QuillAudits

Audit Report
May, 2022

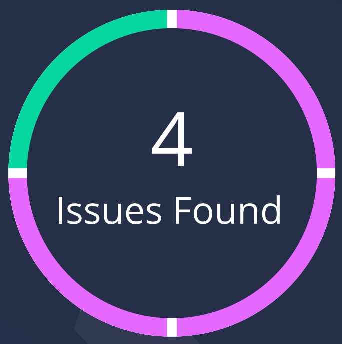
For

METAKILLERS

Table of Content

Executive Summary

Project Name	Metakiller Studio
Overview	MetaKillers is a next generation prison action battle game based on blockchain and knocking between fighters.
Timeline	8th May, 2022 to 10th May, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse 'Metakiller's codebase for quality, security, and correctness.
Contract Address	*0xdf3584186da63f59a44e802e2cc059cb87126c70 | BscScan*



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	**1**	**3**
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

Types of Severities

High

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 be fixed before moving to a live environment.

Medium

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

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 issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open
Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved
These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged
Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved
Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- ✓ 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 leve

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 analysed 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.

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 the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the 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 the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Metakiller

High Severity Issues

No issues were found

Medium Severity Issues

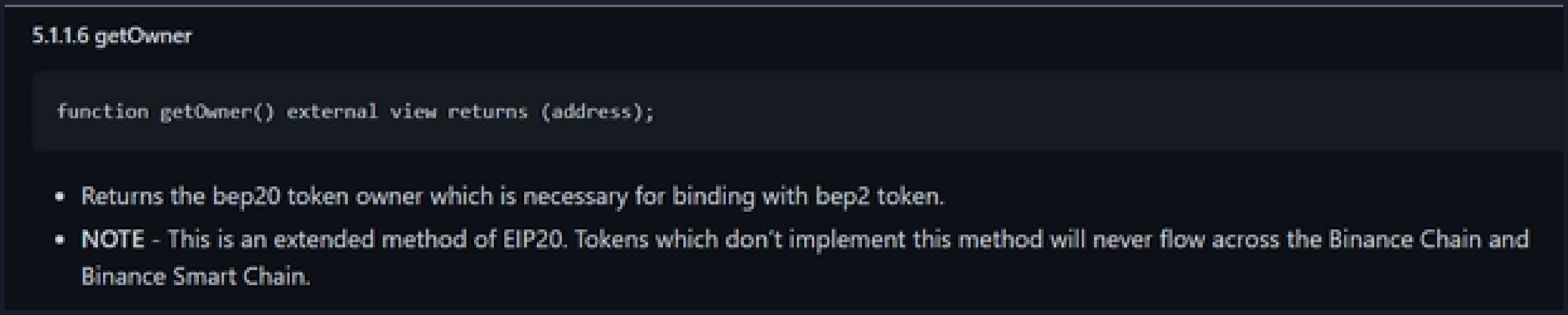
No issues were found

Low Severity Issues

A.1 Not complying with bep20 standard

Description

BEP20 standard makes it mandatory for tokens to define a getOwner() function, which should return the owner of the token contract



5.1.1.6 getOwner

```
function getOwner() external view returns (address);
```

- Returns the bep20 token owner which is necessary for binding with bep2 token.
- **NOTE** - This is an extended method of EIP20. Tokens which don't implement this method will never flow across the Binance Chain and Binance Smart Chain.

However the token contract doesn't define any such function as a result the token may not flow accross Binanace chain and Binanace smart chain, as stated by BEP20 interface documentation

Remediation

Consider using BEP20 implementation with all mandatory methods.

Status

Acknowledged

Informational Issues

A.2 Unlocked pragma

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version get selected while deploying contract which has higher chances of having bugs in it.

Remediation

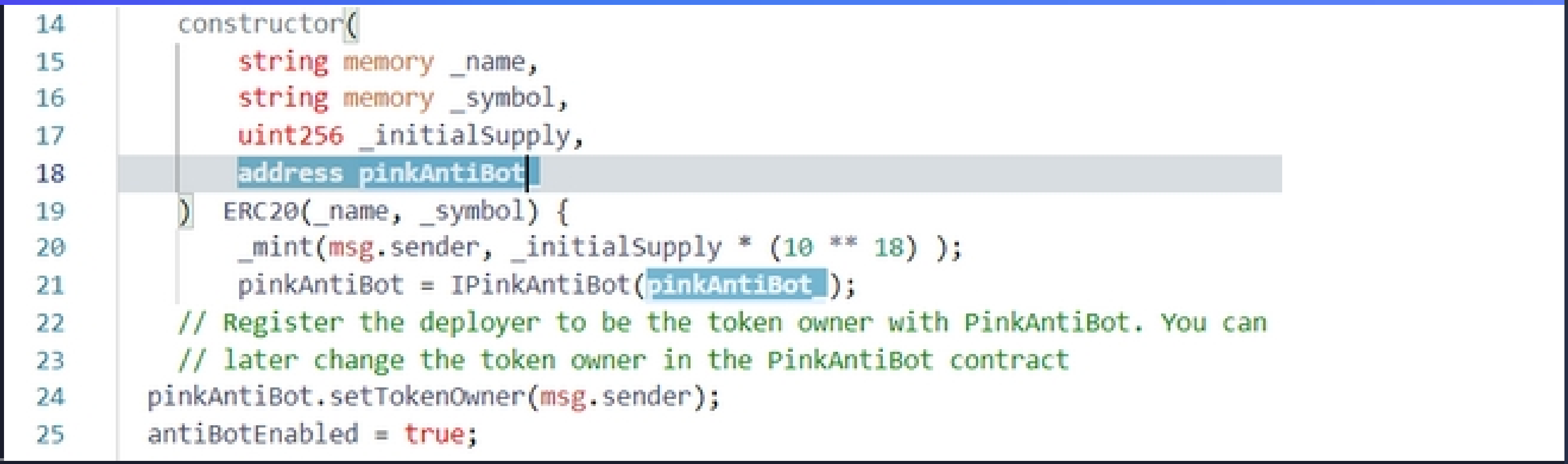
Lock the compiler version that is chosen.

Status

Acknowledged

A.3 Missing Zero Address Validation

Line 18



```
14      constructor(
15          string memory _name,
16          string memory _symbol,
17          uint256 _initialSupply,
18          address pinkAntiBot_
19      ) ERC20(_name, _symbol) {
20          _mint(msg.sender, _initialSupply * (10 ** 18) );
21          pinkAntiBot = IPinkAntiBot(pinkAntiBot_);
22      // Register the deployer to be the token owner with PinkAntiBot. You can
23      // later change the token owner in the PinkAntiBot contract
24    pinkAntiBot.setTokenOwner(msg.sender);
25    antiBotEnabled = true;
```

Description

Constructor lacks zero address validation for pinkAntiBot_ hence is prone to be initialized with zero address.

Remediation

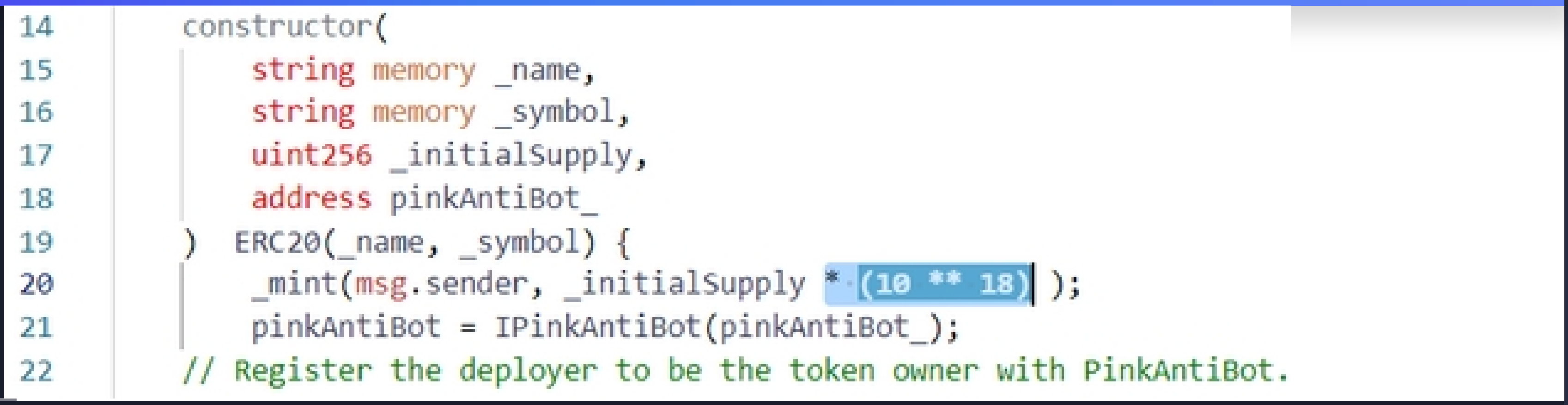
Consider adding zero address checks in order to avoid risks of incorrect address initializations.

Status

Acknowledged

A.4 General Recommendation

Line 20



```
constructor(
    string memory _name,
    string memory _symbol,
    uint256 _initialSupply,
    address pinkAntiBot_
) ERC20(_name, _symbol) {
    _mint(msg.sender, _initialSupply * (10 ** 18) );
    pinkAntiBot = IPinkAntiBot(pinkAntiBot_);
// Register the deployer to be the token owner with PinkAntiBot.
```

_initialSupply is getting multiplied with 10**18 in constructor so in this case make sure to enter _initialSupply without decimals to avoid accidental minting of more tokens than intended.

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below

- ✓ Should mint initial supply to msg.sender on deployment
- ✓ Should update balances of sender and recipient when token transferred
- ✓ Should be able to approve tokens
- ✓ Should be able to increase allowance
- ✓ Should be able to decrease allowance
- ✓ Should be able to spend approved tokens
- ✓ Owner should be able to rescue tokens with clearTokens function
- ✓ Owner should be able to enable and disable anti bot with setEnableAntiBot
- ✓ _beforeTokenTransfer calls onPreTransferCheck when antiBotEnabled set to true
- ✓ Reverts on transfer to zero address
- ✓ Reverts on approve to zero address
- ✓ Reverts if sender doesn’t holds enough token balance for sending
- ✓ Reverts if spender doesn’t hold enough approval to spend someone’s tokens
- ✓ Reverts if owner enters this token address to transfer in clearTokens

Automated Tests

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

In this report, we have considered the security of the Metakiller Studio. We performed our audit according to the procedure described above.

Some issues of Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the end,the Metakiller Team Acknowledged all Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Metakiller Studio Platform. 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 Metakiller Studio Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2022

For

METAKILLERS

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com