



Smart Contract Audit

FOR

BabyWif

DATED : 16 March, 2024



AUDIT SUMMARY

Project name – BabyWif

Date: 16 March, 2024

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: Passed

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	0	0	0	1
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:

A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :

The code has undergone static analysis using Slither.

Testnet version:

The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:

<https://testnet.bscscan.com/address/0xb62D0C9dCA37fF5585AcB05F73B9F15C8429cfDA#code>



Token Information

Token Name : BabyWif

Token Symbol: BabyWif

Decimals: 18

Token Supply: 4200000000000

Network: BscScan

Token Type: BEP-20

Token Address:

0x0b655Bfee868Acfa1E282404148b06b92cb3DfFE

Checksum:

Ae1c3a4fbb6e83e8393a57617b5a112

Owner: --

(at time of writing the audit)

Deployer:

0x2D9a5e1217a3F5cAbD0348d81a05dB1ba361EA47



TOKEN OVERVIEW

Fees:

Buy Fee: 0-0%

Sell Fee: 0-0%

Transfer Fee: 0-0%

Fees Privilege: Owner

Ownership: Owned

Minting: No mint function

Max Tx Amount/ Max Wallet Amount: No

Blacklist: No

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
 - Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
 - Test coverage analysis determines whether the test cases are covering the code and how much code is exercised when we run the test cases.
 - Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
 - Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.
-



VULNERABILITY CHECKLIST

- | | |
|------------------------------------|-------------------------------|
| ✓ Return values of low-level calls | ✓ Gasless Send |
| ✓ Private modifier | ✓ Using block.timestamp |
| ✓ Multiple Sends | ✓ Re-entrancy |
| ✓ Using Suicide | ✓ Tautology or contradiction |
| ✓ Gas Limitand Loops | ✓ Timestamp Dependence |
| ✓ Address hardcoded | ✓ Revert/require functions |
| ✓ Exception Disorder | ✓ Use of tx.origin |
| ✓ Using inline assembly | ✓ Integer overflow/underflow |
| ✓ Divide before multiply | ✓ Dangerous strict equalities |
| ✓ Missing Zero Address Validation | ✓ Using SHA3 |
| ✓ Compiler version not fixed | ✓ Using throw |
-



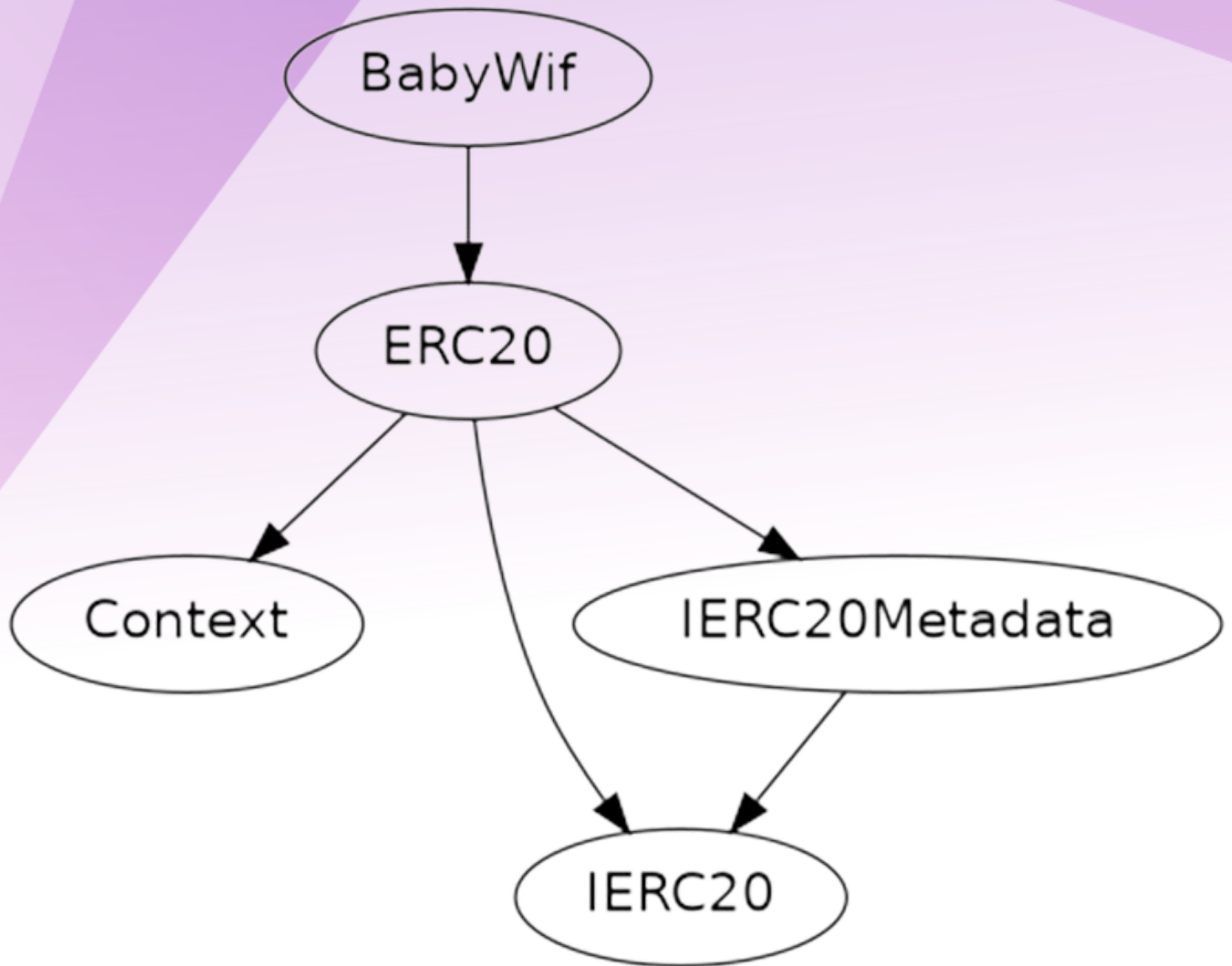
STATIC ANALYSIS

A static analysis of the code was performed using Slither.

No issues were found.

```
INFO:Detectors:
Different versions of Solidity are used:
  - Version used: ['^0.8.15', '^0.8.17']
  - ^0.8.15 (BabyWif.sol#163)
  - ^0.8.17 (BabyWif.sol#517)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Context._msgData() (BabyWif.sol#15-17) is never used and should be removed
ERC20._burn(address,uint256) (BabyWif.sol#434-449) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.15 (BabyWif.sol#163) allows old versions
Pragma version^0.8.17 (BabyWif.sol#517) allows old versions
solc-0.8.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
BabyWif.constructor() (BabyWif.sol#519-521) uses literals with too many digits:
  - _mint(msg.sender,420000000000 * 10 ** decimals()) (BabyWif.sol#520)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:BabyWif.sol analyzed (5 contracts with 93 detectors), 7 result(s) found
```


INHERITANCE TREE





Functional Tests

1- Approve (passed):

<https://testnet.bscscan.com/tx/0xe16aff11796c65e2db11b203c4b82ebfd13b1b6d746b9160403105a355ea95f5>

2- Increase Allowance (passed):

<https://testnet.bscscan.com/tx/0x1d060419d75db8cb09782f7619380a08fb60987fb005c7d51d0e1682658a876e>

3- Decrease Allowance (passed):

<https://testnet.bscscan.com/tx/0x4c7f77412cc62d78a0b3c9c0ba4a2defbab4cedc1371d294f025ca419592ea82>

CLASSIFICATION OF RISK

Severity

Description

◆ Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
◆ High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
◆ Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
◆ Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
◆ Gas Optimization / Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity

Found

◆ Critical	0
◆ High-Risk	0
◆ Medium-Risk	0
◆ Low-Risk	0
◆ Gas Optimization / Suggestions	1

MANUAL TESTING

Optimization

Severity: Optimization

Function: Remove unused code

Status: Open

Overview:

Unused variables are allowed in Solidity, and they do not pose a direct security issue. It is the best practice. though to avoid them.

```
function _msgData() internal view virtual returns (bytes calldata) {
    return msg.data;
}

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];
    require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    unchecked {
        _balances[account] = accountBalance - amount;
    }
    _totalSupply -= amount;

    emit Transfer(account, address(0), amount);

    _afterTokenTransfer(account, address(0), amount);
}
```



DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.



ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.



<https://auditace.tech/>



https://t.me/Audit_Ace



https://twitter.com/auditace_



<https://github.com/Audit-Ace>
