HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Web3 Bazaar
Date: July 27th, 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 Web3 Bazaar
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU Noah Jelich | Senior Solidity SC Auditor at Hacken OU
Type	Escrow, trading
Platform	EVM
Network	Ethereum, Polygon
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://web3bazaar.org/
Timeline	12.07.2022 - 27.07.2022
Changelog	19.07.2022 - Initial Review 27.07.2022 - 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 Web3 Bazaar (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/Web3bazaar/mainnet-contracts/
Commit:
f7f965fcfd196ca5f24feb7514f1afdabe7f6568
Technical Documentation:
Type: Technical description
https://docs.web3bazaar.org/

Type: Functional requirements
https://docs.web3bazaar.org/

Integration and Unit Tests: No
Contracts:
File: ./contracts/src/web3bazaar-batch.sol
SHA3: 97bd5a70fdfed4d370c5e41a2c06c5166a145b96c1fbf92ee802a57bddf732fc

Second review scope

Repository:
https://github.com/Web3bazaar/mainnet-contracts/
Commit:
010395a48c829aa1a32a37f8ac350069643a76e4
Technical Documentation:
Type: Technical description
https://docs.web3bazaar.org/

Type: Functional requirements
https://docs.web3bazaar.org/

Integration and Unit Tests: No
Contracts:
File: ./contracts/src/Web3BazaarEscrow.sol
SHA3: d93d6944bb97daaa2ae4f128975c7c1a73cc2ac3de755a1fb692ce3442534c35

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 superficial functional and technical requirements.
The total Documentation Quality score is **7** out of **10**.

Code quality

The total Code Quality score is **9** out of **10**. The code follows official language style guides and is generally covered with unit tests. Some negative test cases are missing.

Architecture quality

The architecture quality score is **10** out of **10**. Clean and clear architecture and well-configured development environment.

Security score

As a result of the audit, the code does not contain any 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**.

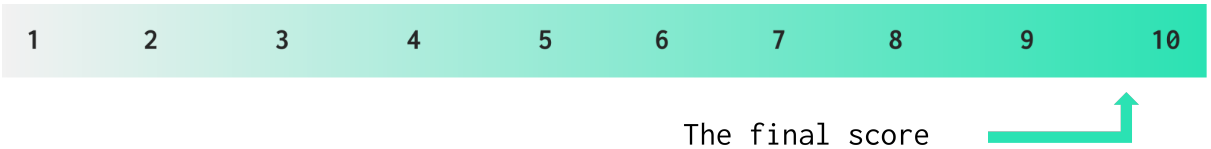

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
19 July 2022	4	2	2	0
27 July 2022	0	0	0	0

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.	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.	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 it is 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.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	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 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.	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.	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.	Passed
Gas Limit and	**Custom**	Transaction execution costs should not	Passed

Loops		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.	
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 100%, 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, that may be changed in the future.	Passed

System Overview

Web3Bazaar - The fully permissionless P2P exchange supported by non-custodial escrow smart contracts to trade all assets on EVM chains. Web3 Bazaar has the following contracts:

- *Web3BazaarEscrow* — is a smart contract that proceeds with its validation and checks if all the assets belong to the wallet addresses provided and if the creator gave the necessary permissions for the smart contract to perform the swap. If an issue is detected, the trade is not submitted, and the code returns an error as described in the ERROR LIST at the end of this Readme.

- *MultiAccessControl* - is a contract for managing contract owners.

Privileged roles

- The *Owner* of the *Escrow* contract will be the Owner of running all the *admin* level functions.
- *Trade Creator* - provides the terms (assets' smart contract addresses and counterparty wallet address.
- *Executer Address* - wallet address of the trade counterparty.
- Executer - trade counterparty.

Risks

- In case of an admin keys leak, an attacker can get access to all funds that belong to *Web3BazaarEscrow* contract and will be able to send them to any address. Attackers will be able to stop/start trades at any time.
- Assets are not transferred to the contract during a deal initiation process. Only allowances are verified. There is no guarantee that assets are still owned by the account when the deal is finalized.

Findings

Critical

No critical severity issues were found.

High

1. **Insufficient balance.**

   Assets are not transferred to the contract during a deal initiation process. Only allowances are verified.

   There is no guarantee that assets are still owned by the account when the deal is finalized.

   File: ./contracts/Web3BazaarEscrow.sol

   Contract: Web3BazaarEscrow

   Functions: executeTrade, startTrade

   Recommendation: Transfer assets from the deal initiator to ensure balance sufficiency.

   Status: Mitigated (with Customer notice)

2. **Denial of Service.**

   Iterating over large structures and performing external calls in loops may lead to out-of-Gas exceptions.

   Iteration over *store._traders* array and execute 3 external calls (swapERC20, swapERC721, swapERC1155) for each element inside *executeTrade* function.

   This can lead to out-of-Gas exceptions.

   File: ./contracts/Web3BazaarEscrow.sol

   Contract: Web3BazaarEscrow

   Function: executeTrade

   Recommendation: Limit the number of possible items that are exchanged during 1 trade.

   Status: Fixed (Revised commit: `010395a48c829aa1a32a37f8ac350069643a76e4`)

Medium

1. **Unchecked return value.**

   The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behavior in the subsequent program logic.

The *swapERC20* function of the escrow contract, ignores the *IERC20(tokenAddress).transferFrom(from,to,amount)* return value.

File: ./contracts/Web3BazaarEscrow.sol

Contract: Web3BazaarEscrow

Function: swapERC20

Recommendation: Use *SafeERC20*, or ensure that the *transfer*/*transferFrom* return value is validated.

Status: Fixed (Revised commit: 010395a48c829aa1a32a37f8ac350069643a76e4)

2. **Not suitable data type.**

The *mapping (address => uint8) internal _owners;* uses uint8 to store a value that can be expressed as boolean.

File: ./contracts/Web3BazaarEscrow.sol

Contract: MultiAccessControl

Function: executeTrade

Recommendation: Use *bool* instead of *uint8*

Status: Fixed (Revised commit: 010395a48c829aa1a32a37f8ac350069643a76e4)

Low

1. **Unused variable.**

Expressions that are tautologies or contradictions are detected.

Function verifyTradeIntegrity contains a tautology or contradiction. tokenType >= 0 is always true.

File: ./contracts/Web3BazaarEscrow.sol

Contract: Web3BazaarEscrow

Function: verifyTradeIntegrity

Recommendation: Fix the incorrect comparison by changing the value type or the comparison.

Status: Fixed (Revised commit: 010395a48c829aa1a32a37f8ac350069643a76e4)

2. **Unused variable.**

Unused variables should be removed from the contracts. Unused variables are allowed in Solidity and do not pose a direct security issue. It is best practice to avoid them as they can cause an increase in computations (and unnecessary Gas consumption) and decrease readability.

The *_symbol* state variable is never used.

File: ./contracts/web3bazaar-batch.sol

Contract: Web3BazaarEscrow

Recommendation: Remove unused variable.

Status: Fixed (Revised commit: 010395a48c829aa1a32a37f8ac350069643a76e4)

3. **Function can be declared external.**

   public functions that are never called by the contract should be declared “external” to save Gas.

   In order to save Gas, public functions that are never called in the contract should be declared as external.

   File: ./contracts/Web3BazaarEscrow.sol

   Contract: Web3BazaarEscrow

   Functions: addOwnership, removeOwnership, cancelTrader, getTrade, executeTrade, startTrade, tradePerUser, balanceOfBatch, set Approval For All

   Recommendation: Use the external attribute for functions never called from the contract.

   Status: Fixed (Revised commit: 010395a48c829aa1a32a37f8ac350069643a76e4)

4. **Native TradeType is never used.**

   Unused enum values should be removed from the contracts. It is best practice to avoid them as they decrease readability.

   TradeType.Native is never used in the smart contract.

   File: ./contracts/Web3BazaarEscrow.sol

   Contract: Web3BazaarEscrow

   Recommendation: Consider removing unused enum value.

   Status: Fixed (Revised commit: 010395a48c829aa1a32a37f8ac350069643a76e4)

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.