HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: VENOM BLOCKCHAIN HOLDING LIMITED
Date: June 21th, 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 VENOM BLOCKCHAIN HOLDING LIMITED
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	TIP-3.1 token
Platform	TVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Timeline	03.05.2022 - 21.05.2022
Changelog	13.05.2022 - Initial Review 20.05.2022 - Second Review 21.06.2022 - Third 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 VENOM BLOCKCHAIN HOLDING LIMITED (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/broxus/ton-eth-bridge-token-contracts
Commit:
6503e61880c03fb45741c177c25f4955dca5f0df
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://docs.everscale.network/standard/TIP-3.1

JS tests: Yes
Contracts:
File: ./contracts/abstract/TokenRootBase.sol
SHA3: 8683e77bb154cf52577d7d4ae6173bf74677b9d9c272bb2aa50b45b6

File: ./contracts/abstract/TokenRootBurnableByRootBase.sol
SHA3: af84bc2113f9e93d9efddd2a22658604e2516228dd1c183ea9aea51e

File: ./contracts/abstract/TokenRootBurnPausableBase.sol
SHA3: ca17173282559d4b3500cc60c3661db83c5a908eb11f72c21f142976

File: ./contracts/abstract/TokenRootDisableableMintBase.sol
SHA3: 325f0ee1d941231521715a321683a1f4aa366ccff25d45e294bdbde2

File: ./contracts/abstract/TokenRootTransferableOwnershipBase.sol
SHA3: 7e80091ff8d68fa10bcb7934b964f9b583e2f8293c7933a09f703f76

File: ./contracts/abstract/TokenWalletBase.sol
SHA3: 41b89e1bf69014613264e32b9a1f22c8fc0b7126ba99e8b00624f884

File: ./contracts/abstract/TokenWalletBurnableBase.sol
SHA3: 11ce2b6832b78a7afaa8652d769d859fa683871dd302898572124de0

File: ./contracts/abstract/TokenWalletBurnableByRootBase.sol
SHA3: 60dbd2afc16f163d01ac4ebde04436bca0dfe4fa35f21cddbf7b403e

File: ./contracts/abstract/TokenWalletDestroyableBase.sol
SHA3: 9796facba05c46b8b7fa1e70cfabb6eb7a0ac6cd73cb771afd83e324

File: ./contracts/additional/Selector.sol
SHA3: 5d446d0b89bc2d8bbebdc51bc32a92d92e0aced4386cdfe37b361f02

File: ./contracts/additional/TokenFactory.sol
SHA3: 8d412d428b662f5eaaa7480dc672e57916434fa1f2c80d49033b9e80

File: ./contracts/additional/Wallet.sol
SHA3: 55c0a847fe05e140dc4c40b97f8c81d14b3b29f1bfe9d0e885f7e101

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

File: ./contracts/libraries/TokenErrors.sol
SHA3: 111c708859e327e0bf946acdb78762f5a545255541a5aa3a70a6233f

File: ./contracts/libraries/TokenGas.sol
SHA3: af55e953ab7d29dd5a4c839b9bdb029b1a596a54e86d5307fe0cf3be

File: ./contracts/libraries/TokenMsgFlag.sol
SHA3: e1c1534e3809d8cc4008bbc50d247ba0207c50497028460bb5470042

File: ./contracts/TokenRoot.sol
SHA3: fc8629520ce1476ead1beea3d12a01a60c1bbd81de62b4ac151c66c4

File: ./contracts/TokenRootUpgradeable.sol
SHA3: 6f898f607d4b2b1d0e36a9ae6cc5687a2b47bb964dc47bdb06f0435a

File: ./contracts/TokenWallet.sol
SHA3: d07269201988c13531c1fcaf005a733bd57e6f3db4b3f7c547160ca9

File: ./contracts/TokenWalletPlatform.sol
SHA3: cc966b04c7d437ccca7668f4ba4d1b2d92ab806d2b8c515869f0195e

File: ./contracts/TokenWalletUpgradeable.sol
SHA3: 9d5b48114094f977eddea284964008cff6c2e69fde6de2e313a92a07

Second review scope

Notice: No scope updation, absorbing Customer comments.

Second review scope

Repository:
https://github.com/broxus/ton-eth-bridge-token-contracts
Commit:
6503e61880c03fb45741c177c25f4955dca5f0df
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://docs.everscale.network/standard/TIP-3.1

Type: Functional requirements
Link: https://docs.google.com/document/d/1u34ovYdEtr7OVkHleT2LR8HmsQThtrjU9xlddlIkwTM

Notice: No scope contracts updation, absorbing new documentation.

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 assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

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 methodology.

Documentation quality

The Customer provided good functional requirements and no technical description. The total Documentation Quality score is **5** out of **10**.

Code quality

The total CodeQuality score is **7** out of **10**. Some code duplications were included. Unit Tests do not process cases of interaction by several users and the root owner.

Architecture quality

The architecture quality score is **5** out of **10**. The code contains hardcoded nondeclarative function and interface ids on which upgradable contracts lean. The system is overwhelmed with low-level code that could be declared in special functions or modifiers.

Security score

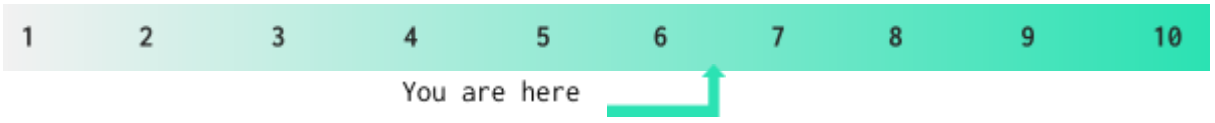
Warning: The security of those contracts depends on off-chain part of the code that is out of the audit scope. Review Risks section below for details.

As a result of the audit, security engineers found **2** medium and **3** low severity issues. The security score is **7** 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: **6.6**.



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

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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.	Failed
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.	Failed
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 destroyed until it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Not Relevant
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Not Relevant
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Failed
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	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Passed
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.	Passed
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	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.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Failed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Failed
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.	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
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
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, that may be changed in the future.	Passed

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

System Overview

TIP-3.1 contracts for TON/ETH bridge is a system of contracts that implements the fungible token interface:

- *TokenRoot (TokenRootUpgradeable)* – contract that implements such features for token platform owner:
  - burning tokens from user wallet
  - minting tokens to user wallet
  - stop burning ability from user wallet
  - stop minting ability to user wallet
  - pause burning ability by users
  - transfer ownership with triggering callback of specified
  - upgrade wallet code and code of the root contract
- *TokenWallet (TokenWalletUpgradeable and TokenWalletPlatform)* – contract that implements such features for token platform users:
  - burning tokens
  - transfer the TIP-3.1 token to another wallet

Privileged roles

- The owner of the *TokenRoot (TokenRootUpgradeable)* contract can arbitrarily change the balance of selected users and pause burning assets by users.

Risks

This is only part of the whole project, and a valuable security part is entrusted to dApp. Ensure using the official dApp and corresponding audit is provided.
In another way, any transactions may be executed partially (look *Missing Gas management* critical issue for details).

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

Findings

Critical

1. Missing Gas management

The amount of Gas proceeded to internal calls should cover all calculations, the storage cost of the recipient, and onBounce callback execution if the corresponding flag is provided. If not enough Gas proceeds, transactions may be done partially, and data consistency may be corrupted.

This can lead to funds loss because of partial transfer execution, unactual total amount value, and as a result, problems with burning and minting tokens that could block the next transfers.

Contract: TokenRootBase.sol

Functions: acceptBurn, _mint, onBounce

Contract: TokenRootBurnableByRootBase.sol

Function: burnTokens

Contract: TokenRootTransferableOwnershipBase.sol

Function: transferOwnership

Contract: TokenWalletBase.sol

Functions: transfer, transferToWallet, acceptMint, acceptTransfer, onBounce, _burn, _deployWallet

Contract: TokenRoot.sol

Function: _deployWallet

Contract: TokenRootUpgradeable.sol

Functions: requestUpgradeWallet, _deployWallet

Contract: TokenWalletUpgradeable.sol

Functions: upgrade, _deployWallet

Recommendation: calculate how much Gas is needed for each message chain and check if it is enough Gas granted.

Status: Mitigated (with a customer notice)

High

No high severity issues were found.

Medium

1. Wrong data packing

According to comments, upgrade data should be packed in another way. Variable *naming* is empty, it should contain *name_* and *symbol_*.

This can lead to wrong data unpacking on code upgrades.

Contract: TokenRootUpgradeable.sol

Function: upgrade

Recommendation: check how data should be packed and update comments or change the code.

Status: Reported

2. **Mixing reserving patterns**

Several reserving patterns are mixed through the code without explanation.

Patterns:

- *tvm.rawReserve(_targetBalance(), 0);*
- *tvm.rawReserve(_reserve(), 0);*
- *tvm.rawReserve(_reserve(), 2);*
- *tvm.rawReserve(address(this).balance - msg.value, 0);*
- *tvm.rawReserve(address(this).balance - msg.value, 2);*

Recommendation: separate these low-level calls to a special library, introduce documentation of why the patterns are mixed.

Status: Reported

Low

1. **Missing variable visibility**

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Contract: TokenRootBase.sol

Variable: totalSupply_

Contract: TokenRootBurnableByRootBase.sol

Variable: burnByRootDisabled_

Contract: TokenRootBurnPausableBase.sol

Variable: burnPaused_

Contract: TokenRootDisableableMintBase.sol

Variable: mintDisabled_

Contract: TokenWalletBase.sol

Variable: balance_

Contract: TokenRootUpgradeable.sol

Variable: walletVersion_

Contract: TokenWalletUpgradeable.sol

Variables: version_, platformCode_

Recommendation: explicitly define visibility for all state variables.

Status: Reported

2. **Outdated compiler version**

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

Contracts: TokenRootBase.sol, TokenRootBurnableByRootBase.sol, TokenRootBurnPausableBase.sol, TokenRootDisableableMintBase.sol, TokenRootTransferableOwnershipBase.sol, TokenWalletBase.sol, TokenWalletBurnableBase.sol, TokenWalletBurnableByRootBase.sol, TokenWalletDestroyableBase.sol, Selector.sol, TokenFactory.sol, Wallet.sol, TokenErrors.sol, TokenGas.sol, TokenMsgFlag.sol, TokenRoot.sol, TokenRootUpgradeable.sol, TokenWallet.sol, TokenWalletPlatform.sol, TokenWalletUpgradeable.sol

Recommendation: use a recent version of the Solidity compiler.

Status: Reported

3. **Floating Pragma**

The contracts use floating pragma >= 0.57.0.

Contracts: TokenRootBase.sol, TokenRootBurnableByRootBase.sol, TokenRootBurnPausableBase.sol, TokenRootDisableableMintBase.sol, TokenRootTransferableOwnershipBase.sol, TokenWalletBase.sol, TokenWalletBurnableBase.sol, TokenWalletBurnableByRootBase.sol, TokenWalletDestroyableBase.sol, Selector.sol, TokenFactory.sol, Wallet.sol, TokenErrors.sol, TokenGas.sol, TokenMsgFlag.sol, TokenRoot.sol, TokenRootUpgradeable.sol, TokenWallet.sol, TokenWalletPlatform.sol, TokenWalletUpgradeable.sol

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

Status: Reported

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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 audit cannot guarantee the explicit security of the audited smart contracts.