P2PSwap Security Review

Duration: December 22, 2024 - December 29, 2024

Date: December 29, 2024

Conducted by: **KeySecurity**
gkrastenov, Lead Security Researcher

Table of Contents

1 About KeySecurity

KeySecurity is a new, innovative Web3 security company that hires top-talented security researchers for your project. We have conducted over 30 security reviews for various projects, collectively holding over $300,000,000 in TVL. For security audit inquiries, you can reach out to us on Twitter/X or Telegram @gkrastenov or check our previous work here.

2 About P2PSwap

P2PSwap is a simple Token swapper project with no fee takers. You can use this project for ERC721,ERC1155,ERC20, xERC20, ERC777 swaps where one party can set up a deal and the other accept.

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

4 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	P2PSwap
Repository	https://github.com/thedarkjester/P2PSwap
Commit hash	e64dec4fb1f81e53146ecc6b4f1940a8286cd2a8
Review Commit hash	da64bdff418113e8330d816415d6a016600e032f
Documentation	https://github.com/thedarkjester/P2PSwap
Methods	Manual review

Scope

ISwapTokens.sol
NonCancunTokenSwapper.sol
NonCancunTokenSwapperUtils.sol
TokenSwapper.sol
TokenSwapperUtils.sol

Timeline

December 22, 2024	Audit kick-off
December 27, 2024	Preliminary report
December 29, 2024	Mitigation review

Issues Found

Severity	**Count**
High	1
Medium	1
Low	1
Information	2
Code Improvement	1
Total	**6**

6 Findings

6.1 High

6.1.1 The real acceptor can not receive the sent ETH

Severity: *High*

Context: TokenSwapper.sol#L152

Description: In cases where `swap.acceptor` is a zero address, it means that anyone can complete the swap. So, if the `initiator` wants to swap 1 ETH for 100k TokenA, the `initiatorETHPortion` will be 1 ETH, and the `swap.acceptor` balance will increase by 1 ETH. However, in this scenario, since the acceptor is a zero address, the real acceptor will transfer their 100k TokenA and receive 0 ETH in return.

```
if (_swap.initiatorETHPortion > 0) {
  unchecked {
    /// @dev This should never overflow - portion is either zero or a number way
        less that max uint256.
    balances[_swap.acceptor] += _swap.initiatorETHPortion;
    //@audit-info wrong address
  }
}

emit SwapComplete(_swapId, _swap.initiator, _swap.acceptor, _swap);

address realAcceptor = _swap.acceptor == ZERO_ADDRESS ? msg.sender : _swap.acceptor;
```

Recommendation: Use the real acceptor instead of `_swap.acceptor` when the balance is incremented by `_swap.initiatorETHPortion`.

Resolution and Client comment: Resolved. Fixed at #30 PR.

6.2 Medium

6.2.1 Swapper contracts do not support USDT

Severity: *Medium*

Context: TokenSwapper.sol#L508

Description: The current version of the contract is expected to support any ERC20 standard tokens. Unfortunately, one of the most popular stablecoins, USDT, is not fully compatible because it does not return true when the transferFrom function is called. As a result, your check for a successful transfer will fail, and the open swap will not be completed.

Recommendation: Use SafeERC20 library from OpenZeppelin, which handles these inconsistencies and ensures compatibility with tokens like USDT.

Resolution and Client comment: Resolved. Fixed at #34 PR.

6.3 Low

6.3.1 NFTs with tokenId = 0 can not be swapped

Severity: *Low*

Context: NonCancunTokenSwapper.sol#L354

Description: ERC721 and ERC1155 tokens with `tokenId == 0` cannot be transferred, which blocks the opening of swaps for these tokens. In most NFT collections, the IDs of their NFTs start at 0, which means this token is minted and being used by someone. A check for `tokenId > 0` will limit these people, preventing them from opening swaps for their tokens.

Recommendation: Do not check if the `tokenId` is strictly greater than 0.

Resolution and Client comment: Resolved. Fixed at #31 PR.

6.4 Information

6.4.1 Wrong event argument

Severity: *Information*

Context: TokenSwapper.sol#L163

Description: The wrong event argument will be used when `_swap.acceptor` is `ZERO_ADDRESS` because, in this case, anyone can be the acceptor, so the event will not use the real acceptor.

Recommendation: Use the real acceptor address.

Resolution and Client comment: Resolved. Fixed at #30 PR.

6.4.2 Typo in event name

Severity: *Information*

Context: TokenSwapper.sol#L224

Description: The event `BalanceWithDrawn` has a typo.

Recommendation: Fix the typo.

Resolution and Client comment: Resolved. Fixed at #32 PR.

6.5 Code Improvement

6.5.1 Remove the duplicate code

Severity: *Code Improvement*

Context: Global

Description: As an additional improvement, I would recommend using an abstract contract to centralize the main logic. Currently, both `NonCancun` and `TokenSwapper` have duplicate code with only

minor differences. Similarly, the libraries could be optimized, as one is merely an extension of the other, with duplicated hashing functionality.

Recommendation: Use an abstract contract as a base contract so that child contracts can inherit and utilize the main functions.

Resolution and Client comment: Resolved. Fixed at #35 PR.