HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: NF3Labs
Date: 17 Nov, 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 NF3Labs
Approved By	Paul Fomichov | SC Audits Expert at Hacken OÜ Kaan Caglan | Senior Solidity SC Auditor at Hacken OÜ
Tags	Exchange
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://nf3.exchange/
Changelog	02.11.2023 - Initial Review 17.11.2023 - 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 NF3Labs (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

The NF3x Broker's Protocol is designed to facilitate secure and trustless over-the-counter (OTC) trading of Fungible Token (FT) and Non-Fungible Tokens (NFTs) through intermediaries (“third-parties/brokers”)

Its primary purpose is to facilitate swaps & trades for users that are looking to buy specific assets, swap between assets or sell an asset. This protocol simplifies the process of finding suitable counterparties and completing OTC swaps through a third-parties network.

- **Brokers.sol:** Central contract governing the OTC Broking Protocol. It facilitates the trust trading mechanism by utilizing brokers as intermediaries between users. Within this contract, the key functionalities include token swaps, ensuring the validity of trades through sanity checks, transferring fees, and managing trade nonces.
- **IBrokers.sol:** An interface contract for **Brokers.sol**.
- **BrokersSignatureUtils.sol:** This contract manages the creation and verification of EIP-712 compliant signatures for broker swaps in a trading system. It focuses on hashing and validating trade data, assets, fees, and related trade information.
- **BrokersStorage.sol:** An abstract storage contract that serves the main Brokers contract, primarily managing nonces for users. Utilizing the OpenZeppelin's BitMaps structure, it allows for efficient setting and retrieval of nonces for specific user addresses.
- **BrokersTokenTransferrer.sol:** An abstract contract derived from the BrokersStorage contract, specialized in managing the transfer of various asset types, especially in the context of the Brokers protocol.
- **DataTypes.sol:** This file defines the main data structures and enumerations utilized in the Brokers protocol, aiding in the organization and handling of assets, fees, and trade-related data.

Privileged roles

- Owner: The owner can set minimum fees for platform and the platform address where fees will be transferred.

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 provided:
  - Project overview is detailed.
  - Deployment instructions are provided.
  - All roles in the system are described.
  - Use cases are described and detailed.
  - For each contract all futures are described.
  - All interactions are described.
- The technical requirements are provided:
  - Technical specification is provided.
  - The description of the development environment is provided.

Code quality

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

- The code is structured and readable.
- The development environment is configured.
- Best practices are followed.

Test coverage

Code coverage of the project is **97.1%** (branch coverage).

- Not all branches are covered with tests.

Security score

As a result of the audit, the code does not contain any severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

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

Summary

According to the assessment, the Customer's smart contract has the following score: **9.9**

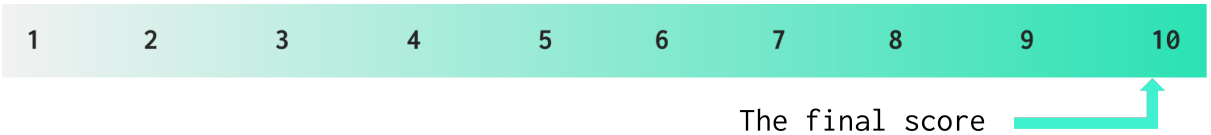

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
02 Nov 2023	1	0	0	2
17 Nov 2023	0	0	0	0

Risks

- No potential security risks were found during the audit research.

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

Findings

■■■■ Critical

C01. Missing `platformAmount` Control

Impact	High
Likelihood	High

The current implementation of the trust trading functionality permits the swapping of tokens wherein a fee is applied for both the broker and the platform. While the broker fee can be optionally set to zero based on the broker's decision, the platform fee is crucial and should always be non-zero to maintain the platform's sustainability and trust. However, the present scenario lacks any enforcement mechanism to ensure this. Due to the absence of any controls, users can potentially bypass the intended fee mechanism by setting the `platformAmount` to zero, resulting in an unfair advantage and loss of expected revenue for the platform.

Path: `./contracts/brokers/Brokers.sol: function _swapSanityChecks()`

POC CODE:

```
function test_noFees() public {
    _mintAndApprove();

    TradeInfo memory _tradeInfo = _mockTradeInfo();
    _tradeInfo.maker = user1;
    _tradeInfo.duration = block.timestamp + 86_400;
    _tradeInfo.makerFees = Fees({
        token: address(ft),
        broker: broker,
        platform: platform,
        brokerAmount: 0 ether,
        platformAmount: 0 ether // 0 ether as platformAmount
    });
    _tradeInfo.makerAssets = _toAssetsMultiple(
        AssetData({
            token: address(nft),
            assetType: AssetType.ERC_721,
            tokenId: 1,
            amount: 1
        }),
        AssetData({
            token: address(ft),
            assetType: AssetType.ERC_20,
            tokenId: 0,
            amount: 0 ether
        })
```

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

```
    );
    _tradeInfo.taker = user2;
    _tradeInfo.takerFees = Fees({
        token: address(ft),
        broker: broker,
        platform: platform,
        brokerAmount: 0 ether,
        platformAmount: 0 ether // 0 ether as platformAmount
    });
    _tradeInfo.takerAssets = _toAssetsMultiple(
        AssetData({
            token: address(nft),
            assetType: AssetType.ERC_721,
            tokenId: 2,
            amount: 1
        }),
        AssetData({
            token: address(ft),
            assetType: AssetType.ERC_20,
            tokenId: 0,
            amount: 0 ether
        })
    );

    bytes memory makerSignature = getOfferSignature(
        _tradeInfo,
        user1PvtKey
    );
    bytes memory takerSignature = getOfferSignature(
        _tradeInfo,
        user2PvtKey
    );

    assertTrue(nft.ownerOf(1) == user1);
    assertTrue(nft.ownerOf(2) == user2);
    assertTrue(ft.balanceOf(user1) == 2 ether);
    assertTrue(ft.balanceOf(user2) == 2 ether);
    assertTrue(ft.balanceOf(broker) == 0 ether);
    assertTrue(ft.balanceOf(platform) == 0 ether);

    vm.prank(broker);
    startMeasuringGas("brokers trade without paying any fees");
    brokers.swap(_tradeInfo, makerSignature, true, takerSignature, true);
    stopMeasuringGas();

    assertTrue(nft.ownerOf(1) == user2);
    assertTrue(nft.ownerOf(2) == user1);
    assertTrue(ft.balanceOf(user1) == 2 ether);
    assertTrue(ft.balanceOf(user2) == 2 ether);
    assertTrue(ft.balanceOf(broker) == 0 ether);
    assertTrue(ft.balanceOf(platform) == 0 ether);
}
```

Output:

```
└─$ forge test --match-contract BrokersFees --match-test "test_noFees"
[⠒] Compiling...
[⠢] Compiling 2 files with 0.8.21
[⠃] Solc 0.8.21 finished in 2.99s
Compiler run successful!

Running 1 test for forge-test/brokersFees.t.sol:BrokersFees
[PASS] test_noFees() (gas: 701846)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.65ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Recommendation:

1. To address this vulnerability, a control mechanism should be integrated within the `_swapSanityChecks` function. Specifically, the function should contain a conditional check that reverts the transaction if `_trade.takerFees.platformAmount` and `_trade.makerFees.platformAmount` are both below a defined `minimumPlatformFee`.

Found in: 0c2d17e

Status: Fixed (Revised commit: 0bcac36e)

C02. Missing Fee Token And Platform Control

Impact	**High**
Likelihood	**High**

The trust trading system, in its current configuration, imposes fees for token swaps, which are intended to be paid to the platform. However, a significant oversight has been identified in the current implementation: there is no control over the type of token used to pay these fees, and also, there is no control over the platform that fees will be paid. This means users can potentially pay fees using any token to any platform, even those which might be of little or no value, misleading, or not recognized by the platform. This poses not only a financial risk but also threatens the integrity of the platform's operations.

Path: `./contracts/brokers/Brokers.sol: function _swapSanityChecks()`

POC CODE:

```
function test_anyTokenAsFeeToken() public {
```

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

```
    _mintAndApproveGarbageToken();

    TradeInfo memory _tradeInfo = _mockTradeInfo();
    _tradeInfo.maker = user1;
    _tradeInfo.duration = block.timestamp + 86_400;
    _tradeInfo.makerFees = Fees({
        token: address(garbageFeeToken), // any token to pay fee.
        broker: broker,
        platform: platform, // any platform to pay fee.
        brokerAmount: 0 ether,
        platformAmount: 1 ether
    });
    _tradeInfo.makerAssets = _toAssetsMultiple(
        AssetData({
            token: address(nft),
            assetType: AssetType.ERC_721,
            tokenId: 1,
            amount: 1
        }),
        AssetData({
            token: address(garbageFeeToken), // any token to pay fee.
            assetType: AssetType.ERC_20,
            tokenId: 0,
            amount: 1 ether
        })
    );
    _tradeInfo.taker = user2;
    _tradeInfo.takerFees = Fees({
        token: address(garbageFeeToken), // any token to pay fee.
        broker: broker,
        platform: platform, // any platform to pay fee.
        brokerAmount: 0 ether,
        platformAmount: 1 ether
    });
    _tradeInfo.takerAssets = _toAssetsMultiple(
        AssetData({
            token: address(nft),
            assetType: AssetType.ERC_721,
            tokenId: 2,
            amount: 1
        }),
        AssetData({
            token: address(garbageFeeToken), // any token to pay fee.
            assetType: AssetType.ERC_20,
            tokenId: 0,
            amount: 1 ether
        })
    );

    bytes memory makerSignature = getOfferSignature(
        _tradeInfo,
        user1PvtKey
    );
    bytes memory takerSignature = getOfferSignature(
```

```
        _tradeInfo,
        user2PvtKey
    );

    assertTrue(nft.ownerOf(1) == user1);
    assertTrue(nft.ownerOf(2) == user2);
    assertTrue(garbageFeeToken.balanceOf(user1) == 2 ether);
    assertTrue(garbageFeeToken.balanceOf(user2) == 2 ether);
    assertTrue(garbageFeeToken.balanceOf(broker) == 0 ether);
    assertTrue(garbageFeeToken.balanceOf(platform) == 0 ether);

    vm.prank(broker);
    startMeasuringGas("brokers trade paying fees with random garbage token.");
    brokers.swap(_tradeInfo, makerSignature, true, takerSignature, true);
    stopMeasuringGas();

    assertTrue(nft.ownerOf(1) == user2);
    assertTrue(nft.ownerOf(2) == user1);
    assertTrue(garbageFeeToken.balanceOf(user1) == 1 ether);
    assertTrue(garbageFeeToken.balanceOf(user2) == 1 ether);
    assertTrue(garbageFeeToken.balanceOf(broker) == 0 ether);
    assertTrue(garbageFeeToken.balanceOf(platform) == 2 ether);
}
```

Output:

```
└─$ forge test --match-contract BrokersFees --match-test
"test_anyTokenAsFeeToken"
[⠒] Compiling...
[⠢] Compiling 1 files with 0.8.21
[⠒] Solc 0.8.21 finished in 3.01s
Compiler run successful!

Running 1 test for forge-test/brokersFees.t.sol:BrokersFees
[PASS] test_anyTokenAsFeeToken() (gas: 737762)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.69ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Recommendation:

To mitigate this issue, the system should integrate a whitelist mechanism for fee tokens and platform. Only tokens that are on this whitelist should be accepted as valid payment for fees. This will ensure that only recognized and valued tokens are accepted, thereby protecting the platform's financial interests and operational integrity. To achieve this, the following steps are suggested:

- **Whitelist Integration**: Incorporate a mapping or array structure in the contract that holds the addresses of whitelisted fee tokens and whitelisted platforms.
- **Validation Mechanism:** Before accepting any token as a fee, validate its address and the platform against the whitelist. If the token or platform is not on the list, the transaction should be reverted.

Found in: 0c2d17e

Status: Fixed (Revised commit: 739590c0)

High

No high severity issues were found.

Medium

No medium severity issues were found.

Low

L01. Floating Pragma

Impact	Low
Likelihood	Medium

The contract utilizes a floating pragma notation ^0.8.18. This approach can pose risks since it might lead to the contract's deployment with a compiler version different from the one it was rigorously tested with. A fixed pragma version ensures that deployments avoid potential issues stemming from older compilers with known bugs or newer versions that might not have undergone thorough testing.

Path: ./contracts/Interfaces/IBrokers.sol, ./contracts/brokers/BrokersStorage.sol, ./contracts/brokers/BrokersSignatureUtils.sol, ./contracts/brokers/BrokersTokenTransferrer.sol, ./contracts/brokers/Brokers.sol:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;
```

Recommendation: It is advised to pin the pragma to a specific version that has been extensively tested to ensure consistent compiler behavior and minimize unforeseen vulnerabilities.

Found in: 0c2d17e

Status: Fixed (Revised commit: 0495d94f)

Informational

I01. Avoid Unnecessary Initializations Of Uint256 And Bool Variable To 0/false

In Solidity, it is common practice to initialize variables with default values when declaring them. However, initializing `uint256` variables to `0` and `bool`variables to `false` when they are not subsequently used in the code can lead to unnecessary Gas consumption and code clutter. This issue points out instances where such initializations are present but serve no functional purpose.

Path: `./contracts/brokers/BrokersTokenTransferrer.sol:`

```
for (i = 0; i < len; ) {
uint256 deduction = 0;
bool deductedContra = false;
```

Recommendation: It is recommended not to initialize integer variables to 0 to and boolean variables to false to save some Gas.

Found in: 0c2d17e

Status: Fixed (Revised commit: 0495d94f)

I02. Redundant Inheritance Of `Ownable` in `Brokers` Contract

In the `Brokers` contract, there's an unnecessary inheritance from the OpenZeppelin's `Ownable` contract. Upon reviewing the contract's methods and functionalities, it's evident that there are no methods utilizing the `onlyOwner` modifier or any other features provided by the `Ownable` contract. This inheritance introduces unnecessary complexity and Gas overhead.

Path: `./contracts/brokers/Brokers.sol:`

```
contract Brokers is Ownable, BrokersTokenTransferrer, BrokersSignatureUtils {
```

Recommendation: To optimize the contract and reduce complexity, it is suggested to remove the inheritance from the `Ownable` contract since it is not being utilized. Ensure to double-check if any dependencies or functionalities might be affected by this change, but based on the current review, the removal should be straightforward.

Found in: 0c2d17e

Fix Status: Now *onlyOwner* modifier is needed for *setMinFees* function.

Status: Fixed (Revised commit: 0bcac36e)

I03. `event` Declared But Not Emitted

An event within the contract is declared but not utilized in any of the contract's functions or operations. Having unused event declarations can consume unnecessary space and may lead to misunderstandings for developers or users expecting this event as part of the contract's functionality.

Path: ./contracts/Interfaces/IBrokers.sol:

```
event BrokerRegistered(address indexed broker);
event NonceSet(address owner, uint256 nonce);
```

Recommendation: Consider removing the unused event declaration to optimize the contract and enhance clarity. If there is an intent for this event to be part of certain operations, ensure it is emitted appropriately. Otherwise, for the sake of clean and efficient code, it's advisable to remove any unused declarations.

Found in: 0c2d17e

Status: Fixed (Revised commit: 0495d94f)

I04. Unconditional Event Emission in Fee Transfer Function

The `_transferFee` function within the contract is designed to handle fee transfers. It contains two conditional checks to determine if fees should be transferred to the broker and/or platform. However, irrespective of whether the `safeTransferFrom` function is executed or not, the `FeeTransferred` event is emitted. Emitting events unnecessarily can consume additional Gas and might confuse event listeners or developers monitoring contract activities.

Path: ./contracts/brokers/BrokersTokenTransferrer.sol:

```
emit FeeTransferred(_from, fees);
```

Recommendation: To optimize Gas consumption and improve clarity, it is advisable to conditionally emit the `FeeTransferred` event based on the execution of the `safeTransferFrom` method. Incorporate a condition that checks if either `fees.platformAmount` or `fees.brokerAmount` is greater than zero before emitting the event. This will ensure the event is emitted only when a fee transfer actually occurs.

```
if (fees.platformAmount > 0 || fees.brokerAmount > 0)
    emit FeeTransferred(_from, fees);
```

Found in: 0c2d17e

Status: Reported

I05. Ownership Irrevocability Vulnerability in Smart Contract

The smart contract under inspection inherits from the *Ownable* library, which provides basic authorization control functions, simplifying the implementation of user permissions. Given this, once the owner renounces ownership using the renounceOwnership function, the contract becomes ownerless. As evidenced in the provided transaction logs, after the renounceOwnership function is called, attempts to call functions that require owner permissions fail with the error message: "Ownable: caller is not the owner."

This state renders the contract's adjustable parameters immutable and potentially makes the contract useless for any future administrative changes that might be necessary.

Path: `./contracts/brokers/Brokers.sol:`

```
contract Brokers is Ownable, BrokersTokenTransferrer, BrokersSignatureUtils {
```

Recommendation: To mitigate this vulnerability:

1. Override the renounceOwnership function to revert transactions: By overriding this function to simply revert any transaction, it will become impossible for the contract owner to unintentionally (or intentionally) render the contract ownerless and thus immutable.
2. Implement an ownership transfer function: While the Ownable library does provide a transferOwnership function, if this is not present or has been removed from the current contract, it should be re-implemented to ensure there is a way to transfer ownership in future scenarios.

Found in: 0c2d17e

Status: Fixed (Revised commit: e518a9d7)

I06. Inappropriate Use of `unchecked` Block Scope in `_transferAssets` Function

The function `_transferAssets` is designed to transfer various types of assets (ERC-721, ERC-1155, KITTIES, PUNK, and ERC-20) from one address to another. It includes a loop to handle each asset in the `_assets.assets` array. A concern arises with the scope of the `unchecked` block that encompasses the entire function body. The current implementation of the `unchecked` block is broader than necessary, which could potentially lead to unchecked arithmetic operations that may cause silent overflow/underflow issues in parts of the code where such checks are critical for contract security.

Path: `./contracts/brokers/BrokersTokenTransferrer.sol:`

```
31:         unchecked{
32:             uint256 i;
33:             uint len = _assets.assets.length;
```

Recommendation: The **unchecked** block should only be used where overflow/underflow checks are intentionally avoided after thorough analysis that their absence does not pose a security risk. In this case, it seems that the intention was to use **unchecked** for the increment of the **i** variable only. Therefore, the scope of the **unchecked** block should be reduced to only cover the **++i** statement, or alternatively, consider validating that the unchecked increment cannot lead to an overflow due to the bounds of the loop being determined by the length of a storage array, which inherently has a maximum size well below the overflow threshold of a **uint256**.

Found in: 0c2d17e

Status: Fixed (Revised commit: e518a9d7)

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/NF3Labs/OTC-Broking-Protocol
Commit	0c2d17e
Whitepaper	NF3X Broker's Protocol Litepaper
Requirements	NF3x Brokers protocol flow
Technical Requirements	NF3x Brokers protocol flow
Contracts	File: Brokers.sol SHA3: 152ca4203e052973a47f8fddd00fd8eada32b905db9f57290bcccfda File: IBrokers.sol SHA3: 6b15d297b8beea87925f78d92fd90f4aee2b63dbd7adf0d494a32c66 File: BrokersSignatureUtils.sol SHA3: d38390c1d5b0d0120fce63ae664f477e112850c9bebd655b9e8f5e1f File: BrokersStorage.sol SHA3: c7d391310f27410daf0d093d7a4a63f3c21a214129e1588a111edb64 File: BrokersTokenTransferrer.sol SHA3: c74e64ffdddd01a9ba95ac0158eca6088a7ab78a97557680027a9ea8 File: DataTypes.sol SHA3: 9294697c3526909dfc57716df1579a8386ea6c66925b3a619742e0b2

Second review scope

Repository	https://github.com/NF3Labs/OTC-Broking-Protocol
Commit	80df06ae
Whitepaper	NF3X Broker's Protocol Litepaper
Requirements	NF3x Brokers protocol flow
Technical Requirements	NF3x Brokers protocol flow
Contracts	File: Brokers.sol SHA3: 6e8e33a8fdbc298c530d8cae06a19760384ccf45677342b315de8013 File: IBrokers.sol SHA3: 261c98913a19e91a1f85b777a0552d9a223dd6bc34c6cdb31cbdb63f File: BrokersSignatureUtils.sol SHA3: 2eae112000dce007d2e7a634a16099ea0d20f68e67deecda9e1a6e82

	File: BrokersStorage.sol SHA3: cfa0b30fdc37e6aca183a4dffab955eb1a0abfe2f32918117f0217d6 File: BrokersTokenTransferrer.sol SHA3: 80ae5b15556a941dee376cdf1918143b319d49635f678f2cfe5ce777 File: DataTypes.sol SHA3: ce8d145aab4c438ac35499107f1e3687acc0ca9412d2888b61ab3a85