PUBLIC

Code Assessment

of the Spark Savings Intents Smart Contracts

March 11, 2026

Produced for

Sky

by

CHAINSECURITY

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 7
4 Terminology 8
5 Open Findings 9
6 Informational 10
7 Notes 13

1 Executive Summary

Dear all,

Thank you for trusting us to help SparkDAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Spark Savings Intents according to Scope to support you in forming an opinion on their security risks.

SparkDAO implements Spark Savings Vault Intents, a contract designed to facilitate large withdrawals from Spark Savings Vaults.

The most critical subjects covered in our audit are access control and functional correctness. Security regarding all the aforementioned subjects is high.

The general subjects covered include usage and integration considerations. Please refer to the Informational and Notes sections for more information.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit, which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Spark Savings Intents repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	26 Feb 2026	2525ad01f076b01754ef1b6f5c5925e90c6592a4	v1.0.0-beta.0
2	03 Mar 2026	d9045fcaabfe162be5c7d7e4b062765b09a7f45a	v1.0.0

For the Solidity smart contracts, the compiler version `0.8.27` was chosen.

The following files were in scope of this review:

```
src/
    interfaces/
        IERC4626Like.sol
        ISavingsVaultIntents.sol
    SavingsVaultIntents.sol
```

2.1.1 Excluded from scope

Any other files were out of scope.

The Spark Savings Vault is covered in another ChainSecurity Review.

The following caveats of integrated tokens are regarded as known risks:

1. Both USDC and PayPalUSD can be paused. In such cases, no request on the respective vault can be fulfilled.
2. Both USDC and PayPalUSD have a blocklist. If the vault or the recipient is blocked, respective fulfillment cannot be processed.
3. Both USDC and PayPalUSD are deployed as proxies, hence bringing inherent centralization risks.
4. USDT may turn on the fee switch. In such a case, the fulfillment amount may be lower than `vault.minIntentAssets`.

Other tokens integrated in the future are out of scope and expected to be carefully analyzed.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

SparkDAO offers Spark Savings Vault Intents, a contract designed to facilitate large withdrawals from Spark Savings Vaults.

Users can deposit into ERC-4626 compliant Spark Savings Vaults, whose liquidity will be allocated by SLL (Spark Liquidity Layer) for optimized yield. However, instant large withdrawals may not be possible due to insufficient idle liquidity in the Spark Savings Vault.

Spark Savings Intents introduces an intent-based, relayer-aided, two-step withdrawal mechanism:

1. **User Request Withdrawals** (`request()`): A new withdrawal can be requested for a whitelisted vault. Lower and upper bounds are defined. The user must have sufficient balance and must have granted sufficient allowance to this contract.
2. **User Cancel Withdrawals** (`cancel()`): An existing request can be cancelled by the withdrawal initiator at any time.
3. **Relayer Fulfill Withdrawals** (`fulfill()`): When sufficient idle funds are ready, the *relayer* can fulfill the request before its expiration. The contract eventually performs a `redeem()` call to the Spark Savings Vault.

Two privileged roles exist in the system:

1. The *admin* (`DEFAULT_ADMIN_ROLE`) can update the configurations and manage other roles.
2. The *relayer* (`RELAYER`) can fulfill requests.

In Spark Savings Intents, vaults need to be explicitly whitelisted and are configured with `minIntentAssets` and `maxIntentAssets` to limit the withdrawal amount. A `maxDeadlineDuration` is defined to limit the request lifetime. Both parameters can be updated by the *admin* with `updateVaultConfig()` or `setMaxDeadlineDuration()`. Note that vault whitelisting, as well as the asset boundaries, is only enforced at request creation time (e.g. requests for unwhitelisted vaults can be fulfilled / cancelled).

Note that a user can have at most 1 pending withdrawal request per vault and can replace their pending request with a new one.

2.3 Trust Model

- `DEFAULT_ADMIN_ROLE`: fully trusted; assumed to be the SparkDAO Subproxy. In the worst case, it can DoS the system or whitelist a malicious contract as a vault.
- `Relayer`: semi-trusted; expected to be the **ALM Planner**. In the worst case, it may DoS the system by not processing the withdrawals.
- `User`: untrusted; can create, update, or cancel withdrawal requests at any time.
- Vaults: semi-trusted; expected to be Spark Vaults V2 instances and are expected to operate accordingly.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe our findings. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

6.1 Flashloan Bypass of Minimum Balance Spam Prevention

Informational | Version 1 | Acknowledged

CS-SPKSI-001

Any user can temporarily satisfy the `minIntentAssets` and balance check in `request()` with flashloans, allowing an attacker to create withdrawal requests at negligible cost and bypass the economic spam barrier.

More specifically, a user can flashloan the underlying asset and deposit it into the vault to obtain shares so that `request()` succeeds. Immediately after, the user undoes the setup. This contradicts the README's security rationale, which explicitly claims the high `minIntentAssets` threshold makes spamming "economically impractical" because "the user would need to hold a significant amount of vault shares".

Acknowledged:

SparkDAO responded:

```
this is more to make it less convenient to spam the contract.
```

6.2 Fulfill on De-Whitelisted Vaults

Informational | Version 1 | Acknowledged

CS-SPKSI-003

Currently, a relayer can fulfill a pending request even after the admin has de-whitelisted the vault via `updateVaultConfig()`.

Acknowledged:

SparkDAO responded:

```
It was determined that we want to keep existing functionality to minimize complexity.
The relayer can just stop fulfilling requests to blacklisted vaults.
```

6.3 Gas Griefing Vectors by Requesters

Informational | Version 1 | Acknowledged

CS-SPKSI-002

While withdrawal request creation validates balance, allowance, and deadline constraints, several conditions may change or otherwise cause fulfillment to fail:

1. The user may revoke the allowance granted to Spark Savings Intents.
2. The user may transfer their shares to another account.
3. The user may set a tight `deadline`, leaving the relayer insufficient time for liquidity orchestration and fulfillment.
4. The user may update or cancel the request, invalidating the `requestId` the relayer is attempting to fulfill.
5. The `recipient` or `account` may be intentionally blacklisted (e.g. USDC, PayPalUSD), causing the `redeem` call during fulfillment to revert.

Consequently, these requests cannot be fulfilled by the relayer. This may lead to gas griefing of the relayer if it does not check the conditions offchain before liquidity orchestration and request fulfillment.

Acknowledged:

SparkDAO responded:

```
These are all accounted for and checked offchain by the relayer.
```

6.4 Silent Request Override Without Cancellation Event

Informational **Version 1** **Acknowledged**

CS-SPKSI-004

When a user calls `request()` while they already have a pending request for the same vault, the old request is silently overwritten. Offchain systems tracking `RequestCreated` and `RequestCancelled` events will have a stale view because no `RequestCancelled` event is emitted for the replaced request.

Acknowledged:

SparkDAO responded:

```
Offchain systems just track for latest events.
```

6.5 Vault Configuration and Intent Asset Bounds Design

Informational **Version 1** **Acknowledged**

CS-SPKSI-005

1. `VaultConfig.whitelisted` could be simplified by letting `minIntentAssets` / `maxIntentAssets` imply whitelisting instead (e.g. a zero max meaning not whitelisted).

2. Given the current code, `minIntentAssets < maxIntentAssets` must hold in `updateVaultConfig()` and would not allow such a definition. This disallows the `==` case. Permitting `minIntentAssets == maxIntentAssets` would enable fixed-size intents and could also serve as a natural non-whitelist signal when both are zero.
3. Though `maxIntentAssets` is defined, a user can bypass it by creating multiple requests from different addresses. As such, it may not provide a meaningful economic constraint in practice.

Acknowledged:

SparkDAO responded:

```
1 and 2 are technically true but do not affect functionality materially,
and 3 is intended as a barrier.
```

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Actual Fulfillment May Exceed Vault Config

Note Version 1

When a request is created, the asset amount to be withdrawn is validated against the defined min and max intent bounds. However, at the time of fulfillment:

1. Assets received upon redemption may exceed the upper bound due to interest accrual between request creation and fulfillment.
2. Assets received upon redemption may fall below the lower bound due to token transfer fees (i.e. if USDT turns on transfer fees). However, the intended amount will respect the lower bound for Spark Vaults.

7.2 Intent Fulfillment Can Be Front-Run

Note Version 1

The documentation specifies that the *relayer* will orchestrate liquidity and that the fulfillment of the redemption will be atomic. While not stated explicitly, both stages should ideally be atomic to prevent front-running the fulfillment. Otherwise, the following scenarios may occur:

- Gas griefing vectors by requesters, which may lead to idle funds lying in the ALM Proxy or the Spark Savings Vault.
- Griefing vectors by other parties who leverage the available funds to perform instant withdrawals.

Note that the atomicity of orchestration and fulfillment cannot always be guaranteed due to the nature of the redemption of certain assets. For instance:

1. Redemption of RWAs may require multiple steps and may be asynchronous, e.g. Centrifuge, Maple, SuperState, and BUIDL.
2. Redemption of staking assets is subject to a delay, e.g. wstETH, weETH, and sUSDe.
3. Retrieval of bridged assets from foreign chains is subject to a delay, e.g. bridging assets via LayerZero or CCTP.

Thus, the offchain planner should aim to orchestrate the fulfillment of intents in an atomic manner, but in cases where this is not possible, it should implement appropriate measures to mitigate the risks of front-running and griefing.

7.3 Whale Withdrawal Intents May Enable Front-Running of ALM Liquidity Operations

Note Version 1

A whale holding significant vault shares can submit large withdrawal intents to force the ALM planner into predictable liquidity operations (e.g. unwinding specific positions or selling particular assets). Because

the ALM is expected to act promptly, especially if service-level commitments are in place, an adversary who understands the ALM's strategy could anticipate and front-run these market actions for profit.