PUBLIC

Code Assessment

of the Gelato Smart Wallet Smart Contracts

June 25, 2025

Produced for

Gelato

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Gelato Team,

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

Gelato implements an EIP-7702-compatible Delegation smart wallet that also supports full account abstraction via EIP-4337, allowing an externally-owned account (EOA) to delegate control to smart-contract logic while retaining the same address.

The most critical subjects covered in our audit are functional correctness access control. Security regarding all the aforementioned subjects is high. Our most important finding is Transient validator variable override which allows some checks to be bypassed as described in the issue. Unit testing is minimal, a deeper testing would likely have revealed this issue and might help to uncover other issues.

The general subjects covered are code EIP compliance, error handling, and correct integration. Security regarding all these subjects is high.

In summary, we find that the codebase currently provides a high level of security, but we recommand improving the tests to increase the confidence.

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	1
• Code Corrected	1
Low-Severity Findings	1
• Code Corrected	1

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 Gelato Smart Wallet 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	13 June 2025	2e4018e9880ccfae7f49b26cba176aec32878158	Initial Version
2	19 June 2025	0a83914bb8bd3af3b4fcffc11526a3fc21edbcaa	Issues Fixes

For the solidity smart contracts, the compiler version `0.8.29` was chosen.

The following contracts are in the scope of this review:

```
src/Delegation.sol
src/types/Constants.sol
src/interfaces/IERC20.sol
src/interfaces/IERC1271.sol
src/interfaces/IERC4337.sol
src/interfaces/IERC7821.sol
src/interfaces/IValidator.sol
```

In (**Version 2**), the following contracts were added:

```
src/interfaces/IERC165.sol
src/interfaces/IERC721.sol
src/interfaces/IERC1155.sol
```

2.1.1 Excluded from scope

Any contracts or files not listed above are excluded from the scope and are therefore not covered.

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.

Gelato's Delegation smart contract is designed to be used as an EIP-7702 wallet contract, that is, as the executable code of an EOA. It also supports full account abstraction via EIP-4337. The contract additionally uses the following EIP standards:

- EIP-1155: Multi-token receiver interface via `onERC1155Received`

- EIP-721: NFT receiver interface via `onERC721Received`
- EIP-1271: Smart contract signature verification via `isValidSignature`
- EIP-7821: Minimal batch execution interface via `execute` and `supportsExecutionMode`
- EIP-7201: Namespaced storage layout to prevent collisions with internal nonce tracking

Furthermore, the contract supports call types encoded with the prefix `0x01` for batch calls and `0x00` as execution type for execution that reverts on failure, as specified in EIP-7579. The mode selector can either be `0x00000000` for the default mode or `0x78210001` for the opData mode, as described in EIP-7821.

Users can interact with the Delegation contract in four ways:

- directly, or
- via the entry point contract (optionally using a validator for signature validation)
- with optional data including the EOA's signature
- with optional data including a signature that can be validated by an approved validator (requires a registered validator)

When a user sets the contract as the delegate of an EOA via the set code transaction, the contract will be able to execute transactions on behalf of the EOA and have the same address as the EOA. This allows the user to call the Delegation contract directly and execute functions protected with the `onlyThis` modifier (`msg.sender == address(this)`). It allows the user to call the administrative functions to set and delete validators and to invalidate nonces. Furthermore, it enables anyone to call the Delegation contract and act as the EOA if they have a valid `opData` object with a valid signature. The valid signature can either be the signature of the EOA or a signature that passes the validator signature check. If the caller can provide a valid signature, the Delegation contract will act as the EOA and execute the provided calls.

In case the user uses optional data mode (`EXEC_MODE_OP_DATA`), the Delegation contract manages its own nonces.

Additionally, the contract can be used as an abstract account in combination with the EIP-4337 standard. In this case, the contract needs to be called via the entry point contract of the respective EIP-4337 enabled chain. The entry point will initiate the signature checks and the calls. As above, a validator can be set before calling the entry point, allowing more signature validation options.

In summary, the contract has the administrative functions to manage the validators, the capability to invalidate nonces, and the ability to execute calls on behalf of the EOA in the four ways.

2.3 Trust Model

First, the account generation of the EOA is assumed to be fully trusted, including the management or destruction of the corresponding private key. Validators could allow any arbitrary call. Therefore, they are also assumed to be fully trusted. Users must carefully assess the contracts used as validators. Regarding the Solady library used, we assume it to be secure and trusted. However, we highlight that it is still marked as experimental software and serves as a laboratory for cutting-edge snippets, and hence, needs to be used with care. In case of switching the delegation contract, we assume the previous validators are still trusted until the delegation switch is successfully completed and the new set of validators is in use.

Besides, we assume the entry point and the underlying technology stack to be fully trusted.

2.4 Version 2 changes

The EIP-165 is now supported via the `supportsInterface` function and returns `true` for compatible interfaces.

The entry point now uses `executeUserOp` to execute user operations, and does not use the `execute` function anymore.

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:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Correctness: Mismatches between specification and implementation

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	1

- Transient Validator Variable Override Code Corrected

Low-Severity Findings	1

- ERC-1155 Compliance Code Corrected

Informational Findings	2

- Event Indexing Code Corrected
- Unused Error Code Corrected

6.1 Transient Validator Variable Override

Security Medium Version 1 Code Corrected

CS-GEL-001

The contract keeps a single:

```
IValidator transient transientValidator;
```

which is written in every `validateUserOp` and read later in `execute`.

Per EIP-4337, `EntryPoint.handleOps`

1. loops over all `UserOperation`, calling `validateUserOp` on each.
2. only after the final validation starts the execution loop.

Therefore, when two (or more) operations from the same account appear in one bundle:

- the last `validateUserOp` overwrites `transientValidator`.
- the first `execute` then sees the wrong validator.

This results in a potential bypass of critical checks in `postExecute`.

Code corrected:

Validators are added to a transient mapping, which is then used in `executeUserOp` to retrieve the correct validator for each operation.

6.2 ERC-1155 Compliance

Correctness **Low** **Version 1** **Code Corrected**

CS-GEL-002

The ERC-1155 specificiation states:

> Smart contracts MUST implement all of the functions in the `ERC1155TokenReceiver` interface to accept transfers. See "Safe Transfer Rules" for further detail.
>
> Smart contracts MUST implement the ERC-165 `supportsInterface` function and signify support for the `ERC1155TokenReceiver` interface to accept transfers. See "ERC1155TokenReceiver ERC-165 rules" for further detail.

However, the contract does not implement the `supportsInterface` function.

Code corrected:

`supportsInterface` has been implemented in the contract, meeting the ERC-1155 specification requirements.

6.3 Event Indexing

Informational **Version 1** **Code Corrected**

CS-GEL-003

The two events:

- `ValidatorAdded(IValidator validator)`
- `ValidatorRemoved(IValidator validator)`

could be indexed to allow better filtering of events.

Code corrected:

The two events are now indexed.

6.4 Unused Error

Informational **Version 1** **Code Corrected**

CS-GEL-004

The error `InvalidSignatureLength` is not used in the contract.

Code corrected: The error was removed.

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 Complex Gas-Dependent Calls

Note Version 1

The `Delegation` contract does not allow callers to specify how much gas is forwarded to each low-level `call` executed inside `_executeCalls`. In most situations this is acceptable, but there are edge-cases where a user must guarantee a minimum gas stipend (or cap the gas forwarded) for a particular downstream call. That flexibility is currently not provided in the implementation.

7.2 Independent Nonce Management

Note Version 1

The Delegation contract implements its own independent nonce management besides the already existing nonce management in the Entrypoint. As there is not synchronization between the nonces, users must be careful when signing two different operation modes. The order is only enforced separately and not across operation modes.

7.3 Transaction Ordering

Note Version 1

If a user constructs two execution payloads and a transaction within the first payload triggers the `execute()` function again, the second payload may be executed before the first one completes. This can lead to unexpected behavior if the second execution depends on state that is only established by the first. Such reentrant flow breaks assumptions about sequential state progression.