HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Hedgey
Date: September 23st, 2022

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 Hedgey
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Swap
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://hedgey.finance/
Timeline	30.08.2022 - 23.09.2022
Changelog	02.09.2022 - Initial Review 23.09.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Hedgey (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/HedgeyDAOSwap
Commit:
091fb2f4cd94bdd8e9137c4e0b72dc89b490f71b
Documentation:
Functional requirements and technical description

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/HedgeyDAOSwap.sol
SHA3: 8906f0081675a05fb162ec40ddc358be5f9741f943adf66803ccc520c0cf30c3

File: ./contracts/interfaces/INFT.sol
SHA3: 3f78171b773a5e82af3583924b44eb4c8853834d06b7e1bb7c1d1ce397e709eb

File: ./contracts/interfaces/IWeth.sol
SHA3: 3e601d1e9768163d66a7ebae868159f5d5e1fd577f9b782afd01ae5e79815b25

File: ./contracts/libraries/NFTHelper.sol
SHA3: 1970c2e923fecda23801f342d936a0dc0e27307113fbca42ad25a7cb8894b606

File: ./contracts/libraries/TransferHelper.sol
SHA3: a4c52f61119573948db2184f0ab0b1a34e17a29e7c72832372393de6f506b0a1

Second review scope

Repository:
https://github.com/hedgey-finance/HedgeyDAOSwap
Commit:
39ded6c7b6908c811a80be47142b5ac4d6b717aa
Documentation:
Functional requirements and technical description

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/HedgeyDAOSwap.sol
SHA3: 8906f0081675a05fb162ec40ddc358be5f9741f943adf66803ccc520c0cf30c3

File: ./contracts/interfaces/INFT.sol
SHA3: 3f78171b773a5e82af3583924b44eb4c8853834d06b7e1bb7c1d1ce397e709eb

File: ./contracts/interfaces/IWeth.sol
SHA3: 3e601d1e9768163d66a7ebae868159f5d5e1fd577f9b782afd01ae5e79815b25

File: ./contracts/libraries/NFTHelper.sol
SHA3: 1970c2e923fecda23801f342d936a0dc0e27307113fbca42ad25a7cb8894b606

```
File: ./contracts/libraries/TransferHelper.sol
SHA3: 9a5f4a64785e8b7301f77221a3d4a6c320ca3140798077846bbf63ffceda71d8
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

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 and technical description are provided and comprehensive.

Code quality

The total Code Quality score is **10** out of **10**. The code structure follows the official language style guides; the contracts are annotated with the NatSpec comments. The development environment is provided, there are instructions on how to compile, build and deploy the code.

Test coverage

Deployment and basic user interactions are covered with tests. Negative cases coverage is present, and interaction by several users is tested.
Test coverage of the project is 100%.

Security score

As a result of the audit, the code contains **1** low severity issue. 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**.

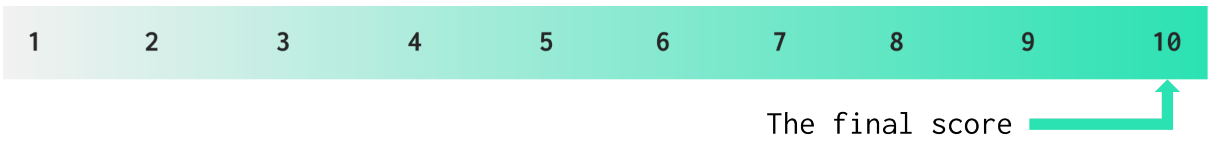

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
21 September 2022	1	0	0	0

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Failed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	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.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Custom	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	Custom	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
Gas Limit and Loops	Custom	Transaction execution costs should not depend dramatically on the amount of	Not Relevant

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Hedgey DAO SWAP is a swap/locking system with the following contracts:

- *HedgeyDAOSwap* — is a contract that allows tokens swapping and locking.

  The functionality of the contract allows users to initiate the swap by defining the following swap parameters:

  - *Token A address;*
  - *Token B address;*
  - *Amount A;*
  - *Amount B;*
  - *Unlock date;*
  - *Executor address;*
  - *NFT locker address;*

  The *token A* in *the amount A* is transferred from the swap initiator to the contract balance.

  Then the defined executor address can execute the swap:

  - If the *unlock date* is bigger than the current time, the *token B* in the *amount B* is transferred from the swap executor to the contract; *token B* in the *amount B* is locked in the NFT locker, minting an NFT to the swap initiator, and *token A* in the *amount A* is locked in the *NFT locker*, minting an NFT to the swap executor. (**NFT locker is out of audit scope**)
  - If the *unlock date* is less than the current time, the *token A* sent by the swap initiator is transferred to the swap executor, *token B* in the *amount B* is transferred from the swap executor to the swap initiator.

  The swap initiator can cancel the swap before its execution.
- *NFTHelper* — is a library that contains the helper function for locking tokens in the NFT locker contract, used in the *HedgeyDAOSwap* contract.
- *TransferHelper* — is a library that contains the collection of functions that help to transfer ERC-20 tokens, handle ETH wrapping and unwrapping of WETH, used in the *HedgeyDAOSwap* contract.
- *INFT* — is an interface for the NFT locker contract, used in the *NFTHelper* library.
- *IWeth* — is an interface for WETH, used in the *TransferHelper* library.

Risks

- The *NFT locker address* is defined by the swap initiator and its matching with the correct *NFT locker* contract can not be verified.

NFT locker contract is out of audit scope. Therefore, the secureness of token locking and NFT minting functionality can not be guaranteed.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

1. Unfinalized code

The *TransferHelper* library imports *'hardhat/console.sol'* for debugging purposes.

This may indicate that the code is not finalized.

Path: ./contracts/libraries/TransferHelper.sol

Recommendation: Remove the debugging import.

Status: Fixed (Revised commit: 39ded6c7b6908c811a80be47142b5ac4d6b717aa)

■ Low

1. Block values as a proxy for time using

The contract uses *block.timestamp* for time calculations. It is not precise and safe.

Paths: ./contracts/HedgeyDAOSwap.sol : initSwap(), executeSwap(); ./contracts/libraries/NFTHelper.sol : lockTokens()

Recommendation: It is recommended to avoid using *block.timestamp* in the time calculations. Alternatively, it is safe to use oracles.

Status: Reported

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.