Cyberscope

A TAC Security Company

Audit Report

Baby Grok

July 2025

Network BSC

Address 0x3303113001c51769f2753C2aFb7B5a6d0535660E

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
Medium	PLT	Potential Locked Tokens	Unresolved
Minor	CR	Code Repetition	Unresolved
Minor	CCR	Contract Centralization Risk	Unresolved
Minor	DDP	Decimal Division Precision	Unresolved
Minor	MEE	Missing Events Emission	Unresolved
Minor	PDO	Potential Downcasting Overflows	Unresolved
Minor	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
Minor	RED	Redundant Event Declaration	Unresolved
Minor	RSP	Redundant Struct Property	Unresolved
Minor	RSD	Redundant Swap Duplication	Unresolved
Minor	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor	L07	Missing Events Arithmetic	Unresolved
Minor	L14	Uninitialized Variables in Local Scope	Unresolved
Minor	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	BabyGrok
Compiler Version	v0.8.26+commit.8a97fa7a
Optimization	200 runs
Explorer	https://bscscan.com/address/0x3303113001c51769f2753c2afb7b5a6d0535660e
Address	0x3303113001c51769f2753c2afb7b5a6d0535660e
Network	BSC
Symbol	BabyGrok
Decimals	9
Total Supply	420,000,000,000,000
Badge Eligibility	Yes

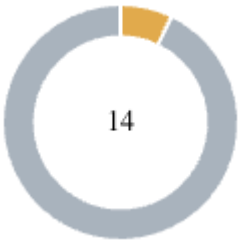
Audit Updates

Initial Audit	25 Jul 2025

Source Files

Filename	SHA256
BabyGrok.sol	6845668f934a132ba0f8d66eff428c63c66afe444caa91c741c004749cbc48c3

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	13	0	0	0

PLT - Potential Locked Tokens

Criticality	Medium
Location	BabyGrok.sol#L791,841,842
Status	Unresolved

Description

In the token swapping mechanism, the contract imposes a cap on the maximum amount of tokens that can be swapped in a single transaction. However, after the swap is completed, the accumulated tax tracking variables are fully reset. This logic introduces a discrepancy: if the contract balance exceeds `swapTokensAtAmt * 4`, only a portion of the tokens are actually swapped, but the tax counters are completely reset. This results in an inaccurate reflection of the remaining token balance and may cause unaccounted or permanently locked tokens within the contract.

```
if(contractBalance > swapTokensAtAmt * 4){
    contractBalance = swapTokensAtAmt * 4;
}
...
tokensForTaxMem.tokensForLiquidity = 0;
tokensForTaxMem.tokensForMarketing = 0;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the contract does not reset the tax variables unconditionally. Instead, it should only reset them in proportion to the tokens actually swapped.

CR - Code Repetition

Criticality	Minor / Informative
Location	BabyGrok.sol#L753
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
if (isAMMPair[from]){
    tax = uint128(amount * 2000 / FEE_DIVISOR);
} else if (isAMMPair[to]) {
    tax = uint128(amount * 2000 / FEE_DIVISOR);
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	BabyGrok.sol#L878
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the contract uses more than one admin addresses (`devAddress`). While this design allows flexibility, it introduces centralization risks, such as:

- A compromised or malicious dev/owner can execute privileged functions.
- Users must trust that the dev and owner will not abuse their roles.
- Multiple admin roles increase the attack surface.
- Without transparency or on-chain governance, decisions can be arbitrary or non-community aligned.

```
require(msg.sender == devAddress,"Not dev");
...
function updateDevAddress(address _address) external onlyOwner {
    require(_address != address(0), "zero address");
    devAddress = _address;
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	BabyGrok.sol#L758,759
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.

```
tokensForTaxUpdate.tokensForLiquidity += uint80(tax * taxes.liquidityTax /
taxes.totalTax / 1e9);
tokensForTaxUpdate.tokensForMarketing += uint80(tax * taxes.marketingTax /
taxes.totalTax / 1e9);
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	BabyGrok.sol#L886,891,897,898,899,905,906,907
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
marketingAddress = _address;
devAddress = _address;
buyTax.marketingTax = _marketingTax;
buyTax.liquidityTax = _liquidityTax;
buyTax.totalTax = totalTax;
sellTax.marketingTax = _marketingTax;
sellTax.liquidityTax = _liquidityTax;
sellTax.totalTax = totalTax;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PDO - Potential Downcasting Overflows

Criticality	Minor / Informative
Location	BabyGrok.sol#L737,751,753,758,759
Status	Unresolved

Description

As part of the transfer flow, the contract performs type downcasting from `uint256` to smaller unsigned integer types such as `uint128` and `uint80` during tax calculations. These downcasts do not include overflow checks, and the resulting values might exceed the maximum capacity of the target types.

```
tax = uint128(amount * taxes.totalTax / FEE_DIVISOR);
tokensForTaxUpdate.tokensForLiquidity += uint80(tax * taxes.liquidityTax /
taxes.totalTax / 1e9);
tokensForTaxUpdate.tokensForMarketing += uint80(tax * taxes.marketingTax /
taxes.totalTax / 1e9);
```

Recommendation

The team is advised to avoid unnecessary downcasting by using `uint256` for all tax-related calculations and storage unless there is a compelling reason to optimize for gas by using smaller types. If smaller types are required (e.g., for tight struct packing), the team could add explicit checks to ensure that values do not exceed the maximum allowed by the target type before downcasting.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	BabyGrok.sol#L773
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
dexRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
    tokenAmt,
    0,
    path,
    address(this),
    block.timestamp
);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	BabyGrok.sol#L667,668,670
Status	Unresolved

Description

There are 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.

The contract declares certain events in its code. However, these events 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 UpdatedTransactionLimit(uint newMax);
event UpdatedWalletLimit(uint newMax);
event SetExemptFromLimits(address _address, bool _isExempt);
```

Recommendation

The team is advised to take these segments into consideration 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.

A recommended approach would be to either remove the declared events that are not being emitted or to incorporate the necessary emit statements within the contract's functions to actually emit these events when relevant actions occur.

RSP - Redundant Struct Property

Criticality	Minor / Informative
Location	BabyGrok.sol#L697
Status	Unresolved

Description

There are 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.

The contract defines specific properties within structs. However, these properties lack meaningful utilization in the codebase. Consequently, these properties are deemed redundant.

```
tokensForTax.gasSaver = true;
```

Recommendation

The team is advised to take these segments into consideration 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.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	BabyGrok.sol#L799,816
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
swapTokensForETH(firstmath);
swapTokensForETH(contractBalance);
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	BabyGrok.sol#L603,640,849,877,884,889,894,902
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);
address public immutable WETH
address _address
bool _isExempt
address _token
uint64 _liquidityTax
uint64 _marketingTax
```

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/stable/style-guide.html#naming-conventions.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	BabyGrok.sol#L860
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.

```
swapTokensAtAmt = 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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	BabyGrok.sol#L739
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.

```
Taxes memory taxes
```

Recommendation

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

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	BabyGrok.sol#L557
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
```

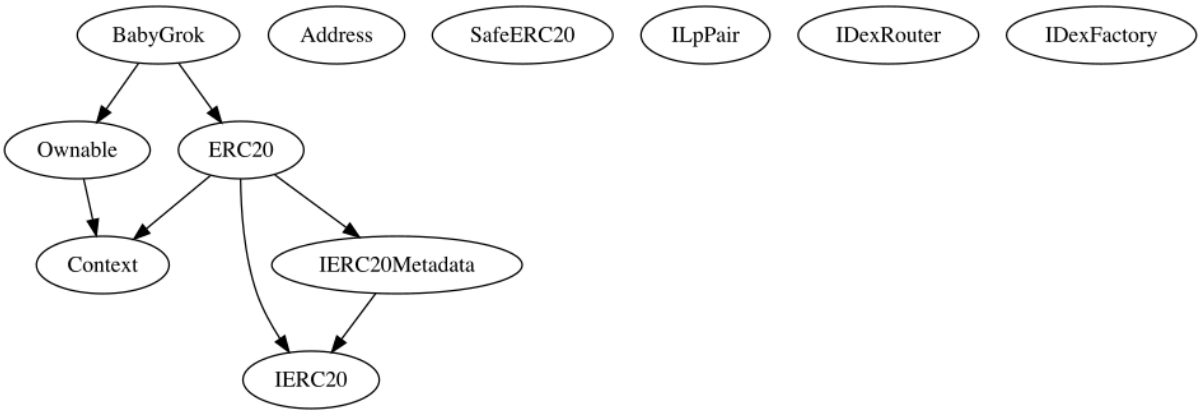
Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

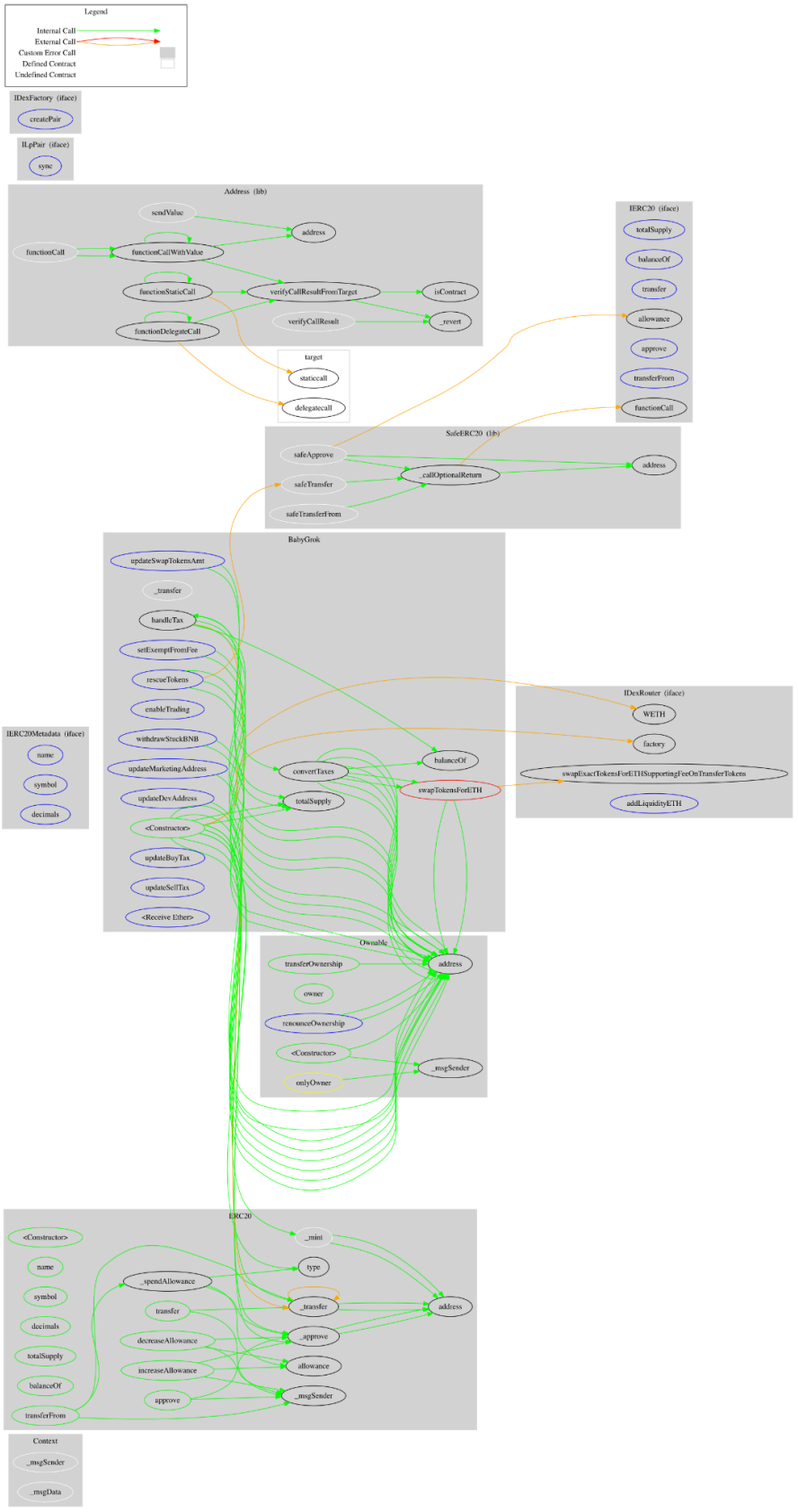
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
BabyGrok	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	_transfer	Internal	✓	
	handleTax	Internal	✓	
	swapTokensForETH	Private	✓	
	convertTaxes	Private	✓	
	setExemptFromFee	External	✓	onlyOwner
	updateSwapTokensAmt	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	withdrawStuckBNB	External	✓	-
	rescueTokens	External	✓	-
	updateMarketingAddress	External	✓	onlyOwner
	updateDevAddress	External	✓	onlyOwner
	updateBuyTax	External	✓	onlyOwner
	updateSellTax	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IDexFactory (iface)
createPair
IUniswapV2Pair (iface)
sync
Address (lib)
sendValue
address
functionCall
functionCallWithValue
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
verifyCallResult
_revert
target
staticcall
delegatecall
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
functionCall
SafeERC20 (lib)
safeApprove
safeTransfer
safeTransferFrom
_callOptionalReturn
address
BabyGrok
updateSwapTokensAmt
_transfer
handleTax
setExemptFromFee
rescueTokens
enableTrading
withdrawStuckBNB
updateMarketingAddress
updateDevAddress
<Constructor>
updateBuyTax
updateSellTax
<Receive Ether>
convertTaxes
totalSupply
balanceOf
swapTokensForETH
IDexRouter (iface)
WETH
factory
swapExactTokensForETHSupportingFeeOnTransferTokens
addLiquidityETH
IERC20Metadata (iface)
name
symbol
decimals
Ownable
transferOwnership
address
owner
renounceOwnership
<Constructor>
_msgSender
onlyOwner
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
_spendAllowance
transfer
decreaseAllowance
increaseAllowance
approve
_mint
type
_transfer
address
_approve
allowance
_msgSender
Context
_msgSender
_msgData

Summary

Baby Grok contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 20% buy and sell fees.

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 TAC 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.

A **TAC Security** Company

The Cyberscope team

cyberscope.io