certora

Security Assessment Report

Safe Allowance Module

February 2026

Prepared for Safe

Table of Contents

Project Summary

Project Scope

Project Name	Repository Link	Commit Hash	Final Commit	Platform
Safe Allowance Module	https://github.com/safe-fndn/safe-modules/	252e864	2d09764	Solidity

Project Overview

This document describes the security review of **Safe Allowance Module**. The work was undertaken from **February 12th, 2026** to **February 16th, 2026**.

The following contract list is included in our scope:

modules/allowances/contracts/AllowanceModule.sol
modules/allowances/contracts/SignatureDecoder.sol
modules/allowances/contracts/Enum.sol

The team performed a manual audit of all the files in scope**.** During the manual audit, the Certora team discovered bugs in the code, as listed on the following page.

Protocol Overview

The Safe Allowance Module is an extension for Safe multisig wallets that enables granular token spending permissions. Safe owners can designate delegates and grant them recurring or one-time allowances for specific tokens. Delegates can then execute transfers from the Safe — either directly or through relayers using off-chain signatures.

Assessment Methodology

Our assessment approach combines design level analysis with a deep review of the implementation to ensure that a protocol is secure, economically sound, and behaves as intended under realistic conditions.

At the design level, we evaluate the architecture, the economic assumptions behind the protocol, and the safety properties that should hold independently of a specific chain or environment. This process includes reviewing internal and cross protocol interactions, state transition flows, trust boundaries, and any mechanism that could be exploited to extract value, deny service, or alter core system behavior. At this stage, a focused threat modelling exercise helps identify key attack surfaces and adversarial capabilities relevant to the system. Design level issues often relate to incentive structures, governance implications, or systemic behavior that emerges under adversarial conditions.

Implementation analysis focuses on the concrete behavior of the code within the execution model of the target chain. This involves reviewing the correctness of logic, access control, state handling, arithmetic behavior, and the nuanced behaviors of the chain environment. Familiar classes of vulnerabilities such as reentrancy conditions, faulty permission checks, precision issues, or unsafe assumptions often surface at this layer. These findings require context aware reasoning that takes into account both the code and the architectural intent.

To support this analysis, the codebase is examined through repeated manual passes and supplemented by automated tools when appropriate. High-risk logic areas receive deeper scrutiny, invariants are validated against both design intent and actual implementation, and potential vulnerability leads are thoroughly investigated. Automated techniques such as static analysis, fuzzing, or symbolic execution may be used to complement manual review and provide additional insight.

Collaboration with the development team plays an important role throughout the audit. This helps confirm expected behaviors, clarify design assumptions, and ensure an accurate understanding of the protocol's intended operation. All findings are documented with clear reasoning, reproducible examples, and actionable recommendations. A follow up review is conducted to validate the applied fixes and verify that no regressions or secondary issues have been introduced.

Threat and Security Overview

The AllowanceModule is designed to give token allowance on a Safe to explicitly declared delegates, granting them the ability to sign a message off-chain that can be used to move funds from the smart wallet.

Fundamental Security Requirement

Only registered delegates may authorize transfers from the Safe — either by calling executeAllowanceTransfer directly or by providing a valid off-chain signature that a relayer submits. Transfers are bounded by the delegate's allowance.

Core Threat Model

Since the system relies on signatures and maps delegates and allowances in different mappings, it must be impossible to:

- replay an old signature
- forge a signature the contract will find valid
- manipulate `allowances` and `delegates`
- execute an unauthorized token transfer

Attack Vectors Investigated

- Reentrancy in `executeAllowanceTransfer`, when the `paymentToken` is in native ETH, via callback on `tx.origin` allowed with the introduction of EIP-7702. The vector has been unfruitful due to the function respecting the Checks-Effects-Interactions pattern.
- Collision with the key derived from the `delegate` address. The likelihood of a collision happening is low but theoretically possible.
- Overflows and underflows, relevant due to the Solidity version of the contract.
- Silent truncation during type down-casting.

- Signature replayability.

The contract is resilient to the listed attack vectors and the issues detected are related to code maintenance and user mistakes.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	-	-	-
Informational	3	3	3
Total	3	3	3

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
I-01	AllowanceModule::removeDelegate fails to check delegate key collision	Informational	Fixed
I-02	SignatureDecoder::recoverKey isn't used anywhere	Informational	
I-03	Missing natpsec on several functions	Informational	

Informational Severity Issues

I-01: AllowanceModule::removeDelegate fails to check delegate key collision

Files:

contracts/AllowanceModule.sol#L332

Description:

The code maps users' delegates by identifying them using the lowest 48 bits of the delegate address as a key.

When setting an allowance with setAllowance there is a check on the delegate parameter to ensure the address the key resolves to corresponds to delegate. In case they don't match, the transaction reverts to prevent setting an allowance to a wrong address that would resolve to the same key due to a collision.

However, the removeDelegate function doesn't implement such a check meaning it is possible to supply a wrong delegate address parameter that would resolve to the same key.

As a consequence, the allowance mapping could persist leaving only the delegates mapping as deleted.

Persisted allowances are usually stale after delegate removal (spending path checks delegate registration), but they remain in storage and can become relevant again if that delegate is re-added later.

Scenario

```
address A = 0x1111111111111111111111111111deadbeefcafe; // real delegate
address B = 0x2222222222222222222222222222deadbeefcafe; // wrong delegate,
collides with A
uint48 keyA = uint48(uint160(A)); // 0xdeadbeefcafe
uint48 keyB = uint48(uint160(B)); // 0xdeadbeefcafe
// keyA == keyB
```

For one Safe S:

1. `addDelegate(A)`:
`delegates[S][0xdeadbeefcafe].delegate = A`

2. `setAllowance(A, USDC, 100, ...)`
Passes, because `delegates[S][keyA].delegate == A`

3. `removeDelegate(B, true)`
No equality guard exists here.
It reads `delegates[S][keyB]` (same slot) and gets A, so it proceeds.
Cleanup loop uses `tokens[S][B]` (usually empty), so `allowances[S][A][USDC]` is not cleared.
Then it deletes `delegates[S][keyB]` (same as keyA), removing delegate entry for A.

Result:

- `delegates` entry is deleted
- `allowances[S][A][USDC]` can remain in storage (stale/orphaned)

Recommendations:

The issue illustrated in the example can be mitigated by executing deleteAllowance to effectively clean up the orphaned allowance. However we still recommend implementing the same check applied in setAllowance against the delegate parameter in removeDelegate.

Customer Response:

Fixed in PR 523.

Fix Review:

Fix confirmed.

I-02: SignatureDecoder::recoverKey isn't used anywhere

Files:

contracts/SignatureDecoder.sol

Description:

SignatureDecoder::recoverKey is currently unused anywhere and can be removed.

Recommendations:

Remove the unused function.

Customer Response:

Fixed in PR 524.

Fix Review:

Fix confirmed.

I-03: Missing natpsec on several functions

Files:

contracts/AllowanceModule.sol

Description:

There are several functions in AllowanceModule that are missing or have incomplete natspec.

- `generateTransferHash`
- `getTokens`
- `getTokenAllowance`
- `getDelegates`
- `getAllowance`
- `updateAllowance`
- `checkSignature`
- `recoverSignature`
- `transfer`

Recommendations:

We recommend adding natspec to all the above mentioned functions.

Customer Response:

Fixed in PR 525.

Fix Review:

Fix confirmed.

Disclaimer

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.