SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Swissborg
Date: 22 Aug, 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 Swissborg
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://swissborg.com
Changelog	27.06.2023 - Initial Review 05.07.2023 - Second Review 22.08.2023 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Swissborg (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

SwissBorgToken is a Solidity smart contract that implements the new BORG token. It is based on the *ERC20* standard and incorporates additional functionality from the *ERC20Burnable*, *ERC20Permit*, and *ERC20Votes* extensions provided by the OpenZeppelin library. Initial supply of new BORG is 1 billion tokens minus CHSB (old BORG) tokens sent to address(0) (to not mint tokens that cannot be migrated).

ChsbToBorgMigrator is a Solidity smart contract that facilitates the migration process from the CHSB token to the BORG token. ChsbToBorgMigrator utilizes various libraries from the OpenZeppelin framework, including *OwnableUpgradeable* for managing ownership, *PausableUpgradeable* for pausing and unpausing contract operations, and *UUPSUpgradeable* for transparent contract upgrades.

Privileged roles

ChsbToBorgMigrator contract has 2 defined role:

- owner - role assigned using OpenZeppelin *OwnableUpgradeable* library. Address with this role can change the address of a manager role.
- manager - role assigned with Smart Contract variable. Address with this role can pause and unpause contract.

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**.

- Technical description is robust.
- Functional requirements are detailed.

Code quality

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

- Code does not contain any code quality violation.

Test coverage

Code coverage of the project is **100%** (branch coverage), with a mutation score of **63.44%**.

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.

Security score

As a result of the audit, the code contains no 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: **10**. The system users should acknowledge all the risks summed up in the risks section of the report.

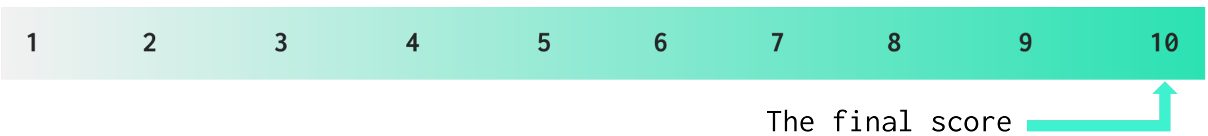

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
27 June 2023	2	1	0	0
05 July 2023	0	0	0	0
22 August 2023	0	0	0	0

Risks

- The system currently lacks the implementation of the *AccessControlUpgradeable* library provided by OpenZeppelin for the purpose of managing two distinct roles within the system. Presently, the system relies on the usage of the *OwnableUpgradeable* library to handle the owner role, while a custom mechanism has been devised to manage the manager role.

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

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	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	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	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	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Flashloan Attack	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	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	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	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	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	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	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	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

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

M01. Copy Of Well Known Contract

Impact	Low
Likelihood	High

The contract utilizes the *OwnableUpgradeable* library to grant privileges to the contract owner. Furthermore, the contract includes a *manager* role, which is stored as a contract variable. When multiple roles are defined within a system, it is considered a best practice to utilize specialized libraries such as *AccessControlUpgradeable* from OpenZeppelin. These libraries provide enhanced functionality and security for managing role-based access control.

Path: /contracts/ChsbToBorgMigrator.sol;

Recommendation: use *AccessControlUpgradeable* from OpenZeppelin library instead of *OwnableUpgradeable*.

Found in: 856cce0

Status: Mitigated (Customer: current implementation is more simple and straight-forward. It guarantees that there will be one address per role without any additional code.)

■ Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Low

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Path: /contracts/SwissBorgToken.sol : constructor();

Recommendation: implement zero address checks.

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

Found in: 856cce0

Status: Fixed (Revised commit: fbf8342)

L02. Unused Variable

Impact	Low
Likelihood	Low

The variable *totalChsbMigrated* is only incremented by migrated tokens in *migrate()*. This variable isn't used in other places of code.

Path: /contracts/ChsbToBorgMigrator.sol : totalChsbMigrated;

Recommendation: remove unused variable.

Found in: 856cce0

Status: Mitigated (*totalChsbMigrated* variable is needed for Client's frontend)

Informational

I01. Public Functions That Should Be External

Functions that are only called from outside the contract should be defined as external. External functions are much more gas efficient compared to public functions.

Path: /contracts/ChsbToBorgMigrator.sol : initialize();

Recommendation: make *initialize()* function external.

Found in: 856cce0

Status: Fixed (Revised commit: fbf8342)

I02. Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, *view* and *pure* functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all functions are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Path: /contracts/ChsbToBorgMigrator.sol;

Recommendation: consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts.

Found in: 856cce0

Status: Fixed (Revised commit: fbf8342)

I03. Invalid Unit

ether keyword should be used only when dealing with native tokens. ERC20 tokens aren't native tokens.

This can lead to wrong assumptions.

Path: /contracts/SwissBorgToken.sol : INITIAL_SUPPLY;

Recommendation: replace ether keyword with **10**18*.

Found in: 856cce0

Status: Fixed (Revised commit: fbf8342)

I04. Use of Hard-Coded Values

Hard-coded values are used in computations.

Paths:

/contracts/ChsbToBorgMigrator.sol : decimalsScale;

/contracts/ChsbToBorgMigrator.sol : migrate() - 1_000_000_000 *

10 ** 8 in first require;

Recommendation: convert these variables into constants.

Found in: 856cce0

Status: Fixed (Revised commit: fbf8342)

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.

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

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.

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

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.

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

Appendix 2. Scope

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

Initial review scope

Repository	https://github.com/SwissBorg/borg-token
Commit	856cce0b4c37514314e73fe6447b10caa8497f36
Whitepaper	not provided
Requirements	https://app.gitbook.com/invite/-M5RNylfUf-bGBi7u-Pi/ts71FlXZ4hPMds04v5fN (only the Developer section)
Technical Requirements	README.md
Contracts	File: contracts/ChsbToBorgMigrator.sol SHA3: 1c3827dc96c9e30084541f8257e13e8706b6f13e7b6739076a4c74b19c591dce File: contracts/SwissBorgToken.sol SHA3: 105a68b54eecbeec5be5b4a0614f31780694bcb1e0932e5a05049df74e5cb994

Second review scope

Repository	https://github.com/SwissBorg/borg-token
Commit	fbf834220649a7c2f9cd820f98638b40c82471a0
Whitepaper	not provided
Requirements	https://docs.swissborg.com/
Technical Requirements	https://docs.swissborg.com/developers/technical-documentation
Contracts	File: contracts/ChsbToBorgMigrator.sol SHA3: 6ee6bc86472489bd8ddf396fee42082d6f2086244dd359d18af7be58c3585210 File: contracts/SwissBorgToken.sol SHA3: 588c4033c7c3d585f096dc4ebd091cf5b15f03419541c7d23068552f4f31801d

Third review scope

Repository	https://github.com/SwissBorg/borg-token
Commit	77e8cb635040d3b8d9c3cf9217f3648ca037baee
Whitepaper	not provided
Requirements	https://docs.swissborg.com/

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

Technical Requirements	https://docs.swissborg.com/developers/technical-documentation
Contracts	File: contracts/ChsbToBorgMigrator.sol SHA3: 6ee6bc86472489bd8ddf396fee42082d6f2086244dd359d18af7be58c3585210 File: contracts/SwissBorgToken.sol SHA3: 61b7e0542adb813fce4ff6b83be95a972cf5e32ec2dd87e9215b675cb5874a52