



Cyberscope

A *TAC Security* Company

Audit Report

Babyphoenix

August 2025

Network BSC

Address 0xa3ea8983e4d9c237fe8b44ac1e38c6a000efd530

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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MRF	Missing Renounce Functionality	Unresolved
●	CR	Code Repetition	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Risk Classification	5
Review	6
Audit Updates	6
Source Files	6
Findings Breakdown	7
ELFM - Exceeds Fees Limit	8
Description	8
Recommendation	8
MRF - Missing Renounce Functionality	9
Description	9
Recommendation	9
CR - Code Repetition	10
Description	10
Recommendation	10
CCR - Contract Centralization Risk	11
Description	11
Recommendation	11
IDI - Immutable Declaration Improvement	12
Description	12
Recommendation	12
MEE - Missing Events Emission	13
Description	13
Recommendation	13
L02 - State Variables could be Declared Constant	14
Description	14
Recommendation	14
L04 - Conformance to Solidity Naming Conventions	15
Description	15
Recommendation	15
L07 - Missing Events Arithmetic	16
Description	16
Recommendation	16
L19 - Stable Compiler Version	17
Description	17
Recommendation	17
Functions Analysis	18

Flow Graph	19
Summary	20
Disclaimer	21
About Cyberscope	22

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	BabyPhoenix
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://bscscan.com/address/0xa3ea8983e4d9c237fe8b44ac1e38c6a000efd530
Address	0xa3ea8983e4d9c237fe8b44ac1e38c6a000efd530
Network	BSC
Symbol	BPX
Decimals	9
Total Supply	400,000,000,000,000,000
Badge Eligibility	Must Fix Criticals

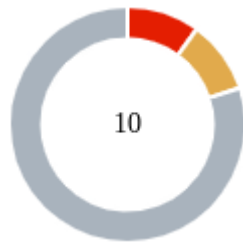
Audit Updates

Initial Audit	24 Aug 2025
----------------------	-------------

Source Files

Filename	SHA256
BabyPhoenix.sol	1bca2da6f95c627ca867d4cddf24941564ac5c5b052e6deaad30c9934701af42

Findings Breakdown



● Critical	1
● Medium	1
● Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	1	0	0	0
● Medium	1	0	0	0
● Minor / Informative	8	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	BabyPhoenix.sol#L36
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setTaxFee` function with a high percentage value and by setting `presaleActive = false`.

```
function setTaxFee(uint256 _fee) external onlyOwner {  
    taxFee = _fee;  
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

MRF - Missing Renounce Functionality

Criticality	Medium
Location	BabyPhoenix.sol
Status	Unresolved

Description

The contract employs the `onlyOwner` modifier to restrict specific administrative functions to the `_taxWallet` owner. As noted in finding `ELFM`, this owner has the authority to set high tax rates on transactions. Furthermore, the contract lacks a function to renounce ownership, which means this control remains centralized. This centralization increases the risk of potential misuse by the owner, as outlined in the finding.

```
modifier onlyOwner() {  
  require(msg.sender == taxWallet, "Only owner");  
  _;  
}
```

Recommendation

The team is advised to revise the current implementation of the contract to ensure that the renounce functionality is properly implemented. This will enhance user trust in the contract and improve overall consistency.

CR - Code Repetition

Criticality	Minor / Informative
Location	BabyPhoenix.sol#L40,62
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.

```
function transfer(address to, uint256 value) public returns (bool success) {  
    ...  
}  
  
function transferFrom(address from, address to, uint256 value) public returns  
(bool success) {  
    ...  
}
```

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	BabyPhoenix.sol#L32,36
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.

```
function setPresaleActive(bool _active) external onlyOwner {
    presaleActive = _active;
}

function setTaxFee(uint256 _fee) external onlyOwner {
    taxFee = _fee;
}
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	BabyPhoenix.sol#L23
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.

```
taxWallet
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
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.

```
function setPresaleActive(bool _active) external onlyOwner {
    presaleActive = _active;
}

function setTaxFee(uint256 _fee) external onlyOwner {
    taxFee = _fee;
}
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BabyPhoenix.sol#L5,6,7
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.

```
string public name = "BabyPhoenix"  
string public symbol = "BPX"  
uint8 public decimals = 9
```

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	BabyPhoenix.sol#L32,36
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.

```
bool _active
uint256 _fee
```

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	BabyPhoenix.sol#L37
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.

```
taxFee = _fee
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BabyPhoenix.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.20;
```

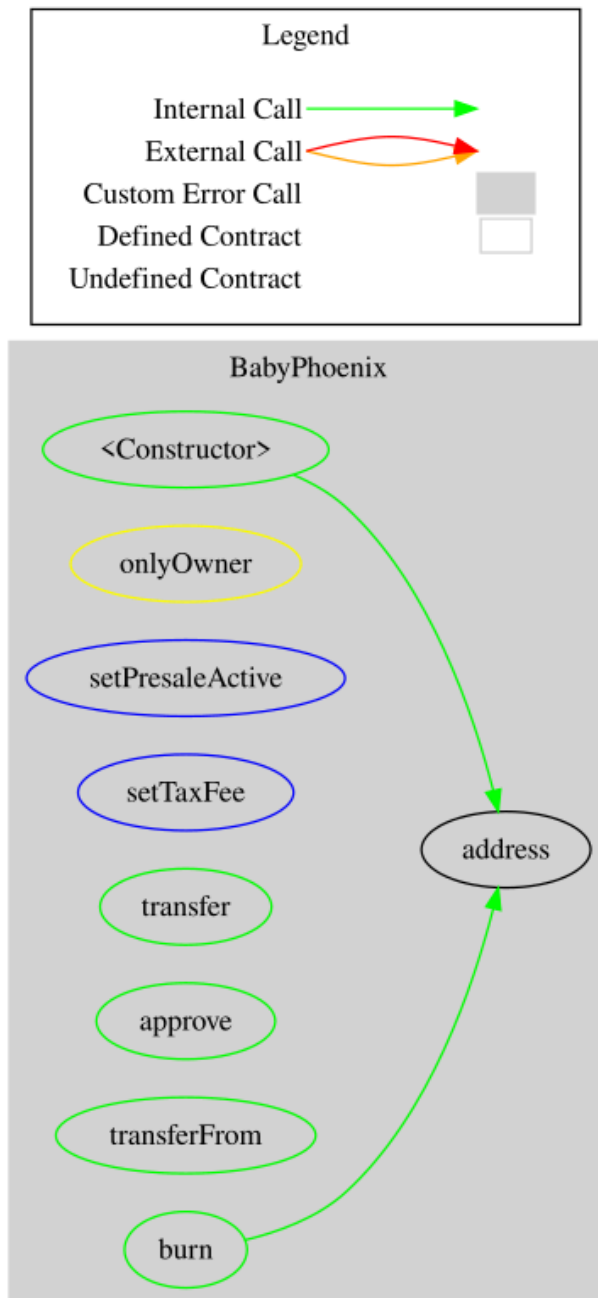
Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
BabyPhoenix	Implementation			
		Public	✓	-
	setPresaleActive	External	✓	onlyOwner
	setTaxFee	External	✓	onlyOwner
	transfer	Public	✓	-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	burn	Public	✓	-

Flow Graph



Summary

Babyphoenix contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like manipulating the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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