Intents Smart Contract Audit

Derive, 09 April 2025

PUBLIC
hello@iosiro.com

Contents

1. Introduction 1
2. Disclaimer 2
3. Methodology 3
4. Audit findings 4
   IO-DRV-INT-001 Rugpull using malicious DeriveAsset 5
   IO-DRV-INT-002 Fail open on maxFee 7
5. Code quality improvement suggestions 8
6. Specification 10
   WithdrawBridgeIntent 10
   SubaccountDepositIntent 10
   StakeDRVIntent 10

1. Introduction

iosiro was commissioned by Derive to perform a smart contract audit of their `WithdrawBridgeIntent`, `SubaccountDepositIntent` and `StakeDRVIntent` contracts. One auditor conducted the audit on 10 and 11 March 2025, using 2 audit days.

Overview

During the audit, one medium risk and one low risk issue were found. The medium risk issue could allow a compromised executor to drain users' wallets, and the low risk issue was related to the insecure default behavior of the maximum fee mechanism.

In addition, several recommendations for code quality improvements were made.

	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Resolved	0	0	1	1	7
Closed	0	0	0	0	0

Scope

The assessment focused on the source files listed below, with all other files considered out of scope. Any out-of-scope code interacting with the assessed code was presumed to operate correctly without introducing functional or security vulnerabilities.

- **Project name:** derivexyz/v2-aa
- **Initial audit commit:** 3093bd5
- **Final review commit:** 1cd53d6
- **Files:**

  `src/intents/IntentExecutorBase.sol`,
  `src/intents/StakeDRVIntent.sol`,
  `src/intents/SubaccountDepositIntent.sol`,
  `src/intents/WithdrawBridgeIntent.sol`

A specification is available in the Specification section of this report.

2. Disclaimer

This report aims to provide an overview of the assessed smart contracts' risk exposure and a guide to improving their security posture by addressing identified issues. The audit, limited to specific source code at the time of review, sought to:

- Identify potential security flaws.
- Verify that the smart contracts' functionality aligns with their documentation.

Off-chain components, such as backend web application code, keeper functionality, and deployment scripts were explicitly not in-scope of this audit.

Given the unregulated nature and ease of cryptocurrency transfers, operations involving these assets face a high risk from cyber attacks. Maintaining the highest security level is crucial, necessitating a proactive and adaptive approach that accounts for the experimental and rapidly evolving nature of blockchain technology. To encourage secure code development, developers should:

- Integrate security throughout the development lifecycle.
- Employ defensive programming to mitigate the risks posed by unexpected events.
- Adhere to current best practices wherever possible.

3. Methodology

The audit was conducted using the techniques described below.

Code review	The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the specification and implementation, design improvements, and high-risk areas of the system.
Dynamic analysis	The contracts were compiled, deployed, and tested in a test environment, both manually and through the test suite provided. Dynamic analysis was used to identify additional edge cases, confirm that the code was functional, and to validate the reported issues.
Automated analysis	Automated tooling was used to detect the presence of various types of security vulnerabilities. Static analysis results were reviewed manually and any false positives were removed. Any true positive results are included in this report.

4. Audit findings

The table below provides an overview of the audit's findings. Detailed write-ups are provided below.

ID	Issue	Risk	Status
`IO-DRV-INT-001`	Rugpull using malicious `DeriveAsset`	Medium	Resolved
`IO-DRV-INT-002`	Fail open on `maxFee`	Low	Resolved
`IO-DRV-INT-003`	Inconsistent use of `SafeERC20`	Informational	Resolved
`IO-DRV-INT-004`	Use of `immutable` in constructor	Informational	Resolved
`IO-DRV-INT-005`	Interface mismatch	Informational	Resolved
`IO-DRV-INT-006`	Gas optimization	Informational	Resolved
`IO-DRV-INT-007`	Mark contract as `abstract`	Informational	Resolved
`IO-DRV-INT-008`	Ensure maximal compatibility	Informational	Resolved
`IO-DRV-INT-009`	Improve storage layout	Informational	Resolved

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

Critical risk	The issue could result in the theft of funds from the contract or its users.
High risk	The issue could result in the loss of funds for the contract owner or its users.
Medium risk	The issue resulted in the code being dysfunctional or the specification being implemented incorrectly.
Low risk	A design or best practice issue that could affect the ordinary functioning of the contract.
Informational	An improvement related to best practice or a suboptimal design pattern.

In addition to a risk rating, each issue is assigned a status:

Open	The issue remained present in the code as of the final commit reviewed and may still pose a risk.
Resolved	The issue was identified during the audit and has since been satisfactorily addressed, removing the risk it posed.
Closed	The issue was identified during the audit and acknowledged by the developers as an acceptable risk without actioning any change.

IO-DRV-INT-001

Rugpull using malicious `DeriveAsset`

Medium	Resolved	SubaccountDepositIntent.sol#L43

In `SubaccountDepositIntent`, the `deriveAsset` parameter is fully under the executor's control. If the executor deploys a malicious contract that implements the `IERC20BasedAsset` interface, the allowance set by users for the wrapped asset could be abused to steal funds.

The following test was developed to illustrate the exploitation of the issue:

```
contract RugPuller {
    address public wrappedAsset;

    function setWrappedAsset(address _wrappedAsset) public {
        wrappedAsset = _wrappedAsset;
    }

    function deposit(uint256, uint256 amount) external {
        IERC20(wrappedAsset).transferFrom(msg.sender, address(this),
amount);
    }
}
```

```
function test_RugPull() public onlyDeriveMainnet {
    RugPuller rugPuller = new RugPuller();
    rugPuller.setWrappedAsset(DAI);

    uint256 erc20BalanceBefore = IERC20(DAI).balanceOf(user);
    uint256 subaccountBalanceBefore =
subaccounts.getBalance(subaccountId, DAIAsset, 0);

    vm.startPrank(executor);
    depositIntent.executeDepositIntent(user, subaccountId,
address(rugPuller), 10 ether);
    vm.stopPrank();

    uint256 erc20BalanceAfter = IERC20(DAI).balanceOf(user);
    uint256 subaccountBalanceAfter =
subaccounts.getBalance(subaccountId, DAIAsset, 0);

    assertEq(erc20BalanceAfter, erc20BalanceBefore - 10 ether);
    assertEq(subaccountBalanceAfter, subaccountBalanceBefore);
    assertEq(IERC20(DAI).balanceOf(address(rugPuller)), 10 ether);
}
```

Recommendation

The `executeDepositIntent` should enforce some validation of the `deriveAsset` address to ensure it is a legitimate address.

Client response

Fixed in commit 9964159. The contract was updated to query the `manager` of the `SubAccount` and determine whether it is a permitted asset type.

IO-DRV-INT-002

Fail open on `maxFee`

Low	Resolved	WithdrawBridgeIntent.sol#L108

The smart contract will accept any fee when `maxFee` is specified as zero. This fail-open behaviour is against security best practices.

Recommendation

The fee check should be updated so that `type(uint256).max` indicates that any fee is acceptable, e.g:

```
if (maxFee != type(uint256).max)
```

Client response

Fixed in commit 0b4ba73.

5. Code quality improvement suggestions

Code improvement suggestions without security implications are listed below.

#	Location	Details
IO-DRV-INT-003	StakeDRVIntent.sol#L33	The `transferFrom` function is used instead of `safeTransferFrom`, as is the case throughout the rest of the codebase and within the StakedDeriveToken contract. Since the `DeriveToken` is known to be `ERC20` compliant, the `SafeERC20` library is not required, but for consistency, the `safeTransferFrom` function could be used instead.
IO-DRV-INT-004	StakeDRVIntent.sol#L23	Since Solidity 0.8.8 it is possible to read an `immutable` variable that has already been set within a constructor. Since the Solidity version is pinned to 0.8.9 and above, this functionality is supported and poses no risk. However, it is considered best practice to reuse constructor arguments instead of `immutable` variables to maintain backwards compatibility.
IO-DRV-INT-005	IStakedDRV.sol#L7	The second argument of `convertTo` in the `IStakedDRV` interface is incorrectly named. The parameter `token` should be renamed as `to`.
IO-DRV-INT-006	IntentExecutorBase.sol#L47	In `rescueToken` it would be more gas efficient to use `msg.sender` opposed `owner()` when calling `safeTransfer`. Since the function uses the `onlyOwner` modifier it is guaranteed that `msg.sender == owner()`.
IO-DRV-INT-007	IntentExecutorBase.sol#L13	The `IntentExecutorBase` contract should be marked as `abstract`.
IO-DRV-INT-008	IntentExecutorBase.sol#L47	The `rescueToken` function could make use of `IERC20::transfer` instead of `safeTransfer` to ensure maximal compatibility. Since this is a recovery function, the widest list of possible tokens should be supported. For example, USDT on the Tron network is incompatible with `safeTransfer` as it always returns `false`.
IO-DRV-INT-009	WithdrawBridgeIntent.sol#L29	The storage layout of `WithdrawBridgeIntent` is not optimal and can be improved. Currently, `withdrawCount` consumes an additional storage slot. While `bucketWidth`, `lastBucketStart` and `maxWithdrawPerBucket` are packed into a single storage slot. It is not expected that `withdrawCount` or `maxWithdrawPerBucket` would require the full range of `uint128` but could rather be `uint64`, thereby ensuring that only a single storage slot is utilized.

Client response

All the above recommendations were implemented in commit 1cd53d6.

6. Specification

The following section outlines the system's intended functionality at a high level, based on its implementation in the codebase. Any perceived points of conflict should be highlighted with the auditing team to determine the source of the discrepancy.

`WithdrawBridgeIntent`

The `WithdrawBridgeIntent` smart contract is designed to facilitate automatic token withdrawals from user-owned smart contract wallets (`LightAccounts`) on the Derive network via cross-chain bridges. It enables trusted executors to initiate withdrawals using either the Socket or LayerZero (OFT) bridge infrastructure, transferring approved ERC20 tokens from a user's wallet to their corresponding recipient address on another chain.

Users must pre-approve this contract to spend their tokens, and the executor must be registered and trusted by the contract's owner. The contract validates that the recipient matches the wallet owner and includes a `maxFee` check to ensure the user isn't overcharged for bridge fees. It supports two execution methods: `executeWithdrawIntentSocket()` for Socket-based bridging and `executeWithdrawIntentLZ()` for LayerZero, each requiring bridge-specific parameters and enforcing the same core validations.

`SubaccountDepositIntent`

The `SubaccountDepositIntent` smart contract facilitates automated token deposits from users' `LightAccounts` into `Subaccounts` using a permissioned execution flow. The user must first approve this contract to spend their tokens. The core logic ensures that deposits can only be made into subaccounts owned by the initiating `LightAccount`.

`StakeDRVIntent`

The `StakeDRVIntent` smart contract is designed to enable automated staking of the DRV token on behalf of users, specifically from `LightAccounts`.

The core function, `executeStakeDRVIntent`, can only be called by an authorized intent executor. When invoked, it transfers the specified amount of DRV from the user's `LightAccount` to the contract and immediately stakes it using the `convertTo` function of the staking interface, crediting the staked tokens to the original user.