HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: NF3X
Date: 23 June, 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 NF3X
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	Airdrop ; EIP-712 signature
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://nf3.exchange/
Changelog	12.05.2023 - Initial Review 31.05.2023 - Second Review 23.06.2023 - 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 NF3X (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 audit scope consists of individual helper contracts for an NFT swapping platform that handles signature verification, validations for certain inputs, variable storage, Airdrops, and data types used in the platform.
This audit covers only part of the project that includes 2 functionalities.

- Airdrop. The protocol wants to allow users to benefit from airdrops even when their NFTs are locked in DEX contracts.
- Signatures. The protocol wants to implement the EIP-712 standard for signatures.

The files in the scope:

- **SigningUtil.sol:** Handles the signature verifications in the platform by using the EIP712 standard.
- **AirdropClaim.sol:** The contract to claim airdrops using NFTs that are reserved and locked in the vault. This allows users to continue receiving airdrops even if their NFTs are locked.
- **ValidationUtils.sol:** Contract to validate complex inputs in the system and ensure that the inputs are not corrupted.
- **StorageRegistry.sol:** The interface that defines all the functions related to storage for the protocol.
- **DataTypes.sol:** The contract that defines complex structs used in the system.
- **IStorageRegistry.sol:** Interface for the StorageRegistry.sol contract.
- **LoanDataTypes.sol:** The contract that defines complex structs related to Loans in the system.
- **ISigningUtils.sol:** Interface for the SigningUtil.sol contract.
- **ERC2771ContextUpgradeable.sol:** The upgradeable implementation of the EIP2721 Meta transactions standard, which is implemented by Openzeppellin.
- **IAirdropClaim.sol:** Interface for the AirdropClaim.sol contract.
- **IWhitelist.sol:** The interface that defines all the functions related to the whitelisting of tokens. The implementation of the interface is out of scope.

Privileged roles

- AirdropClaim :
    - Reserve contract :
        - Can claim an airdrop.

        - Can transfer ownership and complete reservation.
    - Owner :
        - Can withdraw assets.
- StorageRegistry :
    - only approved (swap contract, reserve contract or loan contract) :
        - Can set the nonce.
        - Can set the claim contract address.
    - Owner :
        - Can set the market address.
        - Can set the vault address.
        - Can set the reserve address.
        - Can set the whitelist address.
        - Can set the swap address.
        - Can set the loan address.
        - Can set the signing util address.
        - Can set the airdrop claim address.
        - Can set the position token address.
- Signer: The address that creates signatures to be validated using the SigningUtil.sol contract.

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 present for the functionalities covered in this audit.
- Technical description of the contracts is present.
- Development environment description is present.
- NatSpecs are satisfactory.

Code quality

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

- The development environment is configured.

Test coverage

Code coverage of the project is **72.73%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Some cases are not covered with tests.
- Interactions by several users are not tested thoroughly.

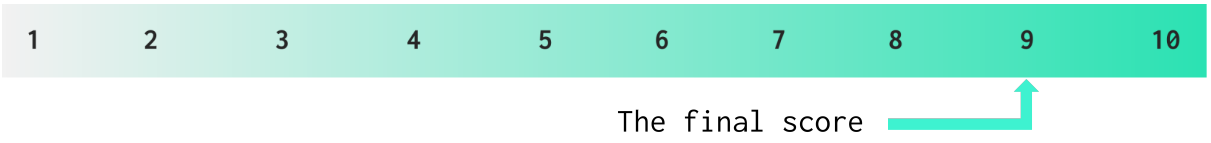
Security score

As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

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

Summary

According to the assessment, the Customer's smart contract has the following score: **9.0**. The system users should acknowledge all the risks summed up in the risks section of the report.



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

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
12 May 2023	4	4	1	2
31 May 2023	0	2	0	0
23 June 2023	0	0	0	0

Risks

- The *SigningUtils.sol* contract follows the EIP712 standard correctly; however, it **does not have the functionality to automatically update nonce values and provide protection against signature replay attacks**. The nonce variable should be updated correctly after signature verification is completed on chain. This functionality is **out of the scope of this audit**.
- The airdrop and ownership transfer is done in AirdropClaim.sol through Reserve.sol. Reserve.sol is **out of the scope of this audit**. However, it has **critical function access** inside the scope.
- The scope has an **ERC2771 contract** for **Gas forwarding;** however, there is **no documentation** on how this logic will be implemented off-chain.

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

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	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	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

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

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	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	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

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

Flashloan Attack	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	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	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	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	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	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	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

■■■■ Critical

C01. Unverifiable Logic - External Call to Untrusted Address

Impact	High
Likelihood	High

The AirdropClaim.sol contracts' *claimAirdrop()* function makes an external call to a user provided address.

There are no restrictions on the address of this external contract, and the logic behind it is out of scope of this audit.

This can lead to unexpected behavior since the security of the called contract cannot be verified.

Path: ./contracts/AirdropClaim.sol : claimAirdrop()

Recommendation: Implement a whitelist of trusted contracts for this call. Implement a standard interface for the contracts that will be called.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Mitigated (Revised commit: 9e40425) (The whitelist check is done on the Reserve.sol contract through the StorageRegistry.sol. Since only the Reserve.sol contract can call *claimAirdrop()*, the issue is fixed. However, further functionality of the Reserve.sol contract is out of this audits scope.)

C02. Invalid Initialization

Impact	High
Likelihood	High

The AirdropClaim.sol contract has variable initializations in the constructor.

This will lead to the proxy having an uninitialized state of those variables that are initialized in the constructor.

Path: ./contracts/AirdropClaim.sol : constructor

Recommendation: Initialize all variables in the *initialize()* function.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

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

High

H01. Undocumented Functionality

Impact	Medium
Likelihood	High

The functionalities that are covered in this audit are not covered by the documentation.

This can lead to misunderstanding in the intended purpose of the functionalities and increase the risk of not noticing some implementation mistakes.

Path: ./

Recommendation: The functionalities should be fully documented.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

Medium

M01. NatSpec Contradiction

Impact	Low
Likelihood	High

In the *DataTypes.sol* contract's *SwapAssets* struct, NatSpec contradicts the implementation. It is stated that the *tokens* array is a 2d array that also stored the tokenIds; however, the code indicates that the tokens array is a 1d array.

This may lead to unexpected behavior.

Path: ./utils/DataTypes.sol : SwapAssets

Recommendation: Either fix the implementation or fix the NatSpec.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

M02. Highly Permissive Role Access

Impact	High
Likelihood	Low

The AirdropClaim.sol contracts' *withdrawAssets()* function allows the owner to withdraw assets and send them to a chosen recipient.

There are no restrictions on the address of this recipient.

This can lead to fund manipulation in case the owner address is corrupted.

Path: ./contracts/AirdropClaim.sol : withdrawAssets()

Recommendation: The assets should be sent only to their legitimate owner.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: a69718e)

M03. Highly Permissive Role Access - Unverifiable Logic

Impact	High
Likelihood	Low

The contract StorageRegistry.sol allows the owner to modify many contract addresses at any time.

The contracts which interact with these addresses are out of the scope of this audit.

Therefore, it is impossible to know what could be the consequences of changes in the contract addresses.

Path: ./contracts/StorageRegistry.sol

Recommendation: Implement a multi-sig access management system with a Timelock controller (like OpenZeppelin Defender) and provide clear explanations to the users in the public documentation.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: a69718e)

M04. Data Consistency

Impact	Medium
Likelihood	Medium

The AirdropClaim.sol contract sets reserve, vault and whitelist addresses in the constructor instead of using those defined in StorageRegistry.

This can lead to the storage of two different addresses for the same value in the system.

Path: ./contracts/AirdropClaim.sol

Recommendation: Store one value in only one place or document why you would store it in two different places.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

Low

L01. Floating Pragma

Impact	Low

The project uses floating pragma `^0.8.9`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Path: all contracts in scope.

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

L02. Duplicate Code

Impact	Low

There is a duplication in the ValidationUtils.sol contracts' `verifySwapAssets()` function. The check of the ETH value can be done using the internal function `checkEthAmount()`, however, it is done by duplicating the function's code.

Duplication of code may lead to unnecessary Gas consumption and decrease code readability.

Path: ./contracts/lib/ValidationUtils.sol : verifySwapAssets()

Recommendation: Use the internal function `checkEthAmount()` to check the ETH value.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

L03. Missing Zero Address Validation

Impact	Low

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

This can lead to unwanted external calls to 0x0.

Path: ./contracts/AirdropClaim.sol : constructor(), initialize(), transferOwnershipAndCompleteReservation()

Recommendation: Implement zero address checks.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

L04. Variable Shadowing

Impact	Low

StorageRegistry.getNonce().owner shadows:

- OwnableUpgradeable.owner()

StorageRegistry.checkNonce().owner shadows:

- OwnableUpgradeable.owner()

StorageRegistry.setNonce().owner shadows:

- OwnableUpgradeable.owner()

Path: ./contracts/StorageRegistry.sol

Recommendation: Rename related variables/arguments.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

Informational

I01. State Variable Default Visibility

Some variable visibilities are not set explicitly.

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

Path: ./contracts/lib/SigningUtil.sol :

ASSETS_TYPE_HASH, SWAP_ASSETS_TYPE_HASH, RESERVE_INFO_TYPE_HASH, ROYALTY_TYPE_HASH, LISTING_TYPE_HASH, SWAP_OFFER_TYPE_HASH, RESERVE_OFFER_TYPE_HASH, COLLECTION_SWAP_OFFER_TYPE_HASH, COLLECTION_RESERVE_OFFER_TYPE_HASH, LOAN_OFFER_TYPE_HASH, COLLECTION_LOAN_OFFER_TYPE_HASH, LOAN_UPDATE_OFFER_TYPE_HASH

Recommendation: Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

Found in: dff58face7ccafe658470fd55f1e1c4e97c78e8e

Status: Fixed (Revised commit: 9e40425)

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.

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/contracts-V2
Commit	dff58face7ccafe658470fd55f1e1c4e97c78e8e
Requirements	Link
Contracts	File: contracts/lib/SigningUtil.sol SHA3: 10b8d5af4e65eee619033ec4f0ec2ddcfd5d4c2f27c36179bd1bb8c38b96af49 File: contracts/AirdropClaim.sol SHA3: 4618029a00eb7a09081cd89d2f301dfeaf8110866176c8215b8bf5e599ba4e48 File: contracts/lib/ValidationUtils.sol SHA3: 23a1458f11d2eb6093caa3793228214003c35a85d9a5e417184ce022eae2187e File: contracts/StorageRegistry.sol SHA3: 608259b7bf3d418f588505348cd84d05b759bc089e1deda83c802362ddcf954b File:utils/DataTypes.sol SHA3: 0fa54f9efc82c22c14aaace535c1d458304323045cd2c3ea9f480bf1db14ca60 File: contracts/Interfaces/IStorageRegistry.sol SHA3: 7b3e600ff4d3ebe4e6121745980acb9253c0607ea7a614d7085e11cc2184dec2 File:utils/LoanDataTypes.sol SHA3: defbaa6bda7ec4ecbac1ec122e202fa8e038bc00c5138dab55169ae1c229430b File: contracts/Interfaces/lib/ISigningUtils.sol SHA3: 5846c825f8cc4014dd114570e9f00d47a57df2bcf6a64d5c0d18a633843579df File: contracts/ERC2771ContextUpgradeable.sol SHA3: 1fcb032a1c4bcbd34199e9c7b30a0679dbdc6a52c01e65605cba181713ecd781 File: contracts/Interfaces/IAirdropClaim.sol SHA3: 7206afe8656404d4cbcbff1b34eb58e0a8e7622bcfb8b54ac828565cdf150b79 File: contracts/Interfaces/IWhitelist.sol SHA3: 90cd7b2834ba8b7137135537fa6dea1ae89c6e9e9bd666ab406d5cea93dc29ab

Second review scope

Repository	https://github.com/NF3Labs/contracts-V2
Commit	9e40425aeabf7e4996dd118f11dbd558a0de178e
Requirements	NF3x ContractsV2 Flows SHA3: 368f22e0e34530d8ccb7d4f4aa812cb57a34763607f91c30d5d9ebf0d40352573749a0e028bd6dfe2351534d254eaa8c6f0693a66378d7cf63b9a4a908710f5a
Contracts	File: contracts/lib/SigningUtil.sol SHA3: 91b68227572f7c1772ced7b54ee5bbc5c0d220c68d81da4f357527715f225eda

	File: contracts/AirdropClaim.sol SHA3: cc7656e65872aab10b0399f88749e3c53753cdcfd046b82f97c0b559f752af2f File: contracts/lib/ValidationUtils.sol SHA3: 6eab11058ee1ea286623e8a85f8b69f0969e93d317327b5518130fc48cc8b099 File: contracts/StorageRegistry.sol SHA3: 661bca8f39a6e3caf6936c332ea5bfc41c720dd38c41369bf92ab8369273c1a6 File: utils/DataTypes.sol SHA3: 543264099b7b289a3a4aeb12289c9f61a84aeb20f4a892c49b1ad34fa4c952ba File: contracts/Interfaces/IStorageRegistry.sol SHA3: 415184673e0ea9dc23c9d401d4eca17d1aa82a986e0204251426418a3fde7bc8 File: utils/LoanDataTypes.sol SHA3: b6772c92ac5a9a2886006a8ca8f8b0356c337d4d39eb07a6f0cf1e3c4c5ee0a0 File: contracts/Interfaces/lib/ISigningUtils.sol SHA3: 8aa3ef9b4448cd8df2d6effc052d01e302031c2567f28b79fea1a57d8ba8f93e File: contracts/ERC2771ContextUpgradeable.sol SHA3: 55433218f4157554c9502f98a0608d235a15080927a45c718b7446e9273b89ab File: contracts/Interfaces/IAirdropClaim.sol SHA3: 926658ec5a647a8a08f036543367020dbcacbc8f47ae6bd9ad0d3b5ee27ca73d File: contracts/Interfaces/IWhitelist.sol SHA3: 05f1b9ac433f52a4e7c07b07ae0e65f195e0b0c71f5a08f8e3e133b547a22506

Third review scope

Repository	https://github.com/NF3Labs/contracts-V2
Commit	a69718e71ba8a6e220ff8665c2799abd8cad8327
Requirements	NF3x ContractsV2 Flows SHA3: bb07f87d39584ea60349d9fd3fd3454fb620930b1a60919fe9049ebb1cf3e9b32873340efc2b23370cd1ed0a14de590d60114779a844f24361f9bfacd672db10
Technical Requirements	NF3x Technical Documentation Sha3: 818dd313f28f43e7a99c00b7e59ae938a469d4496b60da790e578b8a2c9410d2aa6597cfad8700dc9b6ef7e9b742d30a58e9e650ffb256e0ea089b68128b11f4
Contracts	File: contracts/lib/SigningUtil.sol SHA3: 8b7d22dc10b453784deb88fdb8b8d07c3420605dce2fc99d1815110661cb1fe7 File: contracts/AirdropClaim.sol SHA3: 19ad8fc8866db2eda4dd3b0337416fd2f12fac2241064ea3ebe5516bef3344a7 File: contracts/lib/ValidationUtils.sol SHA3: 8c30d6a8713d940e357ff8812c811dc2451449e619d2ca9e8ee81463328dae6e File: contracts/StorageRegistry.sol SHA3: 9456be04f357db9eb189808a724dd233b3513faed671725e3340d3f006b86cb8 File: utils/DataTypes.sol SHA3: 543264099b7b289a3a4aeb12289c9f61a84aeb20f4a892c49b1ad34fa4c952ba

	File: contracts/Interfaces/IStorageRegistry.sol SHA3: c763df9c927146863d930c518d986b3b53ce8ed146c3e55de9831cfdfb7bdd00 File: utils/LoanDataTypes.sol SHA3: b6772c92ac5a9a2886006a8ca8f8b0356c337d4d39eb07a6f0cf1e3c4c5ee0a0 File: contracts/Interfaces/lib/ISigningUtils.sol SHA3: 8aa3ef9b4448cd8df2d6effc052d01e302031c2567f28b79fea1a57d8ba8f93e File: contracts/ERC2771ContextUpgradeable.sol SHA3: 30321a659f18d199ed5f034a472b8cf770ef1bd6c0beacaa551c67c43dabe89d File: contracts/Interfaces/IAirdropClaim.sol SHA3: d5fa478336622ad2a086f886facda79e4fec2564e976600376570f5a7ff66b14 File: contracts/Interfaces/IWhitelist.sol SHA3: e91763630899fac7996a05992d33baa0c8aa45e6f3b839dacc382f6b2719b07f