QuillAudits

Audit Report
May, 2022

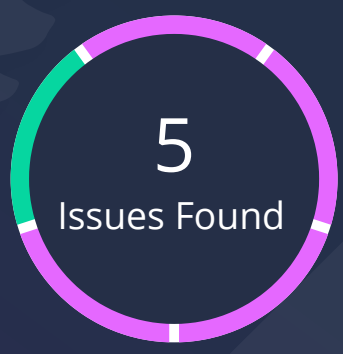
For

Bitsliced

Table of Content

Executive Summary

Project Name	Bitsliced LLC(GmbH)
Overview	SLICED token was designed to function within the Bitsliced marketplace app. Community members are able to mint, swap, buy, sell items using SLICED as a utility token. Furthermore, special activities inside the app generate fees that are also paid with SLICED. The most important key driver of SLICED value proposition is the tokenization of commercial transactions. Bitsliced has been closing framework contracts with partners in diverse industries like real estate and automotive distribution networks
Timeline	April 27, 2022 - April 29, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Sliced.sol contract by Bitsliced LLC(GmbH) for quality, security, and correctness.
Github	*https://github.com/Omar-Bitsliced-LLC/SLICED-Smart-Contract/blob/main/Sliced.sol*
Commit	d243190125c15dbe9628a844f764a5d0e484f3a0
Fixed In	*https://github.com/Bitsliced/SLICED-Smart-Contract/blob/main/Sliced.sol*
Commit	6218730d83420fcd22a093dd6306b30e4c71259a



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	1
Partially Resolved Issues	0	0	0	1
Resolved Issues	0	0	0	2

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 - Sliced.sol

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

A.1 Missing events for updateUserState() and updateAdmin()

Line 81, 86

```
function updateUserState(address _user,bool _state) public onlyAdmin {
    require(_user!=address(0),"SLICED::Address NULL");
    blacklistedAddresses[_user] = _state;
}

    function updateAdmin(address _user,bool _state) public onlyOwner {
        require(_user!=address(0),"SLICED::Address NULL");
        admins[_user] = _state;
    }
```

Description

Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it. Updating the status of an admin or blacklisting an address are some significant actions performed by privileged users (owner and admin), it is recommended to emit an event about it.

Remediation

Consider emitting an event whenever updateUserState() or updateAdmin() are called.

Status

Acknowledged

Auditor's Comment: The Bitsliced team acknowledged this finding and believes that there is no need to add events since they are not using an application that will listen to it. Moreover, emitting an event always consumes some amount of gas. Hence, the team decided to acknowledge this issue.

Informational Issues

A.2 Public functions that could be declared external inorder to save gas

Description

Whenever a function is not called internally, it is recommended to define them as external instead of public in order to save gas. For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If your function is only called externally, then you should explicitly mark it as external. External function's parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge.
Here is a list of functions that could be declared external:

- sendTokens #L70
- updateUserState #L81
- updateAdmin #L86
- pause #L96
- unpause #L100

Status

Fixed

A.3 Variables That Could Be Declared As Constant inorder to save gas

Description

There are multiple variables in the contract, whose value is never updated, it is recommended to declare those variables as constant.
Here is a list of variables that could be declared constant:

- duration #L10
- community #L13
- reward #L14
- marketing #L15
- dev #L16
- team #L17
- legal #L18
- launchpad #L19

Status

Fixed

A.4 Centralization Issue

Description

The contract has implemented blacklisting functionality, which means that the users with Higher privilege, can blacklist any user and prevent them from transferring their tokens. Or pause the contract. While we understand that this is for security reasons and to prevent bad actors, developers must keep in mind that if the accounts with higher privileges are compromised, it can adversely affect the contract and it's users.

Remediation

A proper guideline must be provided about when an address can be blacklisted and when it can be whitelisted. Additionally, keys to the privileged account must be handled diligently.

Status

Partially Resolved

Auditor's Comment: Bitsliced team has provided clear documentation regarding blacklisting functionality in their whitepaper. The intention behind implementing this functionality is to prevent bad actors from interacting with the contract. However, this privilege still remains with the owner of the contract. And in case, the keys of the owner account get compromised, it may work adversely in the future. Currently, we do not see this as a major security vulnerability but we would recommend the owner of the contract to diligently the keys and use this functionality only when necessary.

A.5 Using Block values as a proxy for time

Description

Here in function sendTokens() and constructor(), a control flow decision is made based on The 'block.timestamp' environment variable. Note that the values of variables like coinbase, gaslimit, block number,and timestamp are predictable and can be manipulated by malicious miners. Also, keep in mind that attackers know hashes
of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that the use of these variables introduces a certain level of trust into miners. In this particular case, since it is used for determining a duration of one month, it is not a major issue. Even if the miner manipulates it by some time, it won't be a major threat. However, developers must keep in mind that these values can be manipulated and avoid using block.timestamp if possible.

Remediation

Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may use oracles.

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below

- Should test all getters
- Should validate balance of each address after deployment
- Should not be able to call sendTokens more than once in a month
- OnlyAdmin should be able to blacklist user
- OnlyOwner should be able to modify admin roles
- updateUserState and updateAdmin should validate for zero address
- Transfer must fail if the contract is paused
- Blacklisted user must not be able to transfer
- Onlyowner should be able to pause and unpause contract.
- Should calculate and transfer tokens to respective accounts correctly whenever sendTokens is called.

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 Bitsliced LLC(GmbH) platform. 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.At the end Bitsliced LLC(GmbH) team Resolved few issues and Acknowledged others.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Bitsliced LLC(GmbH) 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 Bitsliced LLC(GmbH) 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

Bitsliced

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com