HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: GateKeep
Date: Dec 12, 2022

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 GateKeep
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Gatekeep contract
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://gatekeep.xyz/
Changelog	05.12.2022 - Initial Review 15.12.2022 - 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 GateKeep (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/gatekeepxyz/contracts
Commit	1a1d5103329f50f4874abd78056c1a657ae0c2e8
Whitepaper	Link
Contracts	File: ./contracts/GatekeepBrink.sol SHA3: ede09ae5c9730ada368cc70a37c40b549711aae234e5dc1536abf783edcc6623 File: ./contracts/GatekeepGuard.sol SHA3: 81d961d5c65695dae5adadd12a8a10839ce2bee82b830309cd731d929cd107f0

Second review scope

Repository	https://github.com/gatekeepxyz/contracts
Commit	cb165883641de46fff1236efda899c5c16fb46c1
Whitepaper	Link
Contracts	File: ./contracts/GatekeepBrink.sol SHA3: ede09ae5c9730ada368cc70a37c40b549711aae234e5dc1536abf783edcc6623 File: ./contracts/GatekeepGuard.sol SHA3: 81d961d5c65695dae5adadd12a8a10839ce2bee82b830309cd731d929cd107f0

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 the 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 missed.

Code quality

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

- A few template code patterns were found.

Test coverage

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

- No tests were provided.

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: **9.6**.

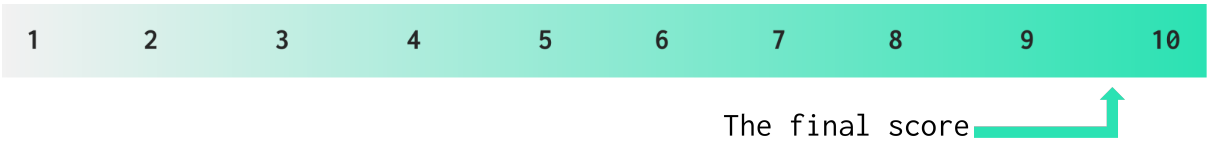

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
5 December 2022	4	2	1	0
14 December 2022	0	0	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 more 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.	Failed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
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.	Passed
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.	Failed
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.	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.	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 100%, 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

System Overview

GateKeep is a guarding system against theft with the following contracts:

- *GatekeepBrink* – a contract that allows to specify *userAddress* and *safeAddress* and has 3 functions to intercept a malicious transfer of ERC20/ERC721/ERC1155.
- *GatekeepGuard* – a contract that can call external functions to trigger the *GuardkeepBrink* contract.

Privileged roles

- The owner of *GatekeepBrink* can change *userAddress* and *safeAddress*
- The guard of *GatekeepBrink* can transfer token from the *userAddress* to the *safeAddress*
- The owner of *GatekeepGuard* can call the functions to trigger the *GatekeepBrink* contract

Risks

- The transaction may be directed to a controlled address, but this address may not be whitelisted, this would result in an unwanted transaction to the *safeAddress*.
- According to the use case, the GatekeepBrink contract should have an allowance for spending users' funds. Users of the contracts **must ensure** that they are owners of the GatekeepBrink contract and target addresses are under their control. Otherwise, the **secureness** of funds **is not guaranteed**.

Findings

Critical

No critical severity issues were found.

High

1. Funds Lock

The contract can accept native coins but lacks a withdrawal mechanism.

Funds can be stuck on the contract.

Path: ./GatekeepBrink.sol : receive(), fallback()

Recommendation: Remove *receive()* and *fallback()* functions or provide mechanisms for native coins withdrawal.

Status: Fixed (Revised commit: 37753bc)

Medium

1. Best Practice Violation

The function does not use *SafeERC20* library for checking the result of ERC20 token transfer. Token may not follow ERC20 standard and return false in case of transfer failure or not returning any value at all.

Path: ./GatekeepBrink.sol : intercept20()

Recommendation: Use *SafeERC20* library to interact with the tokens safely.

Status: Fixed (Revised commit: 37753bc)

2. Inconsistent Data

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

The functions do not emit events on change of important values.

Paths: ./GatekeepBrink.sol : intercept20(), intercept721(), intercept1155();
./GatekeepGuard.sol : triggerIntercept20(), triggerIntercept721(), triggerIntercept1155();

Recommendation: Emit events on critical state change.

Status: Mitigated (with Customer notice)

Low

1. State Variable Can Be Declared Immutable

Variable's *gateKeepGuard* value is set in the constructor. This variable can be declared immutable.

This will lower Gas taxes.

Path: ./GatekeepBrink.sol; ./GatekeepGuard.sol;

Recommendation: Declare mentioned variables as immutable.

Status: Fixed (Revised commit: 37753bc)

2. **Floating Pragma**

   The project uses floating pragma solidity >=0.8.0.

   Path: ./GatekeepBrink.sol : pragma; ./GatekeepGuard.sol : pragma;

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

   Status: Fixed (Revised commit: cb16588)

3. **Missing Zero Address Validation**

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

   This can lead to unwanted external calls to 0x0.

   Path: ./GatekeepBrink.sol : setUserAddress(), setSafeAddress();

   Recommendation: Implement zero address checks.

   Status: Fixed (Revised commit: 37753bc)

4. **Style Guide Violation**

   The provided projects should follow the official guidelines.

   Path: ./GatekeepBrink.sol; ./GatekeepGuard.sol;

   Recommendation: Follow the official Solidity guidelines.

   Status: Fixed (Revised commit: 37753bc)

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 by the best industry practices at the date 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 to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.