PUBLIC

Code Assessment

of the Solana Crosschain Governance Payload Scripts

October 21, 2025

Produced for

Sky

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Sky with this review. Our executive summary provides an overview of subjects covered in our review of the Solana Crosschain Governance Payload Scripts according to Scope to support you in forming an opinion on the implementation.

Sky implements scripts for generating payloads for the Wormhole and LayerZero governance bridge to facilitate the migration from the Wormhole infrastructure to the new LayerZero infrastructure.

The most critical subject of this review was the correctness of the generated payloads and, thus, the correctness of the instruction generation and codec implementations. Other topics covered include the completeness of the validation.

In summary, we find that the codebase achieves the desired outcome.

It is important to note that this assessment is not a security audit but rather a review of whether the programs achieve the desired outcome. Thus, running the code in isolated environments is recommended. Last, it is important to note that reviews 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 following files were in scope of this review:

```
src/
    constants.ts
    index.ts
    lz-governance-codec.ts
    programs/
        bpf-loader-upgradeable.ts
        index.ts
    shared-governance-codec.ts
    simulation-assertions.ts
    simulation-utils.ts
    utils.ts
    wh-governance-codec.ts
scripts/
    ntt-transfer-mint-authority/
        generate-payload.ts
        validate.ts
    set-token-authority/
        generate-payload.ts
        validate.ts
    update-mpl-metadata-authority/
        generate-payload.ts
        validate.ts
    wh-program-upgrade/
        generate-payload.ts
        validate.ts
```

In Version 2 renaming occured and new files for the configs were added. Below is the full scope as of Version 2:

```
src/
    constants.ts
    index.ts
    lz-governance-codec.ts
    programs/
        bpf-loader-upgradeable.ts
        index.ts
    shared-governance-codec.ts
    simulation-assertions.ts
    simulation-utils.ts
    utils.ts
    wh-governance-codec.ts
scripts/
    ntt-transfer-mint-authority/
```

```
        config.ts
        generate-payload.ts
        validate.ts
    set-token-freeze-authority/
        config.ts
        generate-payload.ts
        validate.ts
    update-mpl-metadata-authority/
        config.ts
        generate-payload.ts
        validate.ts
    wh-program-upgrade/
        config.ts
        generate-payload.ts
        validate.ts
```

The assessment was performed on the source code files inside the Solana Crosschain Governance Payload Scripts 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	10 Oct 2025	d6e88a4dd82515024941ec8f30d17e75768742d4	Initial Version
2	16 Oct 2025	0d21ba8ff6877de0ae1ea922597278a3d3e7c551	After Intermediate Report
3	21 Oct 2025	e9ec2e05fddde75941b11baef383326acf61039a	Final Version

2.1.1 Excluded from scope

This review is focused on the correctness of the codecs and payload generation scripts, and the completeness of the validation scripts. The programs involved in the scripts are excluded from the scope, in particular:

- The Wormhole bridge programs, SPL Token program, MPL Metadata program, and the BPF Upgradeable Loader program are not in scope.
- The LayerZero bridge programs are covered in separate ChainSecurity reviews.

The actual payload generated for Sky governance spells are not in scope, and the governance should verify the payload and accounts eventually used in the spells.

The libraries used are out of scope and are expected to work as documented (or as expected if not documented sufficiently). The security of the codebase is out-of-scope, and it is advised to run it in an isolated environment.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the typescript scripts 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.

Sky offers a set of scripts for the construction of cross-chain governance payload used in Sky's governance spells to relay governance actions to Solana.

2.2.1 Governance Overview

Sky relies on bridges to send and execute governance messages to chains other than Ethereum. The initial message passing bridge, based on Wormhole, is being migrated to a LayerZero-based infrastructure. The general workflow of the message passing is outlined below, but details may vary according to the bridge:

1. **Message Initiation On Source Chain:** Sky Governance authorizes and encodes a cross-chain message (i.e. accordingly formatted payload), which is sent via the respective bridge contracts.
2. **Message Verifications:** Off-chain attestations are generated over the cross-chain message by respective guardians or validators. Eventually the message is delivered on the destination chain and ready to be executed.
3. **Governance Action Execution:** The governance action will be executed in a privileged context according to the bridged payload.

The scripts aim to help generate and validate the payloads for the migration process.

2.2.2 Governance SDK

The `src` directory defines helpers used by the scripts. Below is a summary of functionality provided:

- **Instruction Codec**: Two governance codecs have been implemented to support the different data layouts of Wormhole and LayerZero cross-chain messages. These allow for serializing instructions to Solana payloads in the respective codec (used as a parameter for the cross-chain message). Further, they allow for deserializing the payloads to instructions (used for validation in scripts).

  Note that the codec is defined within the respective codebases but is outlined within the comments. Additionally, deserialization is implemented as well as various other helpers.
- **Simulation**: Simulation helpers for RPC integration and LiteSVM are provided. Additionally, assertions for accounts are implemented.
- **Instruction Builders**: Helpers for building instructions are provided where necessary.
- **Other**: Various other helpers are implemented (e.g. filtering, generation of instructions).

2.2.3 Governance Actions

For each action in the `scripts` directory the following functionality is provided:

- `generate_payload`: Script that prepares the instruction and generates the Solana payload in the expected codec.
- `validate`: Script that simulates the execution of the payload and performs various assertions to validate correctness.

The following actions are provided:

- **Wormhole Program Upgrade:** Prepares a WH Solana payload that allows the upgrade authority to upgrade Wormhole NTT program to a new implementation. Note the buffer account that store the new program code is expected to be created in advance, which should be validated off-chain.
- **NTT Transfer Mint Authority:** Prepares a call to the `transfer_mint_authority` instruction to transfer the token mint authority from Wormhole NTT to another account.
- **Set Token Authority:** Prepares a call to the SPL token program to transfer the Freeze Authority to another account.

- **Update MPL Metadata Authority:** Prepares a call to the Metaplex Token Metadata program to update the authority, which has the privilege to update the token metadata.

2.3 Trust Model

The scripts are provided as a permissionless tool to prepare and simulate the governance message payload off-chain. Note:

- The generated execution payload may not match the payload attached in the on-chain governance spells.
- The off-chain simulation may not reflect the actual on-chain behavior due to potential context change or message delivery issue.

Sky Governance is fully trusted to inspect and validate the actual payload used in the spells is correct. **Wormhole:** Wormhole itself is out of scope for this review and assumed to work as documented. The Guardians are fully trusted to deliver and execute the governance cross-chain messages honestly, otherwise the Solana governance contract can be manipulated to execute unexpected instructions. **LayerZero:** LayerZero is out of scope for this review and is trusted to behave correctly and deliver messages to the correct destination. The DVNs are trusted to attest the governance message. The LayerZero executor is trusted to always deliver messages with the correct provided message value and gas limit. Depending on the exact configuration, LayerZero might be fully trusted to not censor messages or similar. However, the configuration is out-of-scope for this review. **RPC:** The RPC used is semi-trusted; which may provide wrong response that lead to incorrect simulation result. The user can always switch to another RPC in case it is malicious.

Further, note that users of the scripts should run the scripts in a protected environment.

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 any open findings. Findings that have been resolved have been moved to the Resolved Findings section. 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 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

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
Informational Findings	4

- Sentinel Account Not Used Code Corrected
- Account Assertions Inconsistent Code Corrected
- Data and Code Are Tightly Coupled Code Corrected
- Inconsistent Account Validation Code Corrected

6.1 Sentinel Account Not Used

Informational Version 2 Code Corrected

CS-SKY-SCGP-006

The payload generation for `update-mpl-metadata-authority` does not use the `WH_OWNER_SENTINEL_KEY` for the update authority.

Code corrected:

Code has been corrected to use `WH_OWNER_SENTINEL_KEY` for the update authority.

6.2 Account Assertions Inconsistent

Informational Version 1 Code Corrected

CS-SKY-SCGP-002

`assertNoAccountChanges()` defines does not raise if one of the following conditions holds:

```
if (!accountInfoBefore && !accountInfoAfter) {
    return;
} else if (!accountInfoBefore) {
    assertAccountEmpty(accountInfoAfter);
    return;
}
```

Mainly, this corresponds to the following:

- If the before and after values are null, they are treated as equal.

- Else if the account before is null, the account after must be empty.

However, this does not consider the scenario where `accountInfoBefore` could be non-null, but the account after could be null.

Ultimately, it could be simpler to simply return if both accounts are considered to be empty.

Code corrected:

Code has been changed to assert both accounts are empty if any account is null.

6.3 Data and Code Are Tightly Coupled

Informational | Version 1 | Code Corrected

CS-SKY-SCGP-003

In the scripts, all the payload generation scripts will place the payload in the same `output.json`, further the accounts and payload are hardcoded as constants in place, requiring users to manually copy-paste or switch.

The data and code can be decoupled to improve the overall readability:

- The payload generation can write to different json files, from which the respective validation scripts can read.
- The constants can be grouped into different json files to reflect the different stages, i.e. devnet testing and mainnet simulation.

Code corrected:

The codebase has been improved. Now the following is implemented:

- Payload generation does not write JSONs anymore but writes the hex string to a custom file for the given action.
- The constants are now grouped in types according to staging.

6.4 Inconsistent Account Validation

Informational | Version 1 | Code Corrected

CS-SKY-SCGP-004

The validation scripts (`validate.rs`) simulates the execution of a payload and validates the pre- and post-state are as expected. However, the accounts' validation in different scripts are inconsistent.

The `wh-program-upgrade` script does not validate the unchanged accounts with `assertNoAccountChanges()`, though this validation is performed in other scripts (e.g. `update-mpl-metadata-authority`). Additionally, more checks could be added to other scripts to ensure the expected end state (e.g. checks applicable from `assertNoAccountChanges()`).

Code corrected:

Since Version 2, while more checks have been introduced, an inconsistency remains: `wh-program-upgrade` does not validate `programUpgradeAuthority` account states while all other actions validate `authority`.

Since Version 3, state validation of `programUpgradeAuthority` has been added in `wh-program-upgrade`.

Below, are additional checks missing but are not necessarily required for various reasons:

- `wh-program-upgrade` checks the NTT program ID while `ntt-transfer-mint-authority` does not. Note that this is due to the LiteSVM validation process.
- Only the account data but no other value of `AccountInfo` is validated for the below accounts:
    - `tokenMint` in actions `ntt-transfer-mint-authority` and `set-token-freeze-authority`.
    - Metadata account in action `update-mpl-metadata-authority`.
    - `programDataAddress` in action `wh-program-upgrade`.
- In `wh-program-upgrade`, only the lamports are checked for `newProgramBuffer` and `spillAccount`.
- governanceProgramId``and external programm accounts (i.e. ``TOKEN_PROGRAM_ID`, `mplProgramAddress`) are never checked.

7 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.

7.1 Missing Input Length Validation

Informational | Version 1 | Code Partially Corrected

CS-SKY-SCGP-005

In general, the helper functions in `src` that accept input with variable length do not validate the input length. Depending on the operations performed, a malformed input may be unintentionally accepted or not. Some of the cases are highlighted below:

1. `deserializeAccountFromBytes()`: If the payload buffer is shorter than expected, the public key may be initialized with less than 32 bytes and the `isSigner` and `isWritable` flag may be initialized with un-allocated memory which will be interpreted as false.
2. `generateSentinelPubkey()`: If the input string is longer than 32 bytes, the call to `buf.set()` will revert.
3. `deserializeInstruction()`:
    1. In Wormhole Codec: If the input payload is longer, the garbage at the end will be ignored. If the input is shorter, it may succeed with truncated data.
    2. In LayerZero Codec: If the input payload is longer, the garbage at the end will be regarded as calldata. If the input is shorter, it may succeed with truncated data.

Code partially corrected:

1. Corrected.
2. Partially corrected. Now it is enforced that the string length is at most 32. However, the length in bytes could exceed that.
3. Partially corrected. Now in Wormhole codec it checks the sufficient calldata must be provided given the decoded calldata length.

8 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.

8.1 Constants For Testing Only

Note Version 1

Note that the codebase defines constants. These are not the final ones and should all be updated before using the scripts for actual spells.