Portals Security Review

Pashov Audit Group

Conducted by: pashov

July 18th, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **Portals** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Portals

Portals is a platform that aggregates multiple calls allowing for best-route swap, stake, LP etc mechanisms. It optimizes for highest USD output + least gas consumption. A user requests a quote from the Portals API by choosing input & output tokens, input amount and sender/recipient. A Smart Order Routing algorithm is used to find the best path for the swap or action. Example use case is adding liquidity to a Curve pool (swapping to the underlying token and calling `add_liquidity`) or executing a folding strategy (borrow, swap, borrow etc). The protocol has a slippage tolerance mechanism implemented.

More docs

Observations

`PortalsRouter` is the contract handling all approvals/allowances. The contracts make heavy usage of signed approvals & ERC20's permits.

The protocol does revenue sharing, where 50% of the fees go to the `partner` or the front-end from which the on-chain orders come from. The `partner` is only logged in an event and then the revenue sharing is executed off-chain, which is not trustless.

The protocol is non-custodial, meaning it doesn't hold funds in between any transaction calls. Only time it holds funds/fees is intra-transaction. This drastically narrows down the attack surface.

The `PortalsMulticall` contract allows for arbitrary code execution by anyone.

Privileged Roles & Actors

- Router owner - can pause/unpause the `PortalsRouter` contract, set the `Multicall` contract address and also receives recovered stuck tokens in the router
- Multicall fees receiver - set by the back end off-chain (added in the `calls` array in `aggregate`), receives some fee
- Portal broadcaster - executes `SignedOrder`s by calling `PortalsRouter`
- Portal user - executes `Order`s by calling `PortalsRouter`, gives `PortalsRouter` spending allowance

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

***review commit hash* - d8be537304c8cd0bd3433b3aca5770d9375fc2ce**

***fixes review commit hash* - c1c223697b1a796623162148e1fd821e109c4107**

Scope

The following smart contracts were in scope of the audit:

- `portals/multicall/interface/**`
- `portals/multicall/PortalsMulticall`
- `portals/router/interface/**`
- `portals/router/RouterBase`
- `portals/router/PortalsRouter`

7. Executive Summary

Over the course of the security review, pashov engaged with Portals to review Portals. In this period of time a total of **3** issues were uncovered.

Protocol Summary

Protocol Name	Portals
Date	July 18th, 2023

Findings Count

Severity	Amount
Low	3
Total Findings	**3**

Summary of Findings

ID	Title	Severity	Status
[L-01]	Fees recipient can arbitrage slippage tolerance	Low	Resolved
[L-02]	The chainId is cached but might change	Low	Resolved
[L-03]	The protocol is using a vulnerable library	Low	Resolved

8. Findings

8.1. Low Findings

[L-01] Fees recipient can arbitrage slippage tolerance

The `calls` array in `PortalsMulticall::aggregate` are always expected to include a fee payment transaction, where fees from the swaps would be transferred to a protocol-controlled account. Also it is sometimes expected that a "sweep ETH" call is included in the end of the array, so that all leftover ETH is swept back to the original caller. The problem is that in this specific scenario, when the fees are sent before the sweep, the fees recipient can reenter the contract by calling `transferEth` and transfer just enough ETH that the caller balance would still be at the slippage tolerance level.

This attack requires multiple conditions (Low likelihood) and is also limited to up to slippage tolerance values (Low to Medium impact), hence the Low severity, but it is still a possible attack vector. One possible solution is to add the `nonReentrant` modifier to the `transferEth` function, but this will disallow calling it as part of the `aggregate` calls.

Discussion

pashov: Acknowledged.

[L-02] The `chainId` is cached but might change

Caching the `chainId` value (`RouterBase`'s constructor) is not a good practice as hard forks might change the chainId for a network. The better solution is to always check if the current `block.chainid` is the same as the cached one and if not, to update it. Follow the approach in OpenZeppelin's EIP712 implementation or just inherit from the OpenZeppelin's EIP712 contract.

Discussion

pashov: Resolved.

[L-03] The protocol is using a vulnerable library

The 4.8.0 version of the OpenZeppelin library has security vulnerabilities that are listed here. While currently the vulnerable code is not used in the codebase, the library should be updated to contain the latest security patches.

Discussion

pashov: Resolved.