QuillAudits

Audit Report
August, 2021

BINAMON

Contents

Scope of Audit

The scope of this audit was to analyze and document the Binamon Energy smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	1	1	3	2
Closed	0	0	0	0

Introduction

During the period of **August 04, 2021 to August 07, 2021** - QuillAudits Team performed a security audit for Binamon Energy smart contracts.

The code for the audit was taken from the following official link:

Note	Date	Commit hash
Version 1	August	https://bscscan.com/address/0xd8813B5Dfa9AbEB693F217Bb905ec99B66FB017C#code

Issues Found – Code Review / Manual Testing

High severity issues

1. Denial of Service in Transactions [Restricted Mode]

```
    modifier launchRestrict(address sender, address recipient, uint256 amount) {
        if (state == State.Locked) {
            require(sender == owner(), "Tokens are locked");
        }
        if (state == State.Restricted) {
            require(amount <= maxRestrictionAmount, "BNRG: amount greater than max limit
in restricted mode");
            require(lastTx[sender].add(60) <= block.timestamp && lastTx[recipient].add(60)
<= block.timestamp, "BMON: only one tx/min in restricted mode");
            lastTx[sender] = block.timestamp;
            lastTx[recipient] = block.timestamp;
        }
        if (state == State.Unlocked) {
            if (isBlacklisted[recipient]) {
                require(lastTx[recipient] + 30 days <= block.timestamp, "BNRG: only one tx/
month in banned mode");
                lastTx[recipient] = block.timestamp;
            } else if (isBlacklisted[sender]) {
                require(lastTx[sender] + 30 days <= block.timestamp, "BNRG: only one tx/
month in banned mode");
                lastTx[sender] = block.timestamp;
            }
        }
        _;
    }
```

Description

The Binamon Team had implemented a modifier to prevent bots and also to limit the number of transactions that a user can do per minute or per month. This feature can have a critical impact in the smart contract and block all the transactions that a user can do. An attacker can exploit this by calling the function transfer of TransferFrom using the address of the Victim as a receiver and an owner with numToken equal to 0, the modifier will be triggered, and the address of the victim will be added to lastTx, when the legit user wants to call the transfer function the modifier will prevent him. A script can be done to block all the addresses each minute, thus launching a denial service attack on the contract.

Remediation

The modifier launchRestrict should only be based on the msg.sender to prevent the caller of the function, not the receiver or the sender.

Status: Acknowledged by the Auditee

Medium severity issues

2. Approve Race

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

Description

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Remediation

```
   function approve(address delegate, uint256 _currentValue ,uint256 numTokens) public
override returns (bool) {
      if(_currentValue == allowed[msg.sender][delegate])
      {
         allowed[msg.sender][delegate] = numTokens;
         emit Approval(msg.sender, delegate, numTokens);
         return true;
      }
      else return false;
   }
```

Status: Acknowledged by the Auditee

Low Severity Issues

3. Missing Address Validation

```
function setStakingContract(address stakingContractAddress_) public onlyOwner {
   stakingContractAddress = stakingContractAddress_;
```

```
function allowBuyingBoosters(address bmonc) public returns (bool) {
   boosterBuyingAllowed[msg.sender] = bmonc;
   return true;
}
```

```
function deliver(address user, uint256 numTokens) public {
    require(msg.sender == stakingContractAddress, "Manual call not allowed");
    balances[user] = balances[user].add(numTokens);
    _totalSupply = _totalSupply.add(numTokens);
    emit Transfer(address(0), user, numTokens);
}
```

Description

Certain functions lack a safety check in the address; the address-type argument should include a zero-address test; otherwise, the contract's functionality may become inaccessible, or tokens may be burned in perpetuity.

Remediation

It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged by the Auditee

4. Lack of MultiSig Transactions

```
function setStakingContract(address stakingContractAddress_) public onlyOwner {
    stakingContractAddress = stakingContractAddress_;
}
```

Description

In the Binamon Energy Contract, QuillAudits team, noticed that the staking contract address can be changed by calling the setStakingContract and the staking contract has the privilege to call the deliver function which can send an infinite number of tokens to users. These situations are often enabled because a single executor role has access to these functions. While sometimes, the developer or owner does not intend to do this malicious act, this risk still exists if the private key stolen since there is nothing preventing the key-holder from calling the setStakingContract.

Remediation

This function will allow the owner to mint an infinite number of tokens; Binamon Team does only uses the onlyOwner modifier to perform critical actions of the tokens. However, these functionalities should be split between multiple based users with multi-signatures wallet for each one.

Status: Acknowledged by the Auditee

5. Usage of BlockimeStamp

```
        require(amount <= maxRestrictionAmount, "BNRG: amount greater than max limit in
restricted mode");
            require(lastTx[sender].add(60) <= block.timestamp && lastTx[recipient].add(60)
<= block.timestamp, "BMON: only one tx/min in restricted mode");
            lastTx[sender] = block.timestamp;
            lastTx[recipient] = block.timestamp;
        }
        if (state == State.Unlocked) {
            if (isBlacklisted[recipient]) {
                require(lastTx[recipient] + 30 days <= block.timestamp, "BNRG: only one tx/
month in banned mode");
                lastTx[recipient] = block.timestamp;
            } else if (isBlacklisted[sender]) {
                require(lastTx[sender] + 30 days <= block.timestamp, "BNRG: only one tx/
month in banned mode");
                lastTx[sender] = block.timestamp;
            }
```

Description

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right; all that is guaranteed is that it is higher than the timestamp of the previous block.

Remediation

You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged by the Auditee

Informational

6. Floating Pragma

pragma solidity >=0.7.0 <0.9.0;

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation
Lock the pragma version and also consider known bugs for the compiler version that is chosen.

Status: Acknowledged by the Auditee

7. Public function that could be declared external

Description
The following public functions that are never called by the contract should be declared external to save gas:

- name()
- symbol()
- decimals()
- totalSupply()
- balanceOf(address)
- transfer(address,uint256)
- approve(address,uint256)
- allowance(address,address)
- transferFrom(address,address,uint256)
- boosterBuyingAllowance(address)
- allowBuyingBoosters(address)
- setStakingContract(address)
- deliver(address,uint256)
- mint(uint256)
- burn(uint256)
- setBotProtection(bool)
- setRestrictionAmount(uint256)
- blacklistAccount(address,bool)

Remediation
Use the external attribute for functions that are not called from the contract.

Status: Acknowledged by the Auditee

Automated Testing

Slither

```
INFO:Detectors:
BNRG.allowance(address,address).owner (binamon.sol#111) shadows:
        - Ownable.owner() (binamon.sol#27-29) (function)
BNRG.transferFrom(address,address,uint256).owner (binamon.sol#115) shadows:
        - Ownable.owner() (binamon.sol#27-29) (function)
BNRG.boosterBuyingAllowance(address).owner (binamon.sol#131) shadows:
        - Ownable.owner() (binamon.sol#27-29) (function)
BNRG.boosterBuyingAllowance(address,address).owner (binamon.sol#135) shadows:
        - Ownable.owner() (binamon.sol#27-29) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Ownable.transferOwnership(address) (binamon.sol#36-39) should emit an event for:
        - _owner = newOwner (binamon.sol#38)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
BNRG.setStakingContract(address).stakingContractAddress_ (binamon.sol#145) lacks a zero-check on :
                - stakingContractAddress = stakingContractAddress_ (binamon.sol#146)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Pragma version>=0.7.0<0.9.0 (binamon.sol#5) is too complex
solc-0.7.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Constant BNRG._name (binamon.sol#44) is not in UPPER_CASE_WITH_UNDERSCORES
Constant BNRG._symbol (binamon.sol#45) is not in UPPER_CASE_WITH_UNDERSCORES
Constant BNRG._decimals (binamon.sol#46) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
BNRG.restrictionLiftTime (binamon.sol#55) is never used in BNRG (binamon.sol#42-210)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
BNRG.restrictionLiftTime (binamon.sol#55) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (binamon.sol#36-39)
name() should be declared external:
        - BNRG.name() (binamon.sol#76-78)
symbol() should be declared external:
        - BNRG.symbol() (binamon.sol#80-82)
decimals() should be declared external:
        - BNRG.decimals() (binamon.sol#84-86)
totalSupply() should be declared external:
        - BNRG.totalSupply() (binamon.sol#88-90)
balanceOf(address) should be declared external:
        - BNRG.balanceOf(address) (binamon.sol#92-94)
transfer(address,uint256) should be declared external:
        - BNRG.transfer(address,uint256) (binamon.sol#96-103)
approve(address,uint256) should be declared external:
        - BNRG.approve(address,uint256) (binamon.sol#105-109)
allowance(address,address) should be declared external:
        - BNRG.allowance(address,address) (binamon.sol#111-113)
transferFrom(address,address,uint256) should be declared external:
        - BNRG.transferFrom(address,address,uint256) (binamon.sol#115-129)
boosterBuyingAllowance(address) should be declared external:
        - BNRG.boosterBuyingAllowance(address) (binamon.sol#131-133)
allowBuyingBoosters(address) should be declared external:
        - BNRG.allowBuyingBoosters(address) (binamon.sol#140-143)
setStakingContract(address) should be declared external:
        - BNRG.setStakingContract(address) (binamon.sol#145-147)
deliver(address,uint256) should be declared external:
        - BNRG.deliver(address,uint256) (binamon.sol#149-154)
mint(uint256) should be declared external:
```

```
        - BNRG.mint(uint256) (binamon.sol#156-161)
burn(uint256) should be declared external:
        - BNRG.burn(uint256) (binamon.sol#163-169)
setBotProtection(bool) should be declared external:
        - BNRG.setBotProtection(bool) (binamon.sol#174-177)
setRestrictionAmount(uint256) should be declared external:
        - BNRG.setRestrictionAmount(uint256) (binamon.sol#179-181)
blacklistAccount(address,bool) should be declared external:
        - BNRG.blacklistAccount(address,bool) (binamon.sol#183-185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:binamon.sol analyzed (4 contracts with 75 detectors), 32 result(s) found
```

Results

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but the denial of service can impact the logic of the contract.

Several issues have been found in the Audit; it is highly recommended to fix them.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Binamon Energy Contract. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Binamon Energy Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

BINAMON

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com