Security Audit Report

Axelar mvx-gas-service
MultiversX smart contract

by ARDA

on April 21, 2025

Table of Contents

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Terminology

Code: The code with which users interact.

Inherent risk: A risk for users that comes from a behavior inherent to the code's design.

Inherent risks only represent the risks inherent to the code's design, which are a subset of all the possible risks. **No inherent risk doesn’t mean no risk.** It only means that no risk inherent to the code's design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the upgradability of the code might also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Objective

Our objective is to share everything we have found that would help assessing and improving the safety of the code:

1. The **inherent risks** of the code, labelled R1, R2, etc.
2. The **issues** in the **code**, labelled C1, C2, etc.
3. The **issues** in the **testing** of the code, labelled T1, T2, etc.
4. The **issues** in the **other** parts related to the code, labelled O1, O2, etc.
5. The **recommendations** to address each issue.

Audit Summary

Initial scope

- **Repository:** https://github.com/multiversx/sc-axelar-cgp-rs
- **Commit:** `4b05e701ae050b6d10e936e43c289413d901a585`
- **MultiversX smart contract path:** ./gas-service/

Final scope

- **Repository:** https://github.com/multiversx/sc-axelar-cgp-rs
- **Commit:** `b863a1ba7fe8180e63961f721a63c6d53d818137`
- **MultiversX smart contract path:** ./gas-service/

1 inherent risk in the final scope

0 issue in the final scope

2 issues reported in the initial scope and 0 remaining in the final scope:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	2	0	0	0	0	0

Inherent Risks

R1: Users might be refunded incorrectly, or too late, or even not refunded at all for the excess of tokens they provided to cover gas costs.

This is because the tokens deposited in the Gas Service smart contract are fully managed by a single "collector" account, who might not perform the refunding as expected in the following situations:

- The collector could make mistakes: for example he could refund a wrong address, a wrong amount, or a wrong token.
- The collector could be manipulated: anyone with access to the collector’s private key can send all the funds deposited in the Gas Service to any address.
- The collector might be inactive: for example he might have lost access to his private key or might be unavailable, in which case refunds will not be processed.

Code Issues & Recommendations

C1: Mechanism to prevent withdrawing more fees than available does not work when fee token repeated twice

Severity: Minor **Status:** Fixed

Location

```
gas-service/src/lib.rs
    collect_fees
```

Description

Current behavior: In the endpoint `collect_fees`, there is a mechanism that skips the withdrawal of a given amount of tokens if this amount exceeds the smart contract's balance, in order to avoid a transaction failure when transferring the tokens. However, this mechanism might not work when a token appears twice in the list of tokens to withdraw.

Namely, when calling `collect_fees`, the collector specifies the list of tokens and amounts of fees he wants to withdraw. For each token, if the amount is bigger than the smart contract's balance, then the withdrawal of this amount is skipped, and then, at the end of `collect_fees`, all tokens are simultaneously transferred to the collector.

```
for index in 0..tokens_length {
    ...
    if amount <= balance {
        payments.push(EsdtTokenPayment::new(token, 0, amount));
    }
}
if !payments.is_empty() {
    self.send().direct_multi(receiver, &payments);
}
```

However, if the same token appears twice in the list of tokens, then even if `amount <= balance` holds for each individual amount, it is possible that the

sum of amounts exceeds `balance`, and in this case, the final transfer would fail and the transaction would fail.

Expected behavior: The endpoint `collect_fees` should be successfully executed even if some amounts to withdraw exceed the smart contract's balance. This is indeed a convention of the Gas Service smart contract deployed on other chains, see for example the Solidity version.

Example: While there are 100 USDC in the smart contract, the collector requests to withdraw 60 USDC twice, and the transaction fails because `60 + 60 = 120 > 100`.

Recommendation

We suggest reproducing the logic from the Solidity version: in `collect_fees`, instead of sending all ESDT payments at once at the end with `direct_multi`, we suggest sending each individual ESDT payment when it is processed using `direct_esdt`.

C2: Owner of smart contract can't change collector

Severity: Minor **Status:** Fixed

Location

```
gas-service/src/lib.rs
    set_gas_collector
```

Description

Current behavior: The owner can’t change the collector address, because only the collector himself is allowed to call `set_gas_collector`.

Expected behavior: The owner should be able to change the collector address, e.g. in case the collector becomes unreliable. On other blockchains, the owner of the Gas Service smart contract is able to change the collector. For example, on Ethereum, the owner would upgrade the Gas Service smart contract and incidentally change the collector address.

Recommendation

We recommend allowing the owner to call the endpoint `set_gas_collector`.