

**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** BTC2.0  
**Date:** 27 Jul, 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

## Document

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for BTC2.0
<b>Approved By</b>	Oleksii Zaiats   SC Audits Head at Hacken OÜ
<b>Tags</b>	ERC20 token
<b>Platform</b>	EVM
<b>Language</b>	Solidity
<b>Methodology</b>	<a href="#">Link</a>
<b>Website</b>	Not provided
<b>Changelog</b>	27.07.2023 - Initial Review

## Table of contents

<b>Introduction</b>	<b>4</b>
<b>System Overview</b>	<b>4</b>
<b>Executive Summary</b>	<b>5</b>
<b>Risks</b>	<b>6</b>
<b>Checked Items</b>	<b>7</b>
<b>Findings</b>	<b>10</b>
Critical	10
High	10
Medium	10
Low	10
L01. Floating pragma	10
Informational	10
I01. Function state mutability can be changed to pure	10
<b>Disclaimers</b>	<b>11</b>
<b>Appendix 1. Severity Definitions</b>	<b>12</b>
Risk Levels	12
Impact Levels	13
Likelihood Levels	13
Informational	13
<b>Appendix 2. Scope</b>	<b>14</b>

## Introduction

Hacken OÜ (Consultant) was contracted by BTC2.0 (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.

## System Overview

*BTC2Token* is a simple token with the following contracts:

- *BTC2Token.sol* – ERC-20 token that mints all initial supply to a deployer. Additional minting is not allowed.

It has the following attributes:

- Name: Bitcoin 2.0
- Symbol: BTC2.0
- Decimals: 8
- Total supply: Could not be verified during the audit

## Privileged roles

There are no privileged roles

## 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 **10** out of **10**.

- Functional requirements are provided.
- The description of the development environment is missing.

### Code quality

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

- The development environment is configured.
- Best practices are not followed: Floating pragma is set

### Test coverage

Code coverage of the project is **0%** (branch coverage), with a mutation score of **0%**.

- No tests were given. However, since the audit Lines of Code does not exceed 250, this does not affect the score.

### Security score

As a result of the audit, the code contains **1** low issue. 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: **9.8**. The system users should acknowledge all the risks summed up in the risks section of the report.



The final score

*Table. The distribution of issues during the audit*

Review date	Low	Medium	High	Critical
27 July 2023	1	0	0	0

## Risks

- All tokens are minted to a single address. The secureness of the supply depends on the secureness of key storage.
- The contract deployer is the only receiver of the whole initialSupply. It creates a centralization risk due to the fact that the documented initial distribution of the initialSupply can be changed by the owner or not distributed at all. As a consequence, the specified receivers of the initial distribution will receive fewer tokens or no tokens at all, which might make the contract unavailable for other users, except the contract deployer.

## Checked Items

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

Item	Description	Status	Related Issues
<b>Default Visibility</b>	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
<b>Integer Overflow and Underflow</b>	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant	
<b>Outdated Compiler Version</b>	It is recommended to use a recent version of the Solidity compiler.	Passed	
<b>Floating Pragma</b>	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Failed	L01
<b>Unchecked Call Return Value</b>	The return value of a message call should be checked.	Not Relevant	
<b>Access Control &amp; Authorization</b>	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
<b>SELFDESTRUCT Instruction</b>	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
<b>Check-Effect-Interaction</b>	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
<b>Assert Violation</b>	Properly functioning code should never reach a failing assert statement.	Passed	
<b>Deprecated Solidity Functions</b>	Deprecated built-in functions should never be used.	Passed	
<b>Delegatecall to Untrusted Callee</b>	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
<b>DoS (Denial of Service)</b>	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

<b>Race Conditions</b>	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
<b>Authorization through tx.origin</b>	tx.origin should not be used for authorization.	Not Relevant	
<b>Block values as a proxy for time</b>	Block numbers should not be used for time calculations.	Not Relevant	
<b>Signature Unique Id</b>	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	
<b>Shadowing State Variable</b>	State variables should not be shadowed.	Passed	
<b>Weak Sources of Randomness</b>	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
<b>Incorrect Inheritance Order</b>	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
<b>Calls Only to Trusted Addresses</b>	All external calls should be performed only to trusted addresses.	Not Relevant	
<b>Presence of Unused Variables</b>	The code should not contain unused variables if this is not <a href="#">justified</a> by design.	Passed	
<b>EIP Standards Violation</b>	EIP standards should not be violated.	Passed	
<b>Assets Integrity</b>	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
<b>User Balances Manipulation</b>	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
<b>Data Consistency</b>	Smart contract data should be consistent all over the data flow.	Passed	



<b>Flashloan Attack</b>	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. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	
<b>Token Supply Manipulation</b>	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Failed	
<b>Gas Limit and Loops</b>	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	
<b>Style Guide Violation</b>	Style guides and best practices should be followed.	Passed	
<b>Requirements Compliance</b>	The code should be compliant with the requirements provided by the Customer.	Passed	
<b>Environment Consistency</b>	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed	
<b>Secure Oracles Usage</b>	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	
<b>Tests Coverage</b>	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	
<b>Stable Imports</b>	The code should not reference draft contracts, which may be changed in the future.	Passed	

## Findings

### Critical

No critical severity issues were found.

### High

No high severity issues were found.

### Medium

No medium severity issues were found.

### Low

#### L01. Floating pragma

Impact	Low
Likelihood	Low

The contract uses floating pragma ^0.8.0.

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.

**Path:** ./contracts/BTC2Token.sol

**Recommendation:** consider locking the pragma version and deploy the smart contract with any of the following Solidity versions: 0.8.18, 0.8.19, 0.8.20.

**Found in:** f3722d3

**Status:** New

## Informational

#### I01. Function state mutability can be changed to pure

A function that does not read or modify the variables of the state should be declared pure.

**Path:** ./contracts/BTC2Token.sol: decimals()

**Recommendation:** Change function state mutability to pure.

**Found in:** f3722d3

**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.

## Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

### Risk Levels

**Critical:** Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

**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.

**Medium:** Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

**Low:** Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don't affect security score but can affect code quality score.

## Impact Levels

**High Impact:** Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

**Medium Impact:** Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

**Low Impact:** Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

## Likelihood Levels

**High Likelihood:** Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

**Medium Likelihood:** Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

**Low Likelihood:** Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

## Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

## Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

### Initial review scope

<b>Repository</b>	<a href="https://github.com/btc2wtf/btc2-contracts">https://github.com/btc2wtf/btc2-contracts</a>
<b>Commit</b>	f3722d3bcf85774e362d4654316d03111654bb6d
<b>Whitepaper</b>	Jira attachments
<b>Requirements</b>	Jira attachments
<b>Technical Requirements</b>	Jira attachments
<b>Contracts</b>	File: contracts/BTC2Token.sol SHA3: f60601bea8b55637f80dcd83228fa7c4dd47f9a913d9d3b91345631f