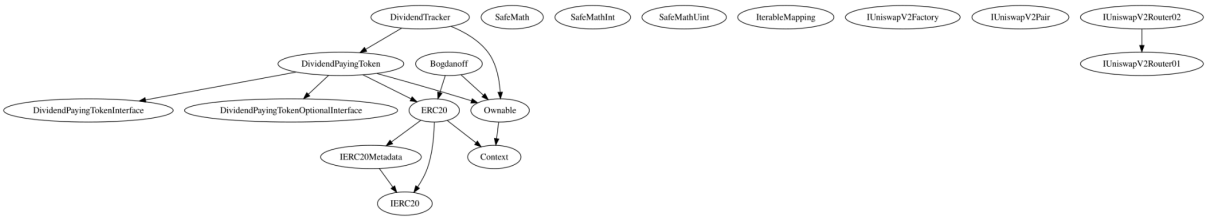
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	withdrawDividend	External	✓	-
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
	distributeDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-

	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
DividendTracker	Implementation	Ownable, DividendPayingToken		
		Public	✓	DividendPayingToken
	_transfer	Internal		
	withdrawDividend	Public		-
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	setLastProcessedIndex	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-

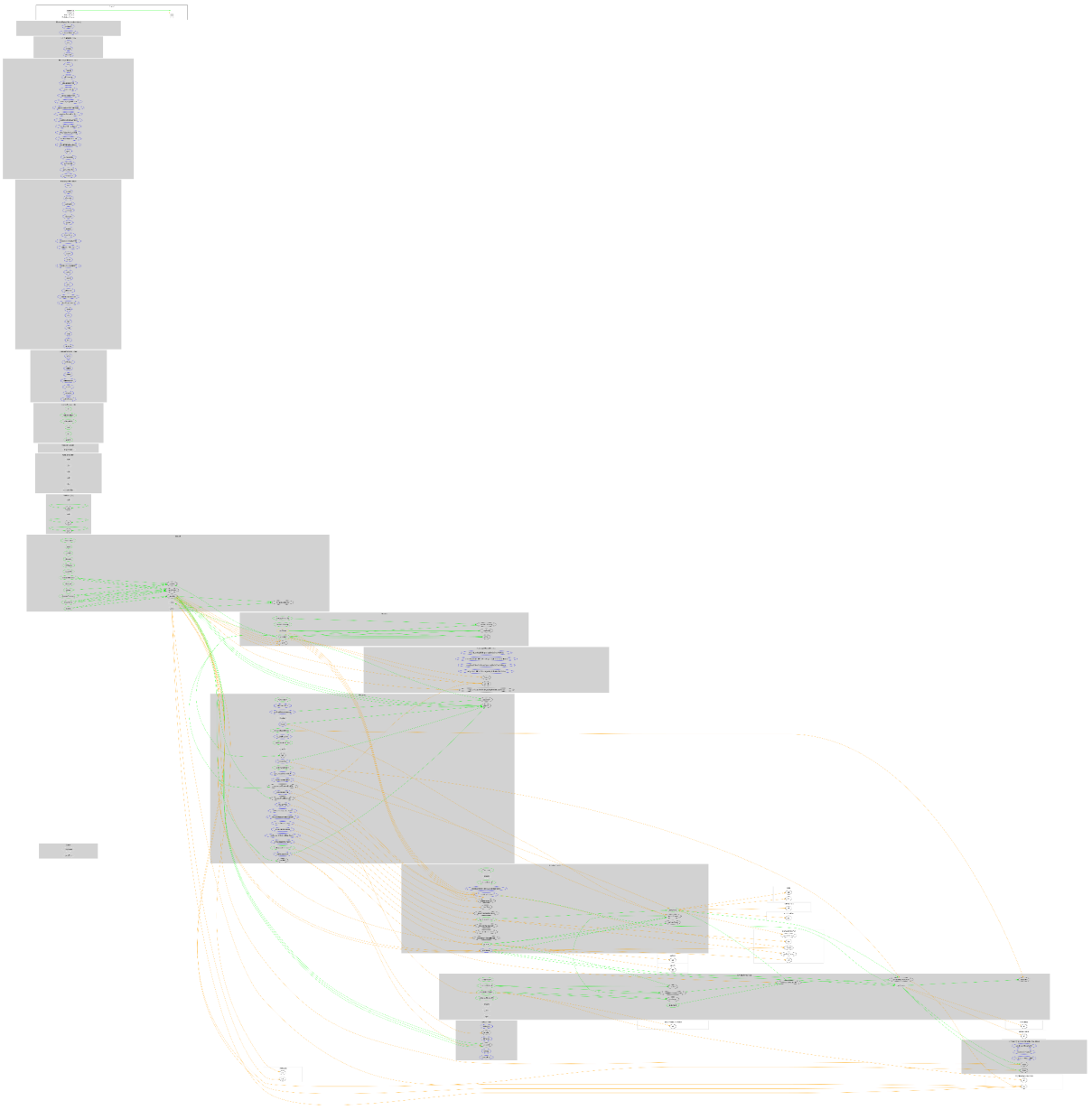
	canAutoClaim	Private		
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
Bogdanoff	Implementation	ERC20, Ownable		
		Public	Payable	ERC20
		External	Payable	-
	claimStuckTokens	External	✓	onlyOwner
	isContract	Internal		
	sendBNB	Internal	✓	
	_setAutomatedMarketMakerPair	Private	✓	
	excludeFromFees	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapAndSendDividends	Private	✓	
	setSwapTokensAtAmount	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	totalRewardsEarned	Public		-

	excludeFromDividends	External	✓	onlyOwner
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	claimAddress	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	setLastProcessedIndex	External	✓	onlyOwner
	getNumberOfDividendTokenHolders	External		-

Inheritance Graph



Flow Graph



Summary

Bogdanoff contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Bogdanoff is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. The fees are set to 5% for buy and sell transactions.

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