HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: OTCOnline
Date: January 17, 2023

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

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

Name	Smart Contract Code Review and Security Analysis Report for OTCOnline
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	OTC, Marketplace
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://otconline.io
Changelog	09.12.2022 - Initial Review 28.12.2022 - Second Review 16.01.2023 - Third Review 17.01.2023 - Forth 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 OTCOnline (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/otconline/smart-contracts
Commit	f20d1047c81e4de442c91d489ed537b1d4041a1b
Whitepaper	Not provided.
Functional Requirements	OTCOnline Smart Contract’s Technical Documentation.pdf
Technical Requirements	OTCOnline Smart Contract’s Technical Documentation.pdf
Contracts	File: ./IERC20.sol SHA3: 1ef8cb258d8bd1f1dea44027b4bd4e25cfd6172eaba93f9154554773e5dd8417 File: ./SafeDeal.sol SHA3: cca6c2f961419cd80f2ffc608e4e4a44cd09da9086b61c9ffcbb6bfe64ac2ff4

Second review scope

Repository	https://github.com/otconline/smart-contracts-v2
Commit	356e4fa979b9950d163b57e0eee2a9c3bebd69c2
Whitepaper	https://medium.com/@otconlineio/otconline-io-2ffcb5528b6b
Functional Requirements	OTCOnline Smart Contract’s Technical Documentation_v1.pdf
Technical Requirements	OTCOnline Smart Contract’s Technical Documentation_v1.pdf
Contracts	File: ./contracts/Moderators.sol SHA3: 4f80ad85c1733a58c1ea33d56d6d86d14dc91a4b799b30cb21777eb745b15bee File: ./contracts/SafeDeal.sol SHA3: 37a770d0c4a29c0d3070135789dde4d4fd63fdfa83a0e38c5887d2853f9d4bbe

Third review scope

Repository	https://github.com/otconline/smart-contracts-v2
Commit	27f472797c697fceefccfb8bd2f0e2bf2cd32b80
Whitepaper	https://medium.com/@otconlineio/otconline-io-2ffcb5528b6b
Functional Requirements	OTCOnline Smart Contract's Technical Documentation_v2_1_1.pdf
Technical Requirements	OTCOnline Smart Contract's Technical Documentation_v2_1_1.pdf
Contracts	File: ./contracts/Moderators.sol SHA3: 0f0ae3c2f925a99192457c04c1bf0706b2134afab55c2721edd4a281b0141902 File: ./contracts/SafeDeal.sol SHA3: 1074b0fbfe3593102255f4b2f3db4f02786887a8fd1871e9e8b6170e5425f40b

Forth review scope

Repository	https://github.com/otconline/smart-contracts-v2
Commit	00f6585f373920c1c35402e3d83f3915712d446a
Whitepaper	https://medium.com/@otconlineio/otconline-io-2ffcb5528b6b
Functional Requirements	OTCOnline Smart Contract's Technical Documentation_v2_1_1.pdf
Technical Requirements	OTCOnline Smart Contract's Technical Documentation_v2_1_1.pdf
Contracts	File: ./contracts/Moderators.sol SHA3: 566175a99a821da5207433635a637ae259ca80a042df8bd6aaad018474f3a6bd File: ./contracts/SafeDeal.sol SHA3: 74f623d47f2b77127d47f6373ee01e198a60a7da7f220c3d094325d137ca3961

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 assets 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 **10** out of **10**.

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

Code quality

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

- The development environment is configured.
- The code follows official language style guides and best practices.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Contract properties are covered with tests.

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**.

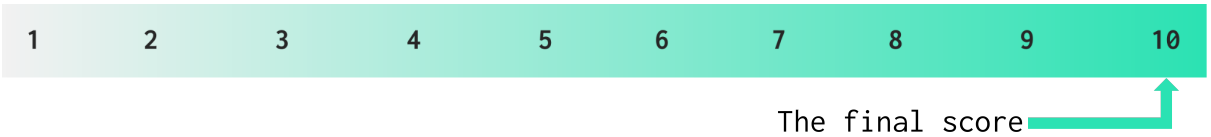

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
7 December 2022	12	9	1	2
28 December 2022	8	4	0	0
16 January 2023	3	1	0	0
17 January 2023	0	0	0	0

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.	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.	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 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.	Not Relevant
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.	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.	Not Relevant
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.	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, which may be changed in the future.	Passed

System Overview

The OTCOnline SafeDeal smart contract is used to transfer funds between users with a temporary hold, as well as to distribute funds between users through a referral program.

It allows for the full automation and security of the process of concluding a deal between the seller and the buyer.

Users can easily and securely make deals, buy, and sell cryptocurrency and other assets without the involvement of third parties within the Web3 ecosystem.

Privileged roles

- Owner:
  - add/remove Moderators
  - count balance
  - withdraw service fee
- Moderator:
  - complete/cancel disputed Deals
- Signer:
  - validate correction of the signatures.

Risks

- The system relies on the security of the private keys of the privileged roles that can be used to assign and revoke roles that can affect the execution flow.

Findings

■■■■ Critical

1. Data Consistency

In the *start()* function, there is no check to prevent the reuse of an *id* with an active Deal.

Malicious actors can perform an action that overrides other users' deals, resulting in users' funds being locked in the contract.

Path: ./SafeDeal.sol : start()

Recommendation: Add validation to prevent the use of an *id* with an active Deal.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

2. Invalid Calculations

The contract contains a floating pragma version with a wide range of Solidity versions: >=0.4.22 <0.9.0

In the *start()* function, there are no checks for an overflow and addition done in *transferFrom()* function will not revert with overflow in situations when the contract is compiled with Solidity version < 0.8.0.

A malicious actor can perform an attack where they overflow the addition in the transferFrom() function, transfer zero BUSD to the contract, perform a deal with themselves, and extract all the funds from the contract.

Path: ./SafeDeal.sol : start()

Recommendation: Use SafeMath OpenZeppelin library or use Solidity version >= 0.8.0 to prevent overflows.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

■■■ High

1. Requirements Violation

In the *start()* function, there is no check that prevents from changing the function parameters that are provided from the off-chain system.

A user can change the referrer address, service fee amount, referrer fee amount. This violates the requirements of 3% service fee or 10% referral fee described in the documentation.

Path: ./SafeDeal.sol : start()

Recommendation: Either the documentation should be updated, or the *start()* function should be updated to prevent manipulation of the function parameters.

For reference, ECDSA signatures signed by the off-chain system can be used.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

Medium

1. **Best Practice Violation - Lock of Native Tokens**

   It is considered following best practices to avoid unclear situations and prevent common attack vectors.

   The contract accepts native tokens in multiple payable functions, but there are no mechanisms for withdrawals.

   This may lead to native coins being locked in the contract.

   Path: ./SafeDeal.sol : start(), completeByBuyer(), completeByModerator(), cancelByModerator(), withdraw()

   Recommendation: Remove *payable* mutability modifier.

   Status: Fixed
   (revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

2. **Inefficient Gas Model - Moderators Array**

   It is considered to avoid inefficient Gas models.

   Storing moderators in the *address[]* array and iterating through the loop to check if the address is a moderator is Gas inefficient.

   Path: ./SafeDeal.sol: _moderators, onlyModerator(), addModerator(), removeModerator()

   Recommendation: Consider using a *mapping(address => bool)* for storing moderators, or using the AccessControl library from OpenZeppelin to reduce Gas inefficiency.

   Status: Fixed
   (revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

3. **Inefficient Gas Model - Array in Events**

   It is considered to avoid inefficient Gas models.

   Emitting events with an array of moderators in case of each state change is Gas inefficient.

   The Gas cost of the operation will increase with the size of the array.

   In most cases, returning the entire array is wasteful and redundant.

Path: ./SafeDeal.sol: ModeratorAdded, ModeratorRemoved

Recommendation: Events emitted should contain data only about the moderator for whom an action was performed.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

4. **Best Practice Violation - Unchecked Transfer**

An unchecked *transferFrom()* function is used in the contract.

Tokens that do not follow the ERC20 standard (such as USDT) may return false in the case of a transfer failure, or they may not return any value at all.

This may lead to denial of service vulnerabilities when interacting with non-standard ERC20 tokens.

Path: ./SafeDeal.sol: start(), completeByBuyer(), completeByModerator(), cancelByModerator(), withdraw()

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

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

5. **Unscalable Functionality - Code Duplication**

It is considered that smart contract systems should be easily scalable.

Duplicated logic takes more Gas for deployment and makes further development difficult. The same code patterns are used in the functions completeByBuyer() and completeByModerator().

This may lead to new issues during further development and higher Gas expenses during deployment and interactions.

Path: ./SafeDeal.sol: completeByBuyer(), completeByModerator()

Recommendation: Move the code patterns to internal functions.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

6. **Best Practice Violation - Checks-Effects-Interactions Pattern**

During the function execution, some state variables are updated after the external calls.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during implementation of new functionality.

Path: ./SafeDeal.sol: start(), completeByBuyer(), completeByModerator(), cancelByModerator(), withdraw(), closeDeal

Recommendation: Common best practices should be followed, functions should be implemented according to the Checks-Effects-Interactions pattern.

Status: Fixed
(revised commit: 00f6585f373920c1c35402e3d83f3915712d446a)

7. **Best Practice Violation - Misuse of approve/transferFrom**

   In all functions that transfer BUSD token from the contract to an external address, the code pattern used is *approve() + transferFrom()*.

   This is redundant and inefficient in terms of Gas consumption, as *transfer()* will perform the same operation.

   Path: ./SafeDeal.sol: completeByBuyer(), completeByModerator(), cancelByModerator(), withdraw()

   Recommendation: Use the SafeERC20 library's *safeTransfer()* function to transfer funds from the contract to the users.

   Status: Fixed
   (revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

8. **Inefficient Gas Model - Loop of Storage Interactions**

   It is considered to avoid inefficient Gas models.

   The function *getBalance()* contains a loop with costly storage interactions when counting the number of *reserved* tokens.

   This may lead to high transaction costs.

   Path: ./SafeDeal.sol: getBalance()

   Recommendation: Design the project to consume a limited amount of Gas regardless of the stored data and the number of users.

   Consider storing the *reserved* value in a storage variable and removing the accounting loop.

   Status: Fixed
   (revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

9. **Inefficient Gas Model - Loop of Storage Interactions**

   It is considered to avoid inefficient Gas models.

   The internal function *deleteDeal()* contains a loop with costly storage interactions when checking if specific *id* exists in the array.

   This may lead to high transaction costs.

   Path: ./SafeDeal.sol: deleteDeal()

Recommendation: Consider using the *EnumerableSet* library from OpenZeppelin, or consider refactoring the *getBalance()* and *deleteDeal()* functions to remove the need for _ids array.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

10. Best Practice Violation - Unchecked Transfer

An unchecked *transfer()* function is used in the contract.

Tokens that do not follow the ERC20 standard (such as USDT) may return false in the case of a transfer failure, or they may not return any value at all.

This may lead to denial of service vulnerabilities when interacting with non-standard ERC20 tokens.

Path: ./SafeDeal.sol: closeDeal(), withdraw()

Recommendation: Use the *SafeERC20* library's *safeTransfer()* function to interact with tokens safely.

Status: Fixed
(revised commit: 27f472797c697fceefccfb8bd2f0e2bf2cd32b80)

11. Contradiction - Unnecessary Access Control

Limiting access to storage values for off-chain reads is redundant.

The private storage values of any Smart Contract are easily accessible off-chain, and reading from them cannot be blocked.

The *@dev* comment of the function *getBalance()* contradicts the way storage variables are accessible off-chain.

Path: ./SafeDeal.sol: getBalance()

Recommendation: Consider removing redundant access control limitations from the *getBalance()* view function and updating the documentation accordingly.

Status: Fixed
(revised commit: 27f472797c697fceefccfb8bd2f0e2bf2cd32b80)

12. Best Practice Violation - Signature Replay

The signatures do not include the CHAIN_ID value or the contract address.

This may lead to the Signature Replay Attacks when the same signature is used on different chains or in different instances of the SafeDeal contract on the BNB chain.

Path: ./SafeDeal.sol: start()

Recommendation: Consider using EIP-712 standard.

Status: Fixed
(revised commit: 27f472797c697fceefccfb8bd2f0e2bf2cd32b80)

Low

1. Floating Pragma

Locking the pragma helps to ensure that contracts are not accidentally deployed using an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths: ./IERC20.sol ./SafeDeal.sol

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

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

2. Misused Payable Modifier

When interacting with ERC20 tokens, there is no need to use a *payable* modifier for functions and variables.

Only when a function or address is expected to receive a native coin (BNB), a *payable* modifier should be used.

In the entire SafeDeal contract, there is no place where interaction with native BNB coin is expected.

Path: ./SafeDeal.sol : Deal, start(), completeByBuyer(), completeByModerator(), cancelByModerator(), withdraw()

Recommendation: Consider removing the *payable* modifier from contracts that do not interact with native coin.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

3. Style Guide Violation

The project should follow the official code style guidelines.
Inside each contract, library, or interface, use the following order:

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

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)

- external
- public
- internal
- private

Within a grouping, place the view and pure functions at the end.
Some contracts are not formatted correctly.

Path: ./SafeDeal.sol

Solidity style guidance defines a naming convention that should be followed. Function argument names and local variable names should use mixedCase.

Path: ./SafeDeal.sol: service_fee, referer_fee

Recommendation: The official Solidity style guidelines should be followed.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

4. **Unfinished NatSpec**

It is recommended that the code should be kept clean and properly documented with NatSpec. There are multiple functions, structs, and public storage variables that are missing proper NatSpec documentation.

Paths: ./SafeDeal.sol ./Moderators.sol

Recommendation: NatSpec documentation best practices should be followed. For reference:
https://docs.soliditylang.org/en/v0.8.17/natspec-format.html#documentation-example
https://dev.to/perelynsama/natspec-the-right-way-to-comment-ethereum-smart-contracts-1b0c

Status: Fixed
(revised commit: 27f472797c697fceefccfb8bd2f0e2bf2cd32b80)

5. **State Variables that Could Be Declared as Immutable**

There are variables in the contract that can be declared as immutable to save Gas.

Path: ./SafeDeal.sol : _token, _owner

Recommendation: Variables that do not change after construction execution should be declared as immutable.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

6. Explicit Size

Across the contract, uint is used for uint256 variables. Using uint256 brings readability and consistency to the code.

Path: ./SafeDeal.sol : withdraw()

Recommendation: Use an explicit size of uint.

Status: Fixed
(revised commit: 00f6585f373920c1c35402e3d83f3915712d446a)

7. Variable Name Contradiction

Spelling error in the multiple variables. "*referer*" => "*referrer*"

Path: ./SafeDeal.sol : referer, referer_fee, Deal.referer

Recommendation: Spellings should be fixed.

Status: Fixed
(revised commit: 27f472797c697fceefccfb8bd2f0e2bf2cd32b80)

8. Unindexed Events

Having indexed event parameters makes it easier to search for these events using indexed event parameters as filters.

Path: ./SafeDeal.sol

Recommendation: The "indexed" keyword should be used for the event parameters.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

9. Missing Zero Address Validation

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

This can lead to unwanted external calls to 0x0.

Path: ./SafeDeal.sol : start(), addModerator(), removeModerator(), withdraw(), setSigner()

Recommendation: Implement zero address validations.

Status: Fixed
(revised commit: 27f472797c697fceefccfb8bd2f0e2bf2cd32b80)

10. Redundant Function Parameter

In the *start()* function, the *buyer* parameter is unnecessary.

Function logic requires that the *buyer* is equal to *msg.sender*, indicating that *msg.sender* can be used in the function flow.

Path: ./SafeDeal.sol : start()

Recommendation: Consider using *msg.sender* as a replacement for *buyer* parameter.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

11. Code Consistency

It is best practice to write code uniformly.

There is no consistency in order of visibility modifiers in functions.

It is considered best practice to put visibility modifier first.

Path: ./SafeDeal.sol : addModerator(), removeModerator(), getBalance(), withdraw()

Recommendation: Be consistent with the approach to visibility modifiers and always put them first.

Status: Fixed
(revised commit: 27f472797c697fceefccfb8bd2f0e2bf2cd32b80)

12. Function that Should Be Declared as View

The *getBalance()* function should be declared as a *view*.

Checking the collected service fee should not require spending Gas on a transaction or emitting events with the data.

Accessing on-chain data by off-chain systems should be done using view or pure functions.

Path: ./SafeDeal.sol : getBalance()

Recommendation: Consider updating the *getBalance()* function to *view* and remove state changes from its body.

Status: Fixed
(revised commit: 356e4fa979b9950d163b57e0eee2a9c3bebd69c2)

13. Functions that Can Be Declared External

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

Paths: ./SafeDeal.sol : start(), completeByBuyer(), completeByModerator(), cancelByModerator(), setSigner(), getBalance(), withdraw() ./Moderators.sol : addModerator(), removeModerator()

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

Status: Fixed
(revised commit: 00f6585f373920c1c35402e3d83f3915712d446a)

14. Missing SPDX-License-Identifier

Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.

Paths: ./SafeDeal.sol ./Moderators.sol

Recommendation: Add SPDX-License identifiers.

Status: Fixed
(revised commit: 00f6585f373920c1c35402e3d83f3915712d446a)

15. Missing Event Emit

The contract does not emit any event after changing an important value. It is recommended to emit events after changing important values to track changes off-chain.

Path: ./SafeDeal.sol : setSigner()

Recommendation: Consider emitting an indexed event after the *signer* address update.

Status: Fixed
(revised commit: 27f472797c697fceefccfb8bd2f0e2bf2cd32b80)

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.