



BLOCK AUDIT REPORT

Smart Contract Security Audit Report



Tiger Zilla



BLOCK AUDIT REPORT

Block Audit Report Team received the tigerzilla.sol file for smart contract security audit of the Tigerzilla on January 04, 2021. The following are the details and results of this smart contract security audit:

Project Name: Tigerzilla

The Contract address: 0x5181b4BdBFcA172283112060B9fD734Cfbc2aA02

Link Address:

<https://bscscan.com/address/0x5181b4BdBFcA172283112060B9fD734Cfbc2aA02>

The audit items and results:

(Other undiscovered security vulnerabilities are not included in the audit responsibility scope)

Audit Result: Passed

Audit Number: BAR0017004012022

Audit Date: January 04, 2022

Audit Team: Block Audit Report Team

Table of Content

Introduction	4-5
Auditing Approach and Methodologies applied.....	4
Audit Details	5
Audit Goals	6-7
Security	6
Sound Architecture	6
Code Correctness and Quality	6
Issue Categories	6
High level severity issues.....	6
Medium level severity issues	6
Low level severity issues	6
Issues Checking Status	7
Functions Outline	8-11
Functions Outline	8-11
Manual Audit:.....	12
Critical level severity issues.....	12
High level severity issues.....	12
Medium level severity issues	12
Low level severity issues	12
Owner Privileges.....	12
Automated Audit.....	13
Remix Compiler Warnings.....	13
Disclaimer.....	14
Summary	15



Introduction

This Audit Report mainly focuses on the extensive security of Tigerzilla Smart Contract. With this report, we attempt to ensure the reliability and correctness of the smart contract by complete and rigorous assessment of the system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Block Audit Report team has performed rigorous testing of the project including the analysis of the code design patterns where we reviewed the smart contract architecture to ensure it is structured along with the safe use of standard inherited contracts and libraries. Our team also conducted a formal line by line inspection of the Smart Contract i.e. a manual review, to find potential issues including but not limited to;

- **Race conditions**
- **Zero race conditions approval attacks**
- **Re-entrancy**
- **Transaction-ordering dependence**
- **Timestamp dependence**
- **Check-effects-interaction pattern (optimistic accounting)**
- **Decentralized denial-of-service attacks**
- **Secure ether transfer pattern**
- **Guard check pattern**
- **Fail-safe mode**
- **Gas-limits and infinite loops**
- **Call Stack depth**

In the Unit testing Phase, we coded/conducted custom unit tests written against each function in the contract to verify the claimed functionality from our client.

In Automated Testing, we tested the Smart Contract with our standard set of multifunctional tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included but not limited to;

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analyzing the complexity of the code in depth and detailed, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.



BLOCK AUDIT REPORT

Audit Details

Project Name: Tigerzilla

Website/ bscscan Code (**Mainnet**):

0x5181b4BdBFC172283112060B9fD734Cfbc2aA02

Languages: Solidity (Smart contract)

Platforms and Tools: Remix IDE, Truffle, Ganache, Mythril, Contract Library, Slither, Dapp.Tools, Echidna, Etheno



Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient and working according to the specifications. The audit activities can be grouped in the following three categories:

Security Sight

Identifying security related issues within each contract and the system of contract.

Sound Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices, standard software design principle, design patterns and practices.

Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Usability vs Security
- Sections of code with high complexity
- Quantity and quality of test coverage

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical Severity Issues

Issues of this level are critical to the smart contract's performance/functionality and should be fixed before moving to a production environment.

High level severity issues

Issues on this level are strongly suggested by the team to be fixed before moving to the production environment.

Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low level severity issues

Issues on this level are minor details and warning's that can remain unfixed but would be better fixed at some point in the future.



BLOCK AUDIT REPORT

Issues Checking Status

No	Issue description	Checking status
1	Compiler warnings.	Passed
2	Race conditions and Reentrancy. Cross-function race conditions.	Passed
3	Oracle calls.	Passed
4	Timestamp dependence.	Passed
5	DoS with Revert.	Passed
6	DoS with block gas limit.	Passed
7	Methods execution permissions.	Passed
8	Economy model.	Passed
9	The impact of the exchange rate on the logic.	Passed
10	Malicious Event log.	Passed
11	Scoping and Declarations.	Passed
12	Uninitialized storage pointers.	Passed
13	Arithmetic Operations accuracy.	Passed
14	Design Logic.	Passed
15	Cross-function race conditions.	Passed
16	Safe usage for Open Zeppelin module.	Passed
17	Fallback function security.	Passed
18	Send & receive ether.	Passed
19	Zero race condition approval attacks.	Passed
20	Short address attack.	Passed
21	Owner's authority to freeze.	Passed
22	Attempt to block ether flows.	Passed
23	Redundant inheritance check.	Passed
24	Silent overrides of mapping structs.	Passed
25	Function state mutability.	Passed
26	Unnecessary conversion of type.	Passed



Used Code from other Framework/Smart Contracts (direct import)

- msgSender()

- msgData()

[+] interface IERC20

- totalSupply()
- balanceOf(address account)
- transfer(address recipient, ...)
- allowance(address owner, address spender)
- approve(address spender, uint256 amount)
- transferFrom(address sender, address recipient, uint256 amount)

[+] library SafeMath

- add(uint256 a, uint256 b)
- sub(uint256 a, uint256 b)
- sub(uint256 a, uint256 b, string memory errorMessage)
- mul(uint256 a, uint256 b)
- div(uint256 a, uint256 b)
- div(uint256 a, uint256 b, string memory errorMessage)
- mod(uint256 a, uint256 b)
- mod(uint256 a, uint256 b, string memory errorMessage)

[+] library Address

- isContract(address account)
- sendValue(address payable receiver, uint256 amount)
- Call(address target, bytes memory data)
- Call(address target, bytes memory data, uint256 value)
- CallWithValue(address target, bytes memory data, uint256 value)
- CallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage)
- CallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage, bytes memory gasData)

[+] contract Ownable is Context

- owner()
- waiveOwnership()
- transferOwnership(address newOwner)
- getUnlockTime()
- getTime()
- lock(uint256 time)



BLOCK AUDIT REPORT

- unlock()

[+] interface IUniswapV2Factory

- feeTo()
- feeToSetter()
- getPair(address tokenA, address tokenB) view
- allPairs(uint) view
- allPairsLength() view
- createPair(address tokenA, address tokenB) external
- setFeeTo(address)
- setFeeToSetter(address)

[+] interface IUniswapV2Pair

- name()
- symbol()
- decimals()
- totalSupply() view
- balanceOf(address owner) view
- allowance(address owner, address spender) view
- approve(address spender, uint amount) external
- transfer(address to, uint amount) external
- transferFrom(address from, address to, uint amount) external
- DOMAINSEPARATOR()
- PERMITTYPEHASH()
- nonces(address owner) view
- permit(address owner, address spender, uint deadline, uint8 v, bytes32 r, bytes32 s) external
- MINIMUMLIQUIDITY() constant
- factory() view
- token0() view
- token1() view
- getReserves() view
- price0CumulativeLast() view
- price1CumulativeLast() view
- kLast() view
- burn(address to)
- swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external
- skim(address to, uint amount0, uint amount1) external
- sync() external
- initialize(address, address) external



BLOCK AUDIT REPORT

[+] interface IUniswapV2Router01

- factory()
- WETH()
- addLiquidity()
- addLiquidityETH()
- removeLiquidity()
- removeLiquidityETH()
- removeLiquidityWithPermit()
- removeLiquidityETHWithPermit ...
- swapExactTokensForTokens()
- swapTokensForExactTokens()
- swapExactETHForTokens(uint a ...)
- swapTokensForExactETH(uint a ...)
- swapExactTokensForETH(uint a ...)
- swapETHForExactTokens(uint a ...)
- quote(uint amountA, uint res ...)
- getAmountOut(uint amountIn, ...)
- getAmountIn(uint amountOut, ...)
- getAmountsOut(uint amountIn, ...)
- getAmountsIn(uint amountOut, ...)

[+] interface IUniswapV2Router02 is IUniswapV2Router01

- removeLiquidityETHSupporting ...
- removeLiquidityETHWithPermit ...
- swapExactTokensForTokensSupporting ...
- swapExactETHForTokensSupporting ...
- swapExactTokensForETHSupporting ...

[+] contract Tigerzilla is Context, IE ... *

- name()
- symbol()
- decimals()
- totalSupply()
- balanceOf(address account)
- allowance(address owner, address spender)
- increaseAllowance(address spender, uint amount)
- decreaseAllowance(address spender, uint amount)
- minimumTokensBeforeSwap(uint amount)
- approve(address spender, uint amount)



BLOCK AUDIT REPORT

- approve(address owner, addr ...)
- setMarketPairStatus(address ...)
- setIsTxLimitExempt(address h ...)
- setIsExcludedFromFee(address ...)
- setBuyTaxes(uint256 newLiqui ...)
- setSellTaxes(uint256 newLiqui ...)
- setDistributionSettings(uint ...)
- setMaxTxAmount(uint256 maxTx ...)
- enableDisableWalletLimit(boo ...)
- setIsWalletLimitExempt(addrre ...)
- setWalletLimit(uint256 newLi ...)
- setNumTokensBeforeSwap(uint2 ...)
- setMarketingWalletAddress(ad ...)
- setTeamWalletAddress(address ...)
- setSwapAndLiquifyEnabled(boo ...)
- setSwapAndLiquifyByLimitOnly ...
- getCirculatingSupply()
- transferToAddressETH(address ...)
- changeRouterVersion(address ...)
- transfer(address recipient, ...)
- transferFrom(address sender, ...)
- transfer(address sender, ad ...)
- basicTransfer(address sende ...)
- swapAndLiquify(uint256 tAmou ...)
- swapTokensForEth(uint256 tok ...)
- addLiquidity(uint256 tokenAm ...)
- takeFee(address sender, addr ...)



BLOCK AUDIT REPORT

Manual Audit:

For this section the code was tested/read line by line by our auditors. We used Remix IDE's JavaScript VM and testnet Kovan to test the contract functionality in a simulated environment.

b888ed68f44e433cb09d55d4060476b71292aa3aa890a4235b885261d59d...

File: Tigerzill... | Language: solidity | Size: 30632 bytes | Date: 2022-01-04T11:01:13.127Z

Critical	High	Medium	Low	Note
0	0	0	0	0



Critical Severity Issues

No critical severity issues found.

High Severity Issues

No high severity issues found.

Medium Severity Issues

No medium severity issues found.

Low Severity Issues

No low severity issues found.

Owner privileges

- None



Automated Audit

Remix Compiler Warnings

It throws warnings by Solidity's compiler. If it encounters any errors the contract cannot be compiled and deployed.

SOLIDITY COMPILER

COMPILER 0.8.4+commit.c7e474f2

Include nightly builds

LANGUAGE Solidity

EVM VERSION compiler default

COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

Compile Tigerzilla (Tigerzilla).sol

CONTRACT

Address (Tigerzilla (Tigerzilla).sol)

Publish on Ipfs IPFS

Compilation Details

ABI Bytecode



Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for the client to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that the client should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for the client to conduct the client's own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, the client agrees to the terms of this disclaimer. If the client does not agree to the terms, then please immediately cease reading this report, and delete and destroy any/all copies of this report downloaded and/or printed by the client. This report is provided for information purposes only and stays on a non-reliance basis, and does not constitute investment advice. No one/ NONE shall have any rights to rely on the report or its contents, and BlockAudit and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives).

(BlockAudit) owes no duty of care towards the client or any other person, nor does BlockAudit claim any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and BlockAudit hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effects in relation to the report. Except and only to the extent that it is prohibited by law, BlockAudit hereby excludes all liability and responsibility, and neither the client nor any other person shall have any claim against BlockAudit, for any amount or kind of loss or damage that may result to the client or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the received smart contracts alone. No related/third-party smart contracts, applications or operations were reviewed for security. No product code has been reviewed.



Summary

Smart contracts received in file named: "Tigerzilla" do not contain any high severity issues!

Note:

Please read the disclaimer above and note, the audit claims NO statements or warranties on business model, investment advice/ attractiveness or code sustainability. This report is provided for the only set of contracts mentioned in the report and does not claim responsibility to include security audits for any other contracts deployed by Owner.





BLOCK AUDIT REPORT

Official Website

www.blockaudit.report



E-Mail

team@blockaudit.report



Twitter

<https://twitter.com/BlockAudit>



Github

<https://github.com/blockauditreport>