HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Hedgey Finance
Date: January 31, 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 Hedgey Finance
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	Batch NFT Minter
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://hedgey.finance/
Changelog	17.01.2023 - Initial Review 31.01.2023 - 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 Hedgey Finance (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/hedgey-finance/NFT_OTC_Core/tree/batchMinter
Commit	e906c06c12a0c9ea960ac75a25cba0a2b5128f36
Functional Requirements	https://github.com/hedgey-finance/NFT_OTC_Core/blob/batchMinter/Documentation/BatchNFTMinter%20Technical%20Documentation.pdf
Technical Requirements	https://github.com/hedgey-finance/NFT_OTC_Core/blob/batchMinter/Documentation/BatchNFTMinter%20Technical%20Documentation.pdf
Contracts	File: ./contracts/libraries/TransferHelper.sol SHA3: 6afcf1fb04f38c7900ddb41c3e8095d0dd84fb481665aaeda3a6879aba4a3f84 File: /contracts/BatchNFTMinter.sol SHA3: 865ad36ce5c0805d5625ab825011bc5e62ca9a0e37cbeeaf9ec4dc30e4efcb67 File: /contracts/interfaces/INFT.sol SHA3: 3f78171b773a5e82af3583924b44eb4c8853834d06b7e1bb7c1d1ce397e709eb File: /contracts/interfaces/IWETH.sol SHA3: 702a6c595a3ea5c615f3aff3008b760d6d70f4c4e23f06f3ea4329160d019ee2

Second review scope

Repository	https://github.com/hedgey-finance/NFT_OTC_Core/tree/batchMinter
Commit	42ab72c8fb298758d8a84814eb4d572b9c7d58de
Functional Requirements	https://github.com/hedgey-finance/NFT_OTC_Core/blob/batchMinter/Documentation/BatchNFTMinter%20Technical%20Documentation.pdf
Technical Requirements	https://github.com/hedgey-finance/NFT_OTC_Core/blob/batchMinter/Documentation/BatchNFTMinter%20Technical%20Documentation.pdf
Contracts	File: ./contracts/libraries/TransferHelper.sol

	SHA3: f0ecba2d20a1afdf1e68b03793f58cc980442c59dce5add819c6e685b78a771d File: ./contracts/BatchNFTMinter.sol SHA3: d80b31c7571c76cc85b9e4058eeeb9f2dc0d926a184bc2436d75f52ee03bdc46 File: ./contracts/interfaces/INFT.sol SHA3: 3f78171b773a5e82af3583924b44eb4c8853834d06b7e1bb7c1d1ce397e709eb File: ./contracts/interfaces/IWETH.sol SHA3: 702a6c595a3ea5c615f3aff3008b760d6d70f4c4e23f06f3ea4329160d019ee2

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 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 asset 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 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 provided.
- Technical description is provided.
- NatSpec documentation is complete.

Code quality

The total Code Quality score is **9** out of **10**.

- The code violates the following official Solidity style guide.

Test coverage

Code coverage of the project is **51.41%** (branch coverage).

Security score

As a result of the audit, the code contains **3** low severity 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: **9.8**.

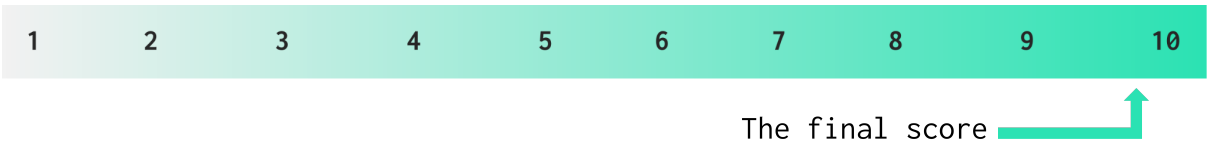

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
17 January 2023	7	2	0	0
31 January 2023	3	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and 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.	Passed
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.	Not Relevant
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.	Not Relevant
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.	Not Relevant
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant
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 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.	Not Relevant
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Failed
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 sufficient, 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

NFT_OTC_Core is a set of NFT-minting contracts:

- *BatchNFTMinter* — simple minting contract allowing to batch mint any NFT contract with any amounts to any addresses specified.
- *TransferHelper* - the library to help safely transfer tokens and handle ETH wrapping and unwrapping of WETH.

Privileged roles

- The contracts in the scope have no role-based access per se.

Risks

- No substantial risks were identified.

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

M01. Inconsistent Data - Unused Return Value

The return value from the INFT createNFT() function is not checked.

Path:
./contracts/BatchNFTMinter.sol

Recommendation: Check the function return value.

Status: Fixed

(revised commit: 42ab72c8fb298758d8a84814eb4d572b9c7d58de)

M02. Unscalable Functionality - Code Duplication

It is possible to write an internal functionality _batchMint() function, and emit BatchMinted at the start in one case and call the internal function.

Path:
./contracts/BatchNFTMinter.sol

Recommendation: Consider removing the duplicated code and moving it to an internal function.

Status: Fixed

(revised commit: 42ab72c8fb298758d8a84814eb4d572b9c7d58de)

Low

L01. Unfinished NatSpec

NatSpec is not complete - some Smart Contract members are undocumented.

Path:
./contracts/BatchNFTMinter.sol

Recommendation: Add NatSpec to undocumented members of the Smart Contract.

Status: Fixed

(revised commit: 42ab72c8fb298758d8a84814eb4d572b9c7d58de)

L02. Style Guide Violation - Single Quotes Instead of Double Quotes

Strings should be quoted with double quotes instead of single quotes.

Path:
./contracts/BatchNFTMinter.sol

Recommendation: Replace single quotes with double quotes.

Status: Reported

L03. Style Guide Violation - Order of Layout

Events should be declared before functions.

Path:
./contracts/interfaces/INFT.sol

Recommendation: Move events to the top of the interface code.

Status: Fixed

(revised commit: 42ab72c8fb298758d8a84814eb4d572b9c7d58de)

L04. Missing Zero Address Validation

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

Paths:
./contracts/BatchNFTMinter.sol
./contracts/libraries/TransferHelper.sol

Recommendation: Add zero-address check e.g. require(_addrParam != address(0x0), “address cannot be zero”).

Status: Fixed

(revised commit: 42ab72c8fb298758d8a84814eb4d572b9c7d58de)

L05. Unindexed Events

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

Path:
./contracts/BatchNFTMinter.sol

Recommendation: Add indexed parameters to the events.

Status: Reported

L06. Functions that Can Be Declared External

The two batchMint() “public” functions that are never called by the contract should be declared “external” to save Gas.

Path:
./contracts/BatchNFTMinter.sol

Recommendation: Declare the two batchMint() functions as external.

Status: Fixed

(revised commit: 42ab72c8fb298758d8a84814eb4d572b9c7d58de)

L07. Comment Contradiction

Spelling errors: Functiont -> Function (fixed), transfering -> transferring, funciton -> function (fixed).

Paths:
./contracts/BatchNFTMinter.sol; (fixed)
./contracts/libraries/TransferHelper.sol

Recommendation: Fix comments orthography.

Status: Reported

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.