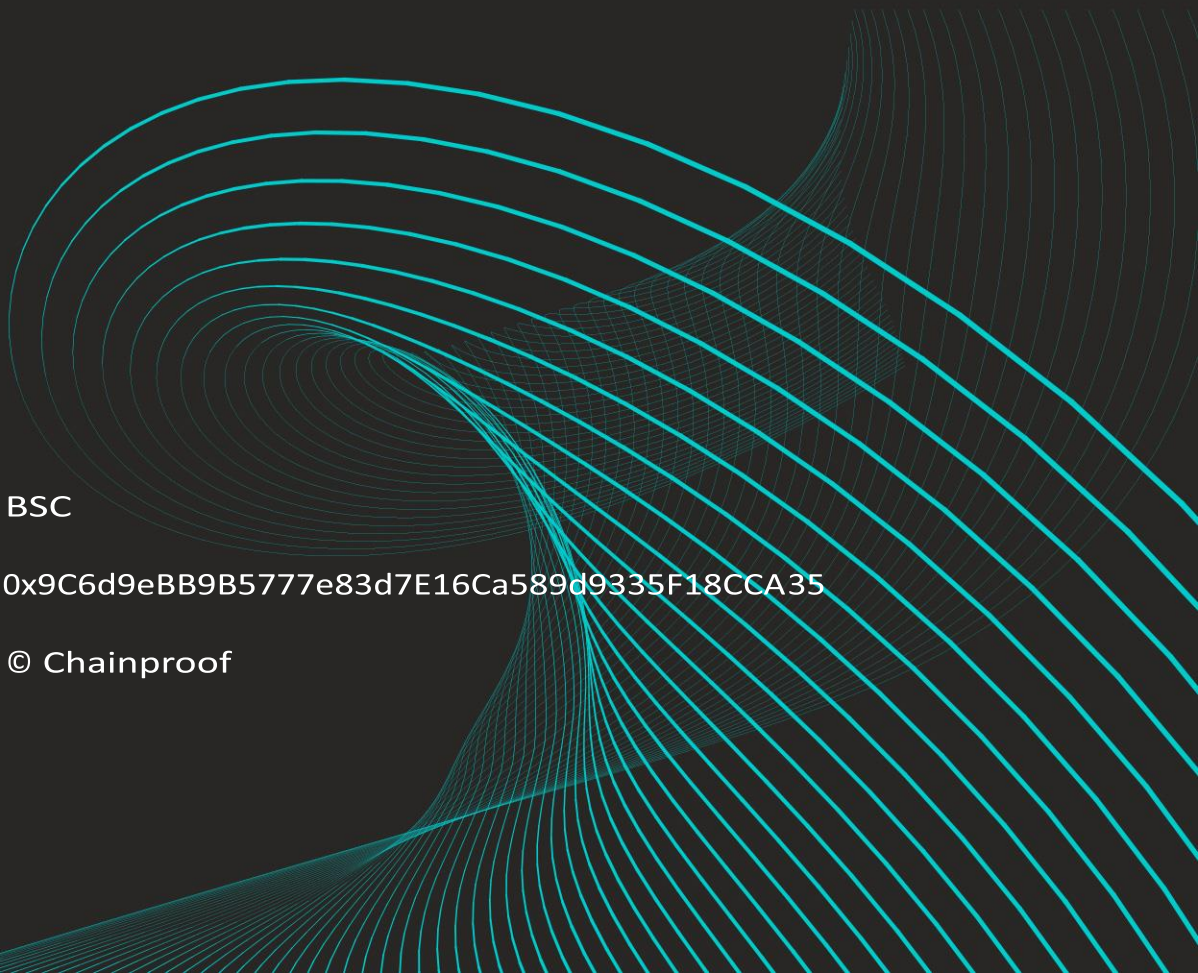
CHAINPROOF

Audit Report

Babyxrp

December 2024



Network BSC

Address 0x9C6d9eBB9B5777e83d7E16Ca589d9335F18CCA35

Audited by © Chainproof

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
●	RSML	Redundant SafeMath Library	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved

●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Chainproof'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.

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

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.

Review

Contract Name	BABYTOKEN
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://bscscan.com/address/0x9c6d9ebb9b5777e83d7e16ca589d9335f18cca35
Address	0x9c6d9ebb9b5777e83d7e16ca589d9335f18cca35
Network	BSC
Symbol	BABYXRP
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Yes

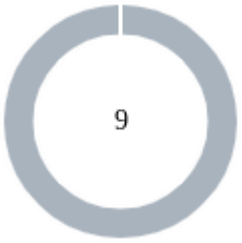
Audit Updates

Initial Audit	16 Dec 2024

Source Files

Filename	SHA256
BABYTOKEN.sol	7cc9a1f0b27a3f70d4c2c376cadad1f2168d797e61d53c90abd884cd4e36df62

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	BABYTOKEN.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 in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	BABYTOKEN.sol#L2467,2576,2678

Status	Unresolved

Description

The contract is missing error messages. Specifically, there 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	BABYTOKEN.sol#L3009,3043
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.

```
rewardToken
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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	BABYTOKEN.sol#L1280,1664,1668,1677,1735,1740,2042,2074,2079,2123,2146,2147,2164,2436,2456,2457,2458,2459,2519,2526,2538,2552,2723,2960
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 __Context_init() internal initializer {
        __Context_init_unchained();
    }
function __Context_init_unchained() internal initializer {
    }
uint256[50] private __gap

function __ERC20_init(string memory name_, string memory symbol_)
internal initializer {
    __Context_init_unchained();
      __ERC20_init_unchained(name_,
                             symbol_);
    }
...
```

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	BABYTOKEN.sol#L3074,3108,3113,3119
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.

```
swapTokensAtAmount = amount totalFees =
tokenRewardsFee.add(liquidityFee).add(marketingFee)
liquidityFee = value marketingFee = value
```

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	BABYTOKEN.sol#L416,1664,1907,2571
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);
        uint256 accountBalance = _balances[account];
...
        }
         _totalSupply -=
                 amount;
      emit Transfer(account,
        address(0), amount);
 _afterTokenTransfer(account,
         address(0), amount);
    }
...
```


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.

-

L15 Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	BABYTOKEN.sol#L2458,2459,2519,2526,2538,2552
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	BABYTOKEN.sol#L3043,3064
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.

```
uniswapV2Pair = _uniswapV2Pair
payable(serviceFeeReceiver_).transfer(serviceFee_)
```

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.

L17 Usage of Solidity Assembly

Criticality	Minor / Informative
Location	BABYTOKEN.sol#L532,707,1148,1166,1184
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 {
            size := extcodesize(account)
        }
assembly {
                    let returndata_size := mload(returndata)
...
assembly {
            let ptr := mload(0x40)
            mstore(ptr,
0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000)
            mstore(add(ptr, 0x14), shl(0x60, implementation))
            mstore(add(ptr, 0x28),
0x5af43d82803e903d91602b57fd5bf300000000000000000000000000000000)
            instance := create(0, ptr, 0x37)
        }
...
```

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

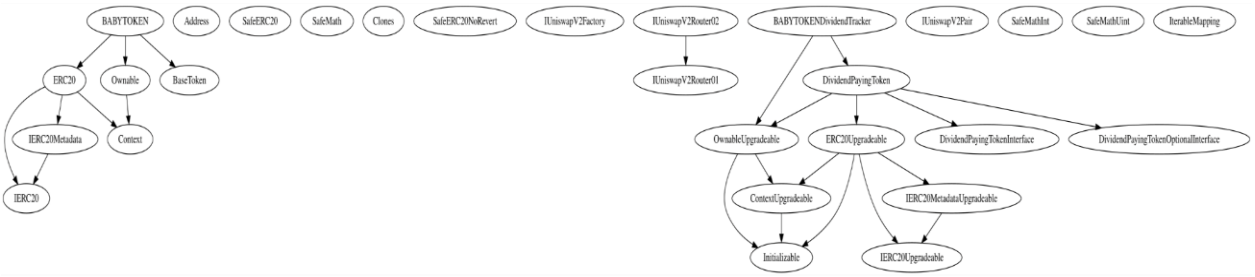
BABYTOKEN	Implementation	ERC20, Ownable, BaseToken		
		Public	Payable	ERC20
		External	Payable	-
	setSwapTokensAtAmount	External	✓	onlyOwner
	excludeFromFees	External	✓	onlyOwner
	excludeMultipleAccountsFromFees	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setTokenRewardsFee	External	✓	onlyOwner
	setLiquiditFee	External	✓	onlyOwner
	setMarketingFee	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-

-

	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	getMinimumTokenBalanceForDividends	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-

	dividendTokenBalanceOf	Public		-
	excludeFromDividends	External	✓	onlyOwner
	isExcludedFromDividends	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	_transfer	Internal	✓	
	swapAndSendToFee	Private	✓	
	swapAndLiquify	Private	✓	
	swapTokensForEth	Private	✓	
	swapTokensForCake	Private	✓	
	addLiquidity	Private	✓	
	swapAndSendDividends	Private	✓	

Inheritance Graph



Flow Graph

Summary

Babyxrp contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Babyxrp 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.

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 Chainproof’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 Chainproof 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 Chainproof’s position is that each company and individual are responsible for their own due diligence and continuous security Chainproof’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 Chainproof 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 ChainProof

Chainproof is an Audit & KYC firm for Blockchain Projects, aimed at securing the Blockchain and the assets at risk. Chainproof is fueled by Industry grade experienced Blockchain Developers from all around the globe. From finding vulnerabilities, potential scams, malicious code mitigation, improper implementation of the token which can lead to loss of user's fund, you name it and we cover and secure them all.

Security testing and risk mitigation is given the highest priority at ChainProof. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.
ChainProof is aiming to make crypto discoverable and efficient globally. We associate with extremely robust testing and code review, leaving no room for any security risks because, when it comes to user's funds, we need to leave no stone unturned. Cheers!

The ChainProof team

ChainProof.dev