CHAINSECURITY

PUBLIC

Code Assessment
of the L2MulticallSpell Smart Contracts

November 03, 2026

Produced for

Sky



Contents

1	Executive Summary	3
2	Assessment Overview	4
3	System Overview	5
4	Trust Model	6
5	System Considerations	7
6	Terminology	8
7	Findings	9
8	Limitations and use of report	10

1 Executive Summary

Dear Sky Team,

Thank you for trusting ChainSecurity with the review of L2MulticallSpell. As a globally trusted security partner for decentralized finance protocols and their governance, ChainSecurity provides specialized, independent technical analysis of complex smart contract systems. We greatly appreciate the opportunity to collaborate with Sky on its cross-chain governance.

L2MulticallSpell is a generic L2 spell for Sky governance. Deployed once per L2 and delegatecalled by the L2 governance relay, it executes a batch of calls encoded in the L1 spell, so that no dedicated L2 spell has to be deployed for simple actions.

We have conducted a thorough review of the scope. The most important areas covered in our review are:

- Functional correctness of `execute`, since it runs with the full privileges of the L2 governance relay: the calls are executed atomically and in order, revert data is bubbled correctly, and the deadline is enforced.
- The integration with every L2 governance relay the spell is intended for (OP Stack, Arbitrum, and LayerZero with and without timelock), since the spell's guarantees depend on how messages are delivered and retried. We highlight that anyone can retry a failed message until its deadline, see **Choosing the Deadline**.
- Access control, since `execute` is permissionless: the spell has no power of its own, as long as it is never granted any permission.

The report contains two system considerations and no issues.

In summary, we find that the codebase provides a high level of security, if operated under the assumptions documented in **System Overview**, **Trust Model** and **System Considerations**.

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 and our observations. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

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 L2MulticallSpell repository based on the documentation files. The code versions relevant to this report and when they were received are indicated below.

Version	Date	Commit Hash	Note
1	2 November 2026	ea3e8445eb4d489ac8ec710c79df36f69ed5c3f0	Initial review

For the Solidity smart contracts, the compiler version `0.8.24` was chosen, with the `cancun` EVM version and the optimizer enabled (200 runs, no `via_ir`).

The following file is in scope:

```
src/
└── L2MulticallSpell.sol
```

2.1.1 Excluded from scope

All other files in the repository, in particular the tests and the dependencies under `lib/`, are not in scope. The Sky L1 and L2 governance relays (`op-token-bridge`, `optimism-dai-bridge`, `arbitrum-dai-bridge` and `lz-governance-relay`), the LayerZero governance OApp and the underlying messaging layers (OP Stack, Arbitrum Nitro and LayerZero V2) are only referenced for context, to assess how the spell is executed. They were not reviewed and are assumed to behave as specified.

3 System Overview

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

Sky governance controls contracts on L2s through governance relays. An L1 spell calls the L1 governance relay of a domain, which sends a cross-chain message naming an L2 spell and the calldata to execute on it. On delivery, the L2 governance relay `delegatecall`s the L2 spell, so the spell's code runs with the relay's privileges. So far, each L2 action required a dedicated L2 spell to be written, reviewed and deployed before the L1 spell could be cast.

`L2MulticallSpell` is a generic, reusable L2 spell that removes this step. It is deployed once per L2, and the calls it makes are encoded in the L1 spell itself. The flow is as follows:

1. The L1 spell calls `relay()` on the L1 governance relay, with the `L2MulticallSpell` of that L2 as target and `execute(calls, deadline)` as data.
2. The bridge delivers the message to the L2 governance relay.
3. The L2 governance relay `delegatecall`s `L2MulticallSpell`. On the timelocked LayerZero relay, this happens only after a delay.
4. `execute` calls each target in order, with the L2 governance relay as `msg.sender`. It reverts if any call fails, if a target has no code, or after the deadline. The deadline limits how long a failed message can be retried or a timelocked action executed.

Calls that send ETH, or that depend on return values or on checks between calls, are not supported. They still require a dedicated L2 spell.

4 Trust Model

`L2MulticallSpell` has no access control, no state and no funds. Its power comes entirely from the L2 governance relay that delegatecalls it.

4.1 Roles

Sky Governance (L1 spells, through the PauseProxy)

- Trust level: Fully trusted.
- Permissions: Choose the L2 spell address, the calls and the deadline of every relayed message, and so execute arbitrary calls as the L2 governance relay.
- Assumptions: Every L1 spell is reviewed and tested against the target L2 before it is cast. The spell address, targets and calldata are correct for that L2. The deadline covers the delivery and timelock delay.
- Worst-case impact: Arbitrary actions on every contract the L2 governance relay controls.

L2 governance relays

- Trust level: Fully trusted. They are not part of this review.
- Assumptions: Only messages sent by the L1 governance relay are executed, and `msg.value` is zero.

Anyone

- Trust level: Untrusted.
- Worst-case impact: Retry a failed message or execute a ready timelocked action at a chosen time and with chosen gas. An older message can then run after a newer one, or a target with only part of its effect, see Choosing the Deadline.

4.2 External Dependencies

The messaging layers (OP Stack, Arbitrum Nitro, LayerZero V2) and the LayerZero governance OApp are trusted to deliver messages correctly and only from the L1 governance relay. The spell must never be granted any permission, as anyone could then use it to make arbitrary calls with that permission.

5 System Considerations

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

SC1 Choosing the Deadline

System Consideration | Version 1

Until the deadline, anyone can retry a failed message, and on the timelocked relay anyone can execute a ready action. The caller chooses when, and usually with how much gas. This means:

- An older failed message can run after a newer one and overwrite its changes.
- A target that ignores failing sub-calls, for example with `try` / `catch`, can be executed with only part of its effect by providing just enough gas.

Hence, deadlines should be short. However, the deadline is checked against the L2 `block.timestamp` when the calls are executed, so the L1 spell also has to consider:

- On the timelocked LayerZero relay, an action can only be executed after the delay. If the deadline ends before the delay, the action can never be executed.
- L2 timestamps can be inaccurate. For example, on Arbitrum the sequencer sets them, and they can lag behind L1 time by up to 24 hours.

SC2 Code Check Does Not Catch All Ineffective Calls

System Consideration | Version 1

`execute` reverts on targets without code, but some calls can still succeed without effect. Examples are an EOA with an EIP-7702 delegation to an address without code, or a target whose `fallback` accepts a wrong selector. Governance has to make sure every target and selector exists on the respective L2.

6 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

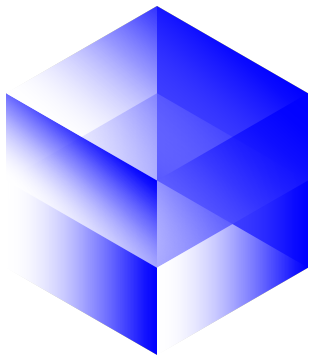
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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.

7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



No Open Issues

Total number of findings

Critical	High	Medium	Low	Informational
0	0	0	0	0

8 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.