HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: BRIKN
Date: February 14, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for BRIKN
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://brikn.io/
Changelog	27.01.2023 - Initial Review 09.02.2023 - Second Review 14.02.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by BRIKN (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository	https://github.com/Decubate-com/smart-contracts
Commit	a41846f188f10e1678afb57909cbe5712f684037
Whitepaper	Link
Technical description	Link
Contracts	File: ./contracts/BRIKToken.sol SHA3: ed4ab404169cd629e6bc082a73d5c7b11dc8e3c15d49a207b4362f3e27de0ee2 File: ./contracts/Whitelisted.sol SHA3: 219f1f59b9d3cc61faec6bd354978db1f4012e8f0606dc1342f4ddf777a4e8ff

Second review scope

Repository	https://github.com/Decubate-com/smart-contracts
Commit	7619853430883901586a96b303ae68f44b193737
Whitepaper	Link
Functional requirements	Link
Technical description	Link
Contracts	File: ./contracts/BRIKToken.sol SHA3: c1ba0bc0a7fed615d70cfcaa40b06d16fceaea07c7f6a132d6bfa75d578fe9b8 File: ./contracts/Whitelisted.sol SHA3: 01ce3d7172782a3f6d8ec0c28697707202cb9d0dc71e3788c20e3547e1cb7ddd

Third review scope

Repository	https://github.com/Decubate-com/smart-contracts
Commit	01827e9e0d3746235776745019e8035d4b20e0bb
Whitepaper	Link

Functional requirements	Link
Technical description	Link
Contracts	File: ./contracts/BRIKToken.sol SHA3: 0b6d87af096451412eb183b13f86d64b1ce59c5c5e2715a2f4ca79a89acac752 File: ./contracts/Whitelisted.sol SHA3: 5eaf98dc78ee080fd53a389b349c9ae34a5b6465dbc9a450814ecba2852e9c62

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Functional requirements are partially outdated.
- Technical description does not correspond to the dev environment.

Code quality

The total Code Quality score is **9** out of **10**.

- The development environment is configured.
- Some project dependencies (*truffle*) are considered to be installed globally and are not mentioned in the *package.json* file.
- Contract misses an event for state variable updation.

Test coverage

Code coverage of the project is **100%** (branch coverage).

Security score

As a result of the audit, the code contains **2** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**.

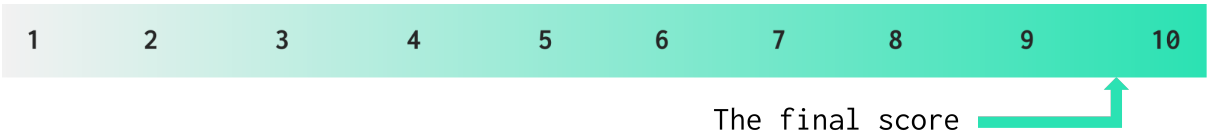

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
26 January 2023	3	3	0	0
9 February 2023	2	1	0	0
14 February 2023	2	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Not Relevant
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant

Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

BRIKN is a mixed-purpose smart contract system which includes the audit scope contracts:

- *Whitelisted* – access control contract.
  Roles supported:
  - Owner
  - Whitelisted accounts
  - Blacklisted accounts
- *BRIKToken* – burnable *ERC20* token (inherit *Whitelisted*).
  Features:
  - is not mintable
  - transfers are blocked for blacklisted accounts
  - transfers are blocked till specified moment (except of transfers from/to whitelisted accounts)
  - swaps in set pairs are blocked till specified moment

Privileged roles

Owner:

- able to set whitelisted accounts
- able to lock owned assets, transferring them to the *0xdEaD* address

Whitelisted accounts:

- able to setup blacklisted accounts
- able to setup date till which tokens selling would not be possible
- able to setup date till which tokens transfers would not be possible

Risks

- In case the user account is blacklisted, user funds are locked.
- Whitelisted users may pause transactions on the contract for any period of time.
- Whitelisted users may pause swaps at any exchange service (in the pairs include the *BRIKToken* asset) for any period of time.

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

M01. Duplicated State Variable

The *pair* variable in the *BRIKToken* contract stores the same value, which *pairAddress* in the *Whitelisted* contract does.

According to the current implementation, *pair* variable is considered redundant as it could be replaced with *pairAddress*.

Path: ./contracts/BRIKToken.sol : pair

Recommendation: remove the redundant state variable and change all references to the duplicated one.

Status: Fixed (second scope)

M02. Requirement Violation

According to the documentation, users should be able to burn their tokens. However, the functionality is missed.

The *burn* function is implemented under *onlyOwner* modifier.

Path: ./contracts/BRIKToken.sol : burn()

Recommendation: accept anyone to burn funds or fix the documentation.

Status: Fixed (second scope)

M03. Undocumented Behavior

The token blacklist, timelock and saleblock functionalities are not described in the documentation.

Note: saleblock functionality may be bypassed by using other DEXes or by combining several swap pairs on the direct DEX.

Path: ./contracts/Whitelisted.sol : isSaleBlocked()

Recommendation: disclose information about implemented restrictions to the users, get rid of unfinalized functionality, or accept the bypass possibility.

Status: Fixed (second scope)

M04. Requirement Violation

According to the documentation, the contract may be locked from trading on multiple pair addresses.

However, only one pair address may be blocked as the pairs list management function is internal and inaccessible by the owner.

Paths:
./contracts/BRIKToken.sol
./contracts/Whitelisted.sol : setPairAddress(), isPair

Recommendation: provide ability for the owner to manage the trading pairs or update documentation to be consistent with implementation.

Status: Fixed (third scope)

Low

L01. Floating Pragma

Locking the pragma helps 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.

The project uses floating pragmas `^0.8.10`.

Paths:
./contracts/BRIKToken.sol
./contracts/Whitelisted.sol

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (second scope)

L02. Redundant Use of SafeMath

Since Solidity `v0.8.0`, the overflow/underflow check is implemented via `ABIEncoderV2` on the language level - it adds the validation to the bytecode during compilation.

There is no need to use the `SafeMath` library.

Path: ./contracts/BRIKToken.sol

Recommendation: remove usage of the `SafeMath` library.

Status: Fixed (second scope)

L03. Unchecked Return Value

The function returns the status of an executed action, but the status is ignored.

During further development, the returned status may become not only `true` and the system may reach an unexpected state.

Paths:
./contracts/BRIKToken.sol : constructor()
./contracts/Whitelisted.sol : setPairAddress()

Recommendation: require the return value to be *true* or remove the redundant return.

Status: Fixed (second scope)

L04. Documentation Mismatch

SafeMath lib is mentioned in the documentation. However, it was removed from the implementation.

Path: ./contracts/BRIKToken.sol : constructor()

Recommendation: keep documentation up-to-date with implementation.

Status: Fixed (third scope)

L05. Redundant Statements

It is unnecessary to inherit the *Whitelisted* contract with *Context* as the *Ownable* contract inherits *Context* and *Whitelisted* the *Ownable* contract.

The *import Context.sol* statement is redundant as *Context* may be loaded from the *Ownable.sol* file.

Path: ./contracts/Whitelisted.sol

Recommendation: remove redundant statements.

Status: Fixed (third scope)

L06. Documentation Mismatch

Amount of constructor parameters mismatch implementation.

Path: ./contracts/BRIKToken.sol : constructor()

Recommendation: keep documentation up-to-date with implementation.

Status: Reported

L07. Missing Event for State Updation

Critical state changes should emit events for tracking things off-chain.

The function does not emit an event on change of a state variable.

This may lead to inability for users to subscribe events and check what is going on with the project.

Path: ./contracts/Whitelisted.sol : setPairAddress()

Recommendation: emit events on critical state changes.

Status: New

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.