HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ByTrade Venture Capital
Date: February 23, 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 ByTrade Venture Capital
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.bytrade.io
Changelog	09.02.2023 - Initial Review 23.02.2023 - Second 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 ByTrade Venture Capital (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	Not Provided
Commit	Not Provided
Whitepaper	https://bytrade.io/src/assets/white_paper_btt_v1.0.pdf
Functional Requirements	Bytrade Token Technical Documentation
Technical Requirements	Bytrade Token Technical Documentation
Contracts Addresses	https://bscscan.com/token/0xfdcb304bef77b5d4b70381b0a1812c43814fb87b#code
Contracts	File: ByTradeToken.sol SHA3:22b0611170f995d3a94d4abfc9e553a4018e378414b08714e5068c66acc1bc60

Second review scope

Repository	https://github.com/bytradeio/Bytrade-ERC20Token
Commit	510e568774ecfef28ab21ca6bf759fbcffdfd3cd
Functional Requirements	Bytrade-Doc.pdf
Technical Requirements	Bytrade-Doc.pdf
Contracts Addresses	https://bscscan.com/token/0xe8bd1ca97f8a0582335407b40b0576df641fa94c#code
Contracts	File: ./contracts/BytradeToken.sol SHA3:f4aaf5d2b9ea0cf14f6fbb4711f755212b4bdb351b76a0f92aea5ccce676b29b

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 **7** out of **10**.

- Functional requirements are provided.
- Technical description is not sufficient:
  - Test coverage and running instructions are not provided.
  - Development environment is not configured.

Code quality

The total Code Quality score is **5** out of **10**.

- Style Guide and Best practices are not followed.
- The development environment is not configured.
- Not all issues are fixed.
- Presence of unused parameters and contracts.

Test coverage

Code coverage of the project is **0%** (branch coverage).

Security score

As a result of the audit, the code contains **2** medium and **9** low severity issues. The security score is **8** 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: **7.3**.

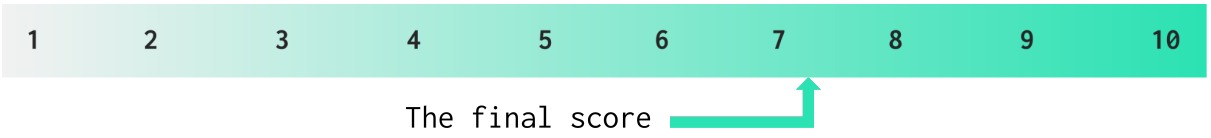

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
9 February 2023	11	1	2	0
23 February 2023	9	2	0	0

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

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.	Passed
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.	Failed
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

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

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.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Failed
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

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

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.	Not Relevant
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Failed
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.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

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

System Overview

The Bytrade token meets the ERC20 standard, and is extended with the following functionalities (restricted to the contract owner):

- Ability to pause/unpause transactions
- Ability to renounce ownership
- Ability to transfer ownership

Privileged roles.

- The owner can pause the contract, blocking the possibility for any user to perform a token transfer or allowance modification.

Risks

- Token can be paused/unpaused by the owner at any time.
- The defined token in ByTradeToken uses the symbol BTT, which is already used by the BitTorrent token.
- The contract ByTradeToken implements a renounceOwnership() functionality, which will remove the contract ownership. Since the contract implements several functions that can only be called by the owner pause(), unpause(). Removing the owner will lock said functions. If renounceOwnership() is called when the contract is paused, the consequence is a Denial of Service for transfer(), transferFrom(), approve(), increaseAllowance() and decreaseAllowance().

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

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Requirements Violation

In the functional documentation of the project, the contract ByTradeToken is said to meet the ERC20 standard. However, the code refers to the BEP20 standard instead, which follows a different standard.

The code should not violate requirements provided by the Customer since it can lead to misinformation for users, as well as lack of transparency and their derived issues.

Paths:
./contracts/ByTradeToken
./contracts/BEP20
./contracts/IBEP20

Recommendation: The code should meet the functional requirements.

Status: Fixed (Revised commit: 510e568)

H02. BEP Standard Violation

The code refers to the BEP20 standard, but it does not meet its requirements.

Not following standards can lead to misinterpretation, unexpected behavior and integration issues with other systems.

Paths:
./contracts/ByTradeToken
./contracts/BEP20
./contracts/IBEP20

Recommendation: Token standards should follow the official sources.

Status: Fixed (Revised commit: 510e568)

■■ Medium

M01. Unscalable Functionality: Repeated Checks in Functions

In ByTradeToken, the functions transfer(), transferFrom(), approve(), increaseAllowance() and decreaseAllowance() implement a require() to make sure none of the addresses involved in the transaction are blacklisted.

This check is done calling isBlacklisted() twice at each of the mentioned functions, resulting in a repetitive code. Instead, a new

modifier can be created, which includes these checks, increasing the code readability, debugging and upgrades.

Path:
./contracts/ByTradeToken: transfer(), transferFrom(), approve(), increaseAllowance(), decreaseAllowance()

Recommendation: It is recommended to create a new modifier to implement the checks.

Status: Fixed (Revised commit: 510e568)

M02. Unscalable Functionality: Copy of Well-Known Contracts

Well-known contracts from projects like OpenZeppelin should be imported directly from the source as the projects are in development and may update the contracts in future.

Paths:
./contracts/Context
./contracts/IERC20
./contracts/SafeMath
./contracts/Address
./contracts/ERC20
./contracts/Ownable

Recommendation: Import the contracts directly from the source instead of copying them into the main file.

Status: New

M03. SafeMath in ^0.8.0

The SafeMath library is integrated and implemented into the project. However, SafeMath is already integrated from Solidity 0.8.0 and its use is redundant.

Path:
./contracts/SafeMath

Recommendation: Remove the implementation of SafeMath.

Status: New

Low

L01. Floating Pragma

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.

Path:
./contracts/ByTradeToken

Recommendation: It is recommended to lock the Solidity pragma version. See more: SWC-103.

Status: Reported

L02. Style Guide: Order of Functions

The provided projects should follow the official guidelines. Functions should be grouped according to their *visibility* and ordered:

1. Constructor
2. Receive function (if exists)
3. Fallback function (if exists)
4. External
5. Public
6. Internal
7. Private

Path:
./contracts/ByTradeToken

Recommendation: Follow the official Solidity guidelines.

Status: Reported

L03. Outdated Solidity Version

Using an outdated compiler version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Using the current version of Solidity is generally considered best practice because it includes the latest updates and bug fixes. Newer versions address security vulnerabilities that may have been discovered in previous versions, making them more secure to use. Additionally, newer versions include new features and improvements that make writing and deploying smart contracts easier and more efficient. Using an outdated version of Solidity may expose contracts to potential security risks and make it more difficult to take advantage of newer features and capabilities.

Path:
./contracts/ByTradeToken

Recommendation: Use an up-to-date compiler version. See more: SWC-102.

Status: Fixed (Revised commit: 510e568)

L04. Outdated OpenZeppelin Version

Using an outdated OpenZeppelin version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Using the current version of OpenZeppelin is generally considered best practice because it includes the latest updates and bug fixes. Newer versions address security vulnerabilities that may have been discovered in previous versions, making them more secure to use. Additionally, newer versions include new features and improvements that make writing and deploying smart contracts easier and more efficient. Using an outdated version of OpenZeppelin may expose contracts to potential security risks and make it more difficult to take advantage of newer features and capabilities.

Path:
./contracts/ByTradeToken

Recommendation: Use an up-to-date OpenZeppelin version. Read more about OpenZeppelin versions.

Status: Reported

L05. Missing Error Message

In ByTradeToken, the modifier whenNotPaused() does not provide an error message in the require() check.

Error messages are intended to notify users about failing conditions, and should provide enough information so that the appropriate corrections needed to interact with the system can be applied. Uninformative error messages greatly damage the overall user experience, thus lowering the system's quality.

If the mentioned require statement fails the checked condition, the transaction will revert silently without an informative error message.

Path:
./contracts/ByTradeToken: whenNotPaused()

Recommendation: It is recommended to add an appropriate error message in the modifier whenNotPaused().

Status: Reported

L06. Functions that Can Be Declared External

In order to save Gas, *public* functions that are never called from the same contract should be declared as *external*.

Path:
./contracts/ByTradeToken: pause(), unpause(), addToBlacklist(), removeFromBlacklist(), transfer(), transferFrom(), approve(), increaseAllowance(), decreaseAllowance()

Recommendation: Use the *external* attribute for functions that are never called from the contract.

Status: Reported

L07. NatSpec Typo

The NatSpec comments line in ByTradeToken's function addToBlacklist() refers to "addess" instead of "address".

Path:
./contracts/ByTradeToken

Recommendation: Correct the terminology in the comments line.

Status: Fixed (Revised commit: 510e568)

L08. Missing NatSpec

The provided contracts include NatSpec comments, although not enough for a proper understanding of every last detail: many functions do not have any NatSpec comment, as well as several variables.

Path:
./contracts/ByTradeToken

Recommendation: Provide NatSpec for all contracts' components.

Status: Fixed (Revised commit: 510e568)

L09. Undocumented Literals

The constructor() sets 10 billion tokens to mint as 10*(10**9)*10**18.

It is advised to use named constants instead of literals when important functionality is performed (e.g. MINT_AMOUNT). The function decimals() can also be used instead of 10**18.

Path:
./contracts/ByTradeToken: constructor()

Recommendation: Consider replacing literals with named constants in the contract.

Status: Reported

L10. Unscalable Functionality: Missing Events Arguments

The contract ByTradeToken defines the events Pause() and Unpause() with no arguments.

Since the contract has an ownership transference functionality, it is recommended to track the address of the pausable functions' calls.

Path:
./contracts/ByTradeToken

Recommendation: It is recommended to provide the msg.sender data in the Pause() and Unpause() events.

Status: Reported

L11. Redundant Initialization of Default Values

The constructor() in ByTradeToken initializes the state variable paused = false. Uninitialized bools are set as false, so it is redundant to set said variable to false.

Accessing storage variables is expensive in terms of Gas, and should therefore be minimized.

Path:
./contracts/ByTradeToken: constructor().

Recommendation: Avoid unnecessary variable initializations.

Status: Reported

L12. Unused Parameters

Unused variables are allowed in Solidity and do not pose a direct security issue. However, it is best practice to avoid them as they can cause an increase in computations (and unnecessary Gas consumption) and decrease readability.

Path:
./contracts/ByTradeToken.sol: blacklist, AddedToBlacklist(), RemovedFromBlacklist().

Recommendation: Any unused parameter should be removed from the code.

Status: New

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

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.