# Arcadia Security Review

## Pashov Audit Group

Conducted by: 0xunforgiven, Shaka, 0xbepresent

October 24th - November 1st

# Contents

## 1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our 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 we try to find as many vulnerabilities as possible. We 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 **arcadia-finance/accounts-v2** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

## 4. About Arcadia

Arcadia is an on-chain system enabling individuals, DAOs, and protocols to manage assets as collateral through margin accounts. It simplifies asset management and liquidity rebalancing, protecting users from slippage and backing liabilities across financial protocols.

# 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 hashes:*

- 08bc2a06dcc73ec13d326fe812179d865e3c67c5
- 56a5cbaaeb1fe4999c4ae52fd2cc1460af7ff544

*fixes review commit hash:*

- 58f43ffe29ff435fcc25417f3199e52beadc0907

## Scope

The following smart contracts were in scope of the audit:

- `Rebalancer`
- `StakedSlipstreamLogic`
- `RebalanceLogic`
- `UniswapV3Logic`
- `SwapLogic`
- `FeeLogic`
- `BurnLogic`
- `SlipstreamLogic`
- `MintLogic`
- `RebalanceOptimizationMath`
- `AccountSpot`
- `DefaultUniswapV4AM`
- `UniswapV4HooksRegistry`

# 7. Executive Summary

Over the course of the security review, 0xunforgiven, Shaka, 0xbepresent engaged with Arcadia to review Arcadia. In this period of time a total of **3** issues were uncovered.

## Protocol Summary

| | |
|---|---|
| **Protocol Name** | Arcadia |
| **Repository** | https://github.com/arcadia-finance/accounts-v2 |
| **Date** | October 24th - November 1st |
| **Protocol Type** | Asset management |

## Findings Count

| **Severity** | **Amount** |
|---|---|
| Medium | 1 |
| Low | 2 |
| **Total Findings** | **3** |

## Summary of Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| [M-01] | ETH withdrawn is sent to msg.sender instead of the to address | Medium | Resolved |
| [L-01] | Incorrect return value in getAssetModule | Low | Resolved |
| [L-02] | getRiskFactors() will revert always | Low | Resolved |

# 8. Findings

## 8.1. Medium Findings

### [M-01] ETH withdrawn is sent to `msg.sender` instead of the `to` address

#### Severity

**Impact:** Medium

**Likelihood:** Medium

#### Description

In the `AccountSpot` contract, the `_withdraw` function does not use the `to` address as the recipient of the withdrawn funds in the case of an ETH withdrawal but instead uses `msg.sender` as the recipient. This removes the ability of the withdrawer to specify a different address to receive the funds and, what is more, can lead to the loss of funds if the caller is expecting the funds to be sent to the `to` address, as the NatSpec comment explicitly states that the `to` address is the recipient of the withdrawn funds.

```
@>    * @param to The address to withdraw to.
      */
    function _withdraw(
        address[] memory assetAddresses,
        uint256[] memory assetIds,
        uint256[] memory assetAmounts,
        uint256[] memory assetTypes,
@>      address to
    ) internal {
        for (uint256 i; i < assetAddresses.length; ++i) {
            // Skip if amount is 0 to prevent transferring addresses that have 0
            // balance.
            if (assetAmounts[i] == 0) continue;

            if (assetAddresses[i] == address(0)) {
@>              (bool success, bytes memory result) = payable
  (msg.sender).call{ value: assetAmounts[i] }("");
                require(success, string(result));
```

## Recommendations

Use the `to` address as the recipient of the withdrawn funds in the case of an ETH withdrawal. If the expected behavior is that the funds can only be withdrawn to the caller's address, add a condition to check that `to == msg.sender` and revert if that is not the case.

## 8.2. Low Findings

### [L-01] Incorrect return value in `getAssetModule`

The `public` `getAssetModule` function in the `UniswapV4HooksRegistry` contract does not check for the existence of the provided `assetId`. This oversight may result in an empty `poolKey` and cause the function to erroneously return the default asset module (`DEFAULT_UNISWAP_V4_AM`) even when the `assetId` does not exist.

```
function getAssetModule(uint256 assetId) public view returns
      (address assetModule) {
>>>     (PoolKey memory poolKey,) = POSITION_MANAGER.getPoolAndPositionInfo
  (assetId);

        // Check if we can use the default Uniswap V4 AM.
>>>     if (
            Hooks.hasPermission(uint160(address
              (poolKey.hooks)), Hooks.BEFORE_REMOVE_LIQUIDITY_FLAG)
                || Hooks.hasPermission(uint160(address
                  (poolKey.hooks)), Hooks.AFTER_REMOVE_LIQUIDITY_FLAG)
        ) {
            // If not a specific Uniswap V4 AM must have been set.
            // Returns the zero address if no Asset Module is set.
            assetModule = hooksToAssetModule[address(poolKey.hooks)];
        } else {
            // If BEFORE_REMOVE_LIQUIDITY_FLAG and AFTER_REMOVE_LIQUIDITY_FLAG
            // are not set,
            //Then we use the default Uniswap V4 AM.
            // The NoOP hook "AFTER_REMOVE_LIQUIDITY_RETURNS_DELTA_FLAG" is by
            // default not allowed,
            // as it can only be accessed if "AFTER_REMOVE_LIQUIDITY_FLAG" is
            // implemented.
            assetModule = DEFAULT_UNISWAP_V4_AM;
        }
    }
```

The absence of a check can lead to misleading information being returned from public functions. Test:

```
function testFuzz_Success_getAssetModule_InvalidTokenId(uint96 tokenId)
        public
    {
        // Calling getAssetModule()
        address assetModule = v4HooksRegistry.getAssetModule(tokenId);

        // It returns DefaultUniswapV4AM. It should returns `address
        //(0)` or revert() since tokenId does not exist.
        assertEq(assetModule, address(uniswapV4AM));
    }
```

Implement a check to verify the existence of the `poolKey`. If the `poolKey` is empty, the call function `getAssetModule` should revert or return a zero address to clearly indicate the non-existence.

## [L-02] `getRiskFactors()` will revert always

Function `getRiskFactors()` in DefaultUniswapV4AM calls `IRegistry(REGISTRY).getRiskFactors(address, address[], uint256[])` to get tokens risk factors. The value of REGISTRY for DefaultUniswapV4AM is UniswapV4HooksRegistry which acts as an intermediate contract between the asset manager and the main registry contract. The issue is that UniswapV4HooksRegistry doesn't have that function and it only implements `getRiskFactors(address creditor, address asset, uint256 assetId)` As result function `getRiskFactors()` will revert for contract DefaultUniswapV4AM and UniswapV4HooksRegistry. The impact is that risk factors for Uniswap V4 assets can't be calculated and used in another smart contract that relies on this information.

Add function `getRiskFactors(address, address[], uint256[])` to UniswapV4HooksRegistry.