BLOCK SOLUTIONS

Smart Contract Code Review and Security Analysis Report for BABYNFTZ Token Smart Contract

Request Date: 2022-09-03
Completion Date: 2022-09-06
Language: Solidity

Contents

Commission

Audited Project	BABYNFTZ Token Smart Contract
Smart Contract Address	0xf74e16EF420C9Fdc81444EA87976341d34D04512
Creator Address	0x56470707172a19ca729d87586dA3a22D14bC1636
Blockchain Platform	Doge Chain Mainnet

Block Solutions was commissioned by BABYNFTZ Token Smart Contract owners to perform an audit of their main smart contract. The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

BABYNFTZ Properties

Contract Token name	Babynftz
Total supply	469,000,000,000,000 BOWU
Symbol	BOWU
Decimals	9
Router	0x6258c967337d3faf0c2ba3adae5656ba95419d5f
Reward Token	0xb7ddc6414bf4f5515b52d8bdd69973ae205ff101
Pair Address	0x9e7905fd7c5776626e688aca03a1649b28b5550a
Marketing Fee Receiver	0x56470707172a19ca729d87586da3a22d14bc1636
Distributer Address	0x7ddf568c700dd8c6073237311d849676eeb597fa
Cronos Pad Anti Bot	0xaad608899ed545132cb335c4642ff81411a14bdc
Auto Liquidity Pair	0x56470707172a19ca729d87586da3a22d14bc1636
Smart Contract Address	0xf74e16EF420C9Fdc81444EA87976341d34D04512
Creator Address	0x56470707172a19ca729d87586dA3a22D14bC1636
Blockchain Platform	Doge Chain Mainnet

Contract Functions

Executables

i. function approve(address spender, uint256 amount)public override returns (bool)
ii. function approveMax(address spender) external returns (bool)
iii. function authorize(address adr) public onlyOwner
iv. function clearBuybackMultiplier() external authorized
v. function setAutoBuybackSettings(bool _enabled, uint256 _cap,uint256 _amount, uint256 _period) external authorized
vi. function setBuybackMultiplierSettings(uint256 numerator, uint256 denominator,uint256 length) external authorized
vii. function setBuyBacker(address acc, bool add) external authorized
viii. function setDistributorSettings(uint256 gas) external authorized
ix. function setDistributionCriteria(uint256 _minPeriod,uint256 _minDistribution) external authorized
x. function setEnableAntiBot(bool _enable) external authorized
xi. function setFeeReceivers(address _autoLiquidityReceiver,address _marketingFeeReceiver) external authorized
xii. function setFees(uint256 _liquidityFee,uint256 _buybackFee,uint256 _reflectionFee,uint256 _marketingFee,uint256 _feeDenominator) public authorized
xiii. function setIsDividendExempt(address holder, bool exempt) external authorized
xiv. function setIsFeeExempt(address holder, bool exempt) external authorized
xv. function setSwapBackSettings(bool _enabled, uint256 _amount)external authorized
xvi. function setTargetLiquidity(uint256 _target, uint256 _denominator)external authorized
xvii. function transfer(address recipient, uint256 amount)external override returns (bool)
xviii. function transferFrom(address sender,address recipient,uint256 amount) external override returns (bool)
xix. function transferOwnership(address payable adr) public onlyOwner
xx. function triggerZeusBuyback(uint256 amount, bool triggerBuybackMultiplier) external authorized
xxi. function unauthorize(address adr) public onlyOwner

Checklist

Compiler errors.	Passed
Possible delays in data delivery.	Passed
Timestamp dependence.	Passed
Integer Overflow and Underflow.	Passed
Race Conditions and Reentrancy.	Passed
DoS with Revert.	Passed
DoS with block gas limit.	Passed
Methods execution permissions.	Passed
Economy model of the contract.	Passed
Private user data leaks.	Passed
Malicious Events Log.	Passed
Scoping and Declarations.	Passed
Uninitialized storage pointers.	Passed
Arithmetic accuracy.	Passed
Design Logic.	Passed
Impact of the exchange rate.	Passed
Oracle Calls.	Passed
Cross-function race conditions.	Passed
Fallback function security.	Passed

Safe Open Zeppelin contracts and implementation usage.	Passed
Whitepaper-Website-Contract correlation.	Not Checked
Front Running.	Not Checked

Executable Functions

BABYNFTZ Contract

function will transfer token for a specified address. “recipient” is the address to transfer’ to. “amount” is the amount to be transferred. Owner's account must have sufficient balance to transfer.

```
function transfer(address recipient, uint256 amount)external override returns (bool)
{
    return _transferFrom(msg.sender, recipient, amount);
}
```

Transfers ownership of the contract to a new account (`adr`). Can only be called by the authorized address.

```
    function transferOwnership(address payable adr) public onlyOwner {
        owner = adr;
        authorizations[adr] = true;
        emit OwnershipTransferred(adr);
    }

    event OwnershipTransferred(address owner);
}
```

Transfer tokens from the “sender” account to the “recipient” account. The calling account must already have sufficient tokens approved for spending from the “sender” account and “sender” account must have sufficient balance to transfer.” Spender” must have sufficient allowance to transfer.

```
function transferFrom(address sender,address recipient,uint256 amount) external
override returns (bool) {
    if (_allowances[sender][msg.sender] != _totalSupply) {
        _allowances[sender][msg.sender] = _allowances[sender][msg.sender]
            .sub(amount, "Insufficient Allowance");
    }

    return _transferFrom(sender, recipient, amount);
}
```

Approve the passed address to spend the specified number of tokens on behalf of msg. sender. “spender” is the address which will spend the funds. “amount” the number of tokens to be spent.

Beware that changing an allowance with this method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards.https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md recommends that there are no checks for the approval double-spend attack as this should be implemented in user interfaces.

```
  function approve(address spender, uint256 amount)public override returns (bool)
{
      _allowances[msg.sender][spender] = amount;
      emit Approval(msg.sender, spender, amount);
      return true;
  }
```

Approve the passed address to spend the maximum number of tokens on behalf of msg. sender

```
function approveMax(address spender) external returns (bool) {
    return approve(spender, _totalSupply);
}
```

Authorizer address can set the distributer setting by setting up the gas cost.

```
function setDistributorSettings(uint256 gas) external authorized {
    require(gas < 750000, "Gas must be lower than 750000");
    distributorGas = gas;
}
```

Authorizer of this contract set the distributed criteria by setting the minimum period and minimum distribution

```
function setDistributionCriteria( uint256 _minPeriod,uint256 _minDistribution)
 external authorized {
    distributor.setDistributionCriteria(_minPeriod, _minDistribution);
}
```

Authorizer of this contract set the swap setting by enabling the swap and setting the swap threshold value.

```
function setSwapBackSettings(bool _enabled, uint256 _amount)external authorized
{
    swapEnabled = _enabled;
    swapThreshold = _amount;
}
```

Authorizer of this contract set the target liquidity.

```
function setTargetLiquidity(uint256 _target, uint256 _denominator)external
authorized
{
    targetLiquidity = _target;
    targetLiquidityDenominator = _denominator;
}
```

Authorizer of this contract can set the auto liquidity fee receiver address, marketing fee receiver address.

```
function setFeeReceivers(address _autoLiquidityReceiver,
address _marketingFeeReceiver) external authorized {
    autoLiquidityReceiver = _autoLiquidityReceiver;
    marketingFeeReceiver = _marketingFeeReceiver;
}
```

Authorizer of this contract can set the feeses such as liquidity fees, buy back fee , reflection fee, marketing fee and fee denominator value.

```
function setFees(uint256 _liquidityFee,uint256 _buybackFee,uint256 _reflectionFee,
uint256 _marketingFee,uint256 _feeDenominator) public authorized {
    _setFees(
        _liquidityFee,
        _buybackFee,
        _reflectionFee,
        _marketingFee,
        _feeDenominator
    );
}
```

Authorizer of this contract can exempt any address from fee.

```
function setIsFeeExempt(address holder, bool exempt) external authorized {
    isFeeExempt[holder] = exempt;
}
```

Authorizer of this contract can exempt any address from dividend.

```
function setIsDividendExempt(address holder, bool exempt) external authorized
{
    require(holder != address(this) && holder != pair);
    isDividendExempt[holder] = exempt;
    if (exempt) {
        distributor.setShare(holder, 0);
    } else {
        distributor.setShare(holder, _balances[holder]);
    }
}
```

Authorizer of this contract can set the buy back multiplier setting by adding the numerator, denominator and length values.

```
function setBuybackMultiplierSettings(
    uint256 numerator,
    uint256 denominator,
    uint256 length
) external authorized {
    require(numerator / denominator <= 2 && numerator > denominator);
    buybackMultiplierNumerator = numerator;
    buybackMultiplierDenominator = denominator;
    buybackMultiplierLength = length;
}
```

Authorizer of this contract can clear the buy back multiplier.

```
function clearBuybackMultiplier() external authorized {
    buybackMultiplierTriggeredAt = 0;
}
```

Authorizer of this contract can enable/disable the antibot.

```
function setEnableAntiBot(bool _enable) external authorized {
    enableAntiBot = _enable;
}
```

Authorizer of this contract can set the buyback setting by changing the auto buy back cap, auto buy back accumulator value, auto buyback block period values.

```
function setAutoBuybackSettings(bool _enabled, uint256 _cap,uint256 _amount,
 uint256 _period) external authorized {
    autoBuybackEnabled = _enabled;
    autoBuybackCap = _cap;
    autoBuybackAccumulator = 0;
    autoBuybackAmount = _amount;
    autoBuybackBlockPeriod = _period;
    autoBuybackBlockLast = block.number;
}
```

Authorizer of this contract can trigger the zeus buy back by giving the amount and Boolean value.

```
function triggerZeusBuyback(uint256 amount, bool triggerBuybackMultiplier) external
 authorized
{
    buyTokens(amount, DEAD);
    if (triggerBuybackMultiplier) {
        buybackMultiplierTriggeredAt = block.timestamp;
        emit BuybackMultiplierActive(buybackMultiplierLength);
    }
}
```

Owner of this contract can add addresses to the authorizer list.

```
function authorize(address adr) public onlyOwner {
    authorizations[adr] = true;
}
```

Owner of this contract can add addresses to the unauthorizer list.

```
function unauthorize(address adr) public onlyOwner {
    authorizations[adr] = false;
}
```

Quick Stats:

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	Assert () misuse	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	“Out of Gas” Attack	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: Passed

Executive Summary

According to the standard audit assessment, Customer`s solidity smart contract is Well-secured. Again, it is recommended to perform an Extensive audit assessment to bring a more assured conclusion.

We used various tools like Mythril, Slither and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Quick Stat section.
We found 0 critical, 0 high, 0 medium and 0 low level issues.

Code Quality

The BABYNFTZ Smart Contract protocol consists of one smart contract. It has other inherited contracts like IERC20Extended, Auth, BaseToken. These are compact and well written contracts. Libraries used in BABYNFTZ Smart Contract are part of its logical algorithm. They are smart contracts which contain reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in protocol. The BLOCKSOLUTIONS team has **not** provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.
Overall, the code is not commented. Commenting can provide rich documentation for functions, return variables and more.

Documentation

As mentioned above, it's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic. We were given a BABYNFTZ Smart Contract smart contract code in the form of File.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based

on well-known industry standard open-source projects. And even core code blocks are written well and systematically. This smart contract does not interact with other external smart contracts.

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Conclusion

The Smart Contract code passed the audit. We were given a contract code. And we have used all possible tests based on given objects as files. Since possible test cases can be unlimited for such extensive smart contract protocol, hence we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything. Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in Quick Stat section of the report.
Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract is "Well Secured".

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our

suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally, follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.