HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ByTrade Venture Capital
Date: Mar 03, 2023

Document

Name	Smart Contract Code Review and Security Analysis Report for ByTrade Venture Capital
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OU
Type	Staking
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://bytrade.io/
Changelog	24.02.2023 - Initial Review 03.03.2023 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by ByTrade Venture Capital 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/bytradeio/Bytrade-Staking
Commit	00cc24ea2bdbe16a77237b64b9f6b35da0acae00
Whitepaper	https://docs.google.com/document/d/1i6Imuu7W7bHMkh1-LjycnhRpz9QlTzM5rvdLQkPIx5E
Functional Requirements	https://docs.google.com/document/d/1i6Imuu7W7bHMkh1-LjycnhRpz9QlTzM5rvdLQkPIx5E
Technical Requirements	-
Contracts	File: ./contracts/Bytrade_Staking.sol SHA3:72a70efd80caebc5591dd6f583e2bff4efbb7f9b84e42486f60a295781c3f50b

Second review scope

Repository	https://github.com/bytradeio/Bytrade-Staking
Commit	a686c324db38d378b45fe758a3a7ad83e8341473
Whitepaper	https://docs.google.com/document/d/1i6Imuu7W7bHMkh1-LjycnhRpz9QlTzM5rvdLQkPIx5E
Functional Requirements	https://docs.google.com/document/d/1i6Imuu7W7bHMkh1-LjycnhRpz9QlTzM5rvdLQkPIx5E
Technical Requirements	https://docs.google.com/document/d/1i6Imuu7W7bHMkh1-LjycnhRpz9QlTzM5rvdLQkPIx5E
Contracts	File: ./contracts/Bytrade_Staking.sol SHA3:c60b4b5656db4eb2b69c4c6bcc5b1cd90970732c633caab1c1807b402c72774b
Contracts Addresses	https://bscscan.com/address/0x87a39f76d45378bf995aa89048ae015ffeff477f#code

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

www.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**.
- Overall system requirements are provided.
- NatSpec is insufficient.
- Run instructions are provided.

Code quality

The total Code Quality score is **8** out of **10**.
- Best practice violations.
- Test coverage is insufficient.

Test coverage

Code coverage of the project is **0.00%** (branch coverage).
- Code is not covered with tests.

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: **9.4**.

1	2	3	4	5	6	7	8	9	10

The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
24 Feb 2023	3	1	2	0
03 Mar 2023	2	0	0	0

www.hacken.io

System Overview

ByTrade Venture Capital is a system with the following contracts:
- *Bytrade_Staking* — a contract that rewards users for staking their tokens. APY depends on the tokens provided by the owner and cannot be calculated before reward tokens are deposited.

Privileged roles

The contract does not have any privileged roles.

Risks

- The view function returns a dynamic array of unlimited size. It is theoretically possible that the Gas cost of executing this function will exceed the Gas limit set in the node.
- Smart contract is available to receive Ether, but it is not necessary for it to work.

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.	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

www.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.	Passed
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.	Passed
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

www.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.	Not Relevant
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.	Passed
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

www.hacken.io

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Funds Lock

Function *stake(uint,uint)* is an external payable function. This means that the contract can accept native coins to its account, but does not have any withdrawal mechanisms. It is an inefficient use of Gas and does not match the technical and functional requirements.

Path: ./contracts/Bytrade_Staking.sol : stake(uint,uint)

Recommendation: Forbid the contract to accept native coins or provide mechanisms of withdrawal.

Found in: 00cc24ea2bdbe16a77237b64b9f6b35da0acae00

Status: Fixed (Revised commit: a686c324db38d378b45fe758a3a7ad83e8341473)

H02. Insufficient Balance

The contract operates with some *initialBalance* value which is considered as some initial value that should be available on the contract. Availability of this balance is not guaranteed by the contract code.

Users' balances can be used for paying rewards that can lead to the inability to withdraw funds from the contract.

Path: ./contracts/Bytrade_Staking.sol : stake(uint,uint)

Recommendation: Ensure that the *initialBalance* is really available on the contract before allowing to *stake*.

Found in: 00cc24ea2bdbe16a77237b64b9f6b35da0acae00

Status: Fixed (Revised commit: a686c324db38d378b45fe758a3a7ad83e8341473)

■■ Medium

M01. Gas Limit and Loops

Function *getUserDeposits(address)* returns a value depending on the amount of data stored in the smart contract. It is possible to increase the cost of the transaction so that it will not be executed.

Path: ./contracts/Bytrade_Staking.sol : getUserDeposits(address)

Recommendation: Return constant size data.

www.hacken.io

Found in: 00cc24ea2bdbe16a77237b64b9f6b35da0acae00

Status: Fixed (Revised commit: 2ba351c)

Low

L01. Floating Pragma

The smart contract uses floating pragma *^0.8.4*.

Path: ./contracts/Bytrade_Staking.sol

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

Found in: 00cc24ea2bdbe16a77237b64b9f6b35da0acae00

Status: Fixed (Revised commit: a686c324db38d378b45fe758a3a7ad83e8341473)

L02. Boolean Equality

Boolean constants can be used directly and do not need to be compared to *true* or *false*.

Path: ./contracts/Bytrade_Staking.sol : stake(uint,uint), unstake(uint,uint)

Recommendation: Remove boolean equality.

Found in: 00cc24ea2bdbe16a77237b64b9f6b35da0acae00

Status: Fixed (Revised commit: a686c324db38d378b45fe758a3a7ad83e8341473)

L03. Use of Hard-Coded Values

The following lines of code contain values that can be replaced with const variables: *63*, *69*, *88*, *172*, *173*, *193*.

Path: ./contracts/Bytrade_Staking.sol

Recommendation: It is recommended to use const variables instead of hardcoding values in the code.

Found in: 00cc24ea2bdbe16a77237b64b9f6b35da0acae00

Status: Fixed (Revised commit: a686c324db38d378b45fe758a3a7ad83e8341473).

L04. Use of Hard-Coded Values

The following lines of code contain values that can be replaced with const variables: *62*, *186*, *211*.

Path: ./contracts/Bytrade_Staking.sol

www.hacken.io

Recommendation: It is recommended to use const variables instead of hardcoding values in the code.

Found in: a686c324db38d378b45fe758a3a7ad83e8341473

Status: New

L05. Variables that Should Be Declared External

The state variable that does not change its *month* value should be declared constant to save Gas.

Path: ./contracts/Bytrade_Staking.sol

Recommendation: Declare the above-mentioned variables as constants.

Found in: a686c324db38d378b45fe758a3a7ad83e8341473

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.