BLOCKSEC

Security Audit
Report for Upheaval Contracts

Date: September 17, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Upheaval
Target	Upheaval Contracts

Version History

Version	Date	Description
1.0	September 17, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-09-17 07:24:10+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the deployed contracts [1] [2] [3] [4] of Upheaval Contracts of Upheaval.

The Upheaval Contracts of Upheaval is a decentralized exchange (DEX) protocol forked from PancakeSwap V3, deployed on the HyperEVM chain. This protocol enables users to perform operations such as swapping and providing liquidity. In addition, the protocol leverages the HyperEVM chain's high transaction-per-second (TPS) capability to deliver efficient performance for user operations.

Note this audit only focuses on the smart contracts in the following files:

- NonfungiblePositionManager.sol
- SwapRouter.sol
- UpheavalV3Factory.sol
- UpheavalV3PoolDeployer.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The deployed addresses of the contracts during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Contract	Version	Address
NonfungiblePositionManager.sol	`Version 1`	`0xC8352A2EbA29F4d9BD4221c07D3461BaCc779088`
SwapRouter.sol	`Version 1`	`0xf49A33dF1Bb5DA03B85396a768666523c1b7Eb9e`
UpheavalV3Factory.sol	`Version 1`	`0x2566163ea012C9E67c1C7080e0a073f20B548030`
UpheavalV3PoolDeployer.sol	`Version 1`	`0xDcF0ba9A0AE5B7574849152546f961f5ef0EEeD6`

[1] https://hyperevmscan.io/address/0x2566163ea012C9E67c1C7080e0a073f20B548030

[2] https://hyperevmscan.io/address/0xDcF0ba9A0AE5B7574849152546f961f5ef0EEeD6

[3] https://hyperevmscan.io/address/0xf49A33dF1Bb5DA03B85396a768666523c1b7Eb9e

[4] https://hyperevmscan.io/address/0xC8352A2EbA29F4d9BD4221c07D3461BaCc779088

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness

- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [5] and Common Weakness Enumeration [6]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.

[5] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[6] https://cwe.mitre.org/

- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we have **two** recommendation and **three** notes.

- Recommendation: 2
- Note: 3

ID	Severity	Description	Category	Status
1	-	Add non-zero address and duplicate checks	Recommendation	Confirmed
2	-	Revise the inputs and dependencies	Recommendation	Confirmed
3	-	Potential Centralization Risks	Note	-
4	-	Token integration issues	Note	-
5	-	Hardcoded values and the contract deployment chains	Note	-

The details are provided in the following sections.

2.1 Recommendation

2.1.1 Add non-zero address and duplicate checks

Status Confirmed

Introduced by `Version 1`

Description In the contract `UpheavalV3Factory`, the functions `constructor()` and `setOwner()` fail to implement proper non-zero address checks for the input `_poolDeployer` and `_owner`. Furthermore, the function `setOwner()` should include a duplicate check for the input `_owner` to prevent redundant owner assignments. In addition, the function `constructor()` in the contract `NonfungiblePositionManager` lacks non-zero address checks for the inputs `_deployer`, `_factory`, `_WETH9` and `_tokenDescriptor_`. It is recommended to implement these checks to prevent unintentional behaviors.

```
83    function setOwner(address _owner) external override onlyOwner {
84        emit OwnerChanged(owner, _owner);
85        owner = _owner;
86    }
```

Listing 2.1: UpheavalV3Factory.sol

```
35
36    constructor(address _poolDeployer) {
37        poolDeployer = _poolDeployer;
38        owner = msg.sender;
```

Listing 2.2: UpheavalV3Factory.sol

```
71    constructor(
72        address _deployer,
73        address _factory,
```

```
74        address _WETH9,
75        address _tokenDescriptor_
76    ) ERC721Permit('Pancake V3 Positions NFT-V1', 'PCS-V3-POS', '1') PeripheryImmutableState(
          _deployer, _factory, _WETH9) {
77        _tokenDescriptor = _tokenDescriptor_;
78    }
```

Listing 2.3: NonfungiblePositionManager.sol

Suggestion Add non-zero address and duplicate checks accordingly.

2.1.2 Revise the inputs and dependencies

Status Confirmed

Introduced by `Version 1`

Description The inputs and dependencies (i.e., `Pancake V3 Positions NFT-V1`, `PCS-V3-POS`, and `@pancakeswap`) used by the contract `NonfungiblePositionManager` are related to the `Pancakeswap` protocol. It is recommended to revise the corresponding inputs and dependencies.

```
71    constructor(
72        address _deployer,
73        address _factory,
74        address _WETH9,
75        address _tokenDescriptor_
76    ) ERC721Permit('Pancake V3 Positions NFT-V1', 'PCS-V3-POS', '1') PeripheryImmutableState(
          _deployer, _factory, _WETH9) {
77        _tokenDescriptor = _tokenDescriptor_;
78    }
```

Listing 2.4: NonfungiblePositionManager.sol

```
5import '@pancakeswap/v3-core/contracts/interfaces/IPancakeV3Pool.sol';
6import '@pancakeswap/v3-core/contracts/libraries/FixedPoint128.sol';
7import '@pancakeswap/v3-core/contracts/libraries/FullMath.sol';
```

Listing 2.5: NonfungiblePositionManager.sol

Suggestion Revise the inputs and dependencies accordingly.

2.2 Note

2.2.1 Potential Centralization Risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., the role `owner` in the contract `UpheavalV3Factory`) can conduct sensitive operations, which introduces potential centralization risks. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.2.2 Token integration issues

Introduced by `Version 1`

Description The project exhibits integration challenges with certain token types (e.g., fee-on-transfer, rebasing, and reflection tokens). Users should be explicitly informed about these limitations and advised to carefully consider token selection when creating pools to ensure proper functionality.

2.2.3 Hardcoded values and the contract deployment chains

Introduced by `Version 1`

Description In this project, some hardcoded values are used in some contracts (e.g., the contract `SwapRouter`). The project must ensure all hardcoded values are correct and all contracts are deployed on the proper chains for use.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS