

SECURITY AUDIT REPORT

Axelar mvx-interchain-token- factory MultiversX smart contract


by  **ARDA**
on April 21, 2025



Table of Contents

Disclaimer	3
Terminology	3
Objective	4
Audit Summary	5
Code Issues & Recommendations	6
C1: "deploy_remote_token_callback" can fail and user loses gas payment for interchain call	6
C2: Initial supply might be minted after initialization of the token	8
C3: ESDT tokens are accepted in "deploy_interchain_token" but are unused and can be lost	10
C4: Token might be issued with name and ticker different from the ones asked by the user	12
C5: Remote deployment can be made with destination minter that was not asked by the user	13
C6: EGLD can't be registered as a custom token	15
C7: Unclear separation between deployment steps in "deploy_interchain_token"	17
C8: Function "ascii_to_u8" is unnecessarily complex	19
C9: Unnecessarily complex logic to extract ticker from token identifier	21
C10: Inconsistent and misleading way to not specify address argument	22
C11: No sanity check that a token is registered in Token Manager before registering on other blockchains	23
C12: Unused functions "its_deployed_token_manager" and "token_manager_token_identifier"	25

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:** 6ccc55290af7c2e3a14909e2bb331b113eef8ab3
- **MultiversX smart contract path:** ./interchain-token-factory/

Final scope

- **Repository:** <https://github.com/multiversx/sc-axelar-cgp-rs>
- **Commit:** b863a1ba7fe8180e63961f721a63c6d53d818137
- **MultiversX smart contract path:** ./interchain-token-factory/

0 inherent risk in the final scope

0 issue in the final scope

12 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	6	0	0	0	0	0
Minor	6	0	0	0	0	0

Code Issues & Recommendations

C1: "deploy_remote_token_callback" can fail and user loses gas payment for interchain call

Severity: Medium

Status: Fixed

Location

interchain-token-factory/src/proxy.rs
deploy_remote_token_callback

Description

Current behavior: When performing a token deployment on a destination blockchain, an asynchronous call to the ESDT System smart contract is made to retrieve the token's name, ticker and decimals, and then in the callback we register an interchain call to the ITS Hub.

However, the callback `deploy_remote_token_callback` might fail, and this would make the user lose all the tokens that were provided for paying the interchain call.

Namely, in `deploy_remote_token_callback`, there are requirements that

- The destination blockchain is trusted,
- The destination blockchain is different than MultiversX,
- The destination address is trusted,
- The ITS is not be paused.

Therefore, the callback would fail if any of these properties does not hold.

Expected behavior: The callback `deploy_remote_token_callback` should always succeed, so that either the interchain call is executed, or the user gets back the gas tokens he provided for the interchain call.

Recommendation

We recommend removing all the sources of failure from the callback `deploy_remote_token_callback` . For this, if the following properties do not hold, instead of failing, we would simply send back the gas tokens to the user and return:

- The destination blockchain is not trusted,
- The destination blockchain is MultiversX,
- The destination address is not trusted,
- The ITS is paused.

C2: Initial supply might be minted after initialization of the token

Severity: Medium

Status: Fixed

Location

```
interchain-token-factory/src/lib.rs  
    deploy_interchain_token
```

Description

Current behavior: The initial supply of a token might be minted after the completion of the token's initialization—that is, after the Token Manager was deployed, the token was issued, and the initial supply (if any) was minted.

Namely, even if the initialization is complete, the initialization endpoint `deploy_interchain_token` would not fail in the special case where the ITS Factory has the minter role, and in this case it would mint the supply given as `initial_supply` argument.

Here are situations in which the ITS Factory would have the minter role although the initialization was complete:

- When the initialization of the token is finalized, the endpoint `deploy_interchain_token` forwards the minter role to the address given as `minter` argument, however this address could be that of the ITS Factory itself.
- If the minter is another address, it could always transfer the minter role back to the ITS Factory at any later time.

In particular, the initial supply of the token might be minted multiple times.

Expected behavior: The initial supply of a token should be minted at initialization when setting up the token. This is to ensure that the initial supply is minted before users start interacting with the token, and that it is minted at most once.

Recommendation

In the endpoint `deploy_interchain_token` , at the last step of the initialization of a token, we suggest recording that the token is initialized in a new `UnorderedSetMapper` `initialized_tokens` . Then, if `deploy_interchain_token` is called with a token already present in `initialized_tokens` , then the endpoint would fail.

Whether the last step is being executed in `deploy_interchain_token` can be detected as follows.

Case 1: If `initial_supply == 0` , the last step is the 2nd step (issuing the token), i.e. after we deployed the Token Manager.

Case 2: If `initial_supply > 0` , the last step is the 3rd step (minting the initial supply), i.e. after we issued the token.

```
require(!self.initialized_tokens.contains(token_id));

let opt_token_id =
  self.token_manager_invalid_token_identifier(token_manager);
let is_last_step =
  (initial_supply == 0 && !token_manager.is_zero) ||
  (initial_supply > 0) && opt_token_id.is_some();
if is_last_step {
  self.initialized_tokens.insert(token_id);
}
```

C3: ESDT tokens are accepted in "deploy_interchain_token" but are unused and can be lost

Severity: Medium

Status: Fixed

Location

```
interchain-token-factory/src/lib.rs  
    deploy_interchain_token
```

Description

Current behavior: The endpoint `deploy_interchain_token` is payable in any token, although it only uses EGLD for issuing the Token Manager's token.

Moreover, in case the user sends ESDT tokens to the endpoint, he could lose them. This is because:

- The EGLD payment is obtained via `self.call_value().egld_value()`, and there is no check on whether ESDT tokens were also received.
- The deployment procedure occurs in 2 or 3 steps, and each step is performed by re-calling the endpoint `deploy_interchain_token`. However, only in the 2nd step of the deployment procedure (issuing the token), it is checked that the EGLD amount is non-zero. This check incidentally ensures that no ESDT tokens were received.
- In the 1st and 3rd steps of the deployment procedure, there is a check that the EGLD amount is 0, but this does not prevent ESDT tokens from being received. In this case, ESDT tokens would be received and would simply stay in the smart contract.

Expected behavior: The endpoint `deploy_interchain_token` should only be payable in EGLD, because it only uses EGLD to issue the Token Manager's token, and does not need to receive any other token.

Worst consequence: Users send ESDT tokens to the endpoint `deploy_interchain_token` and lose them.

Recommendation

We recommend making the endpoint `deploy_interchain_token` payable only in EGLD, by using the annotation `#[payable("EGLD")]`.

C4: Token might be issued with name and ticker different from the ones asked by the user

Severity: Medium

Status: Fixed

Location

```
interchain-token-factory/src/lib.rs  
    deploy_interchain_token
```

Description

Current behavior: When a user issues a Native Interchain Token, in case the name or ticker is incorrectly formatted, the transaction does not revert. Instead, the name and ticker are normalized so that they comply with MultiversX requirements. Therefore, the token might be issued with a name and ticker that do not match the intent of the user.

This is because, in the endpoint `deploy_interchain_token`, no checks are made on the name and ticker provided as arguments, rather they are normalized by the methods `to_normalized_token_name` and `to_normalized_token_ticker`.

Expected behavior: When a user deploys a Native Interchain Token, the token should be issued with the exact name and ticker provided by the user.

Note: The normalization should remain for issuances coming from other blockchains, as then the name and ticker might be in an incorrect format, and the user on the source blockchain might have no control on it.

Recommendation

In the endpoint `deploy_interchain_token`, we recommend verifying that the arguments `symbol` and `name` correspond to valid ESDT ticker and name respectively. This can be done by verifying that they equal their normalization through the methods `to_normalized_token_name` and `to_normalized_token_ticker`.

C5: Remote deployment can be made with destination minter that was not asked by the user

Severity: Medium

Status: Fixed

Location

interchain-token-factory/src/lib.rs
 deploy_remote_interchain_token_with_minter

Description

Current behavior: In the endpoint

`deploy_remote_interchain_token_with_minter`, even if the optional argument `destination_minter` is not provided, i.e. there is no address provided to be set as a minter on the destination chain, then it is still possible that the interchain call is made with a destination minter.

This would happen if the argument `minter` is provided (i.e. a minter on the source chain, MultiversX), as then the destination minter is automatically set to this source minter:

```
if !minter.is_zero() {  
    ...  
} else {  
    destination_minter = minter;  
}
```

This means that the interchain call would be performed with a destination minter that was not asked by the user, and there are two possible outcomes:

- Either the remote deployment fails. This would happen on most blockchains (e.g. EVM blockchains) because the address format there would be incompatible with the MultiversX format, hence the address of the minter on MultiversX would fail to be decoded to a valid address on the destination blockchain. However, this might forever prevent deploying the token on that destination blockchain again, because the Axelar Hub currently prevents replaying token deployments, even if they did not succeed on the first trial.

- Or the remote deployment succeeds. In this case an unintended minter address would be set on that blockchain, and would forever be allowed to mint arbitrary amounts of tokens.

Note: A similar approach was taken in the Solidity code ([link](#)), because at the time of the implementation, only EVM blockchains integrated with Axelar, which share the same address format. In this context, it was useful to automatically use the same minter address on different blockchains, so that the same private key can be used as a minter over different blockchains.

Expected behavior: A remote deployment should be performed with a destination minter only if a destination minter was explicitly provided as argument to the endpoint `deploy_remote_interchain_token_with_minter`. Otherwise, the remote deployment should be performed without destination minter.

Recommendation

In the endpoint `deploy_remote_interchain_token_with_minter`, in case the optional argument `destination_minter` is not provided, we suggest performing the remote deployment without destination minter.

C6: EGLD can't be registered as a custom token

Severity: Medium

Status: Fixed

Location

```
interchain-token-factory/src/lib.rs  
    register_custom_token
```

Description

Current behavior: It is not possible to register EGLD as a custom token, because the endpoint `register_custom_token` verifies that the token identifier given as argument is a valid ESDT token identifier (using `is_valid_esdt_identifier`), and EGLD is not an ESDT token.

Conversely, when the link is made from another blockchain, it would fail to link to EGLD because in the method `process_link_token_payload`, it is also checked that the token identifier is a valid ESDT token identifier.

Expected behavior: According to the Axelar team, it should be possible to register EGLD as a custom token, as is already the case for all other fungible tokens. This is because linking custom tokens is a feature provided by Axelar for projects to be able to bridge existing tokens between different blockchains, which is relevant for all tokens on MultiversX including EGLD.

Recommendation

We recommend allowing registrations of EGLD as a custom token. For this:

- In the endpoint `register_custom_token`, we change the type of the argument `token_identifier` from `TokenIdentifier` to `EgldOrEsdtTokenIdentifier`, and perform the check that the token is a valid ESDT token identifier only if it is not EGLD.
- In the method `process_link_token_payload`, we decode the token identifier from the interchain call's payload as `EgldOrEsdtTokenIdentifier` instead of `TokenIdentifier`, and perform the check that the token is a valid ESDT token identifier only if it is not EGLD.

In addition, we update the ITS endpoint `register_token_metadata` so that it also supports EGLD:

- we change the type of the argument `token_identifier` from `TokenIdentifier` to `EgldOrEsdtTokenIdentifier`, and perform the check that the token is a valid ESDT token identifier only if it is not EGLD.
- If the token is EGLD, instead of fetching the token decimals from the ESDT System smart contract, we directly forward the EGLD decimals (i.e. `18`) in a message to the Axelar Hub.

Finally, we would add a test where EGLD is registered as a custom token and where an outgoing interchain call is registered to link EGLD to another blockchain. Conversely, we would add a test where an incoming interchain call links a token from another blockchain to EGLD.

C7: Unclear separation between deployment steps in "deploy_interchain_token"

Severity: Minor

Status: Fixed

Location

interchain-token-factory/src/lib.rs
deploy_interchain_token

Description

Current behavior: Deploying a new token is done in 2 or 3 steps, each time by calling the same endpoint `deploy_interchain_token`. However, the code of `deploy_interchain_token` makes it unclear that the 1st and 2nd steps are actually distinct. Namely, the code relies on Rust logical OR (`||`) evaluation behavior to handle the first 2 deployment steps together:

```
if token_manager.is_zero() ||
    token_manager_invalid_token_identifier(token_manager).is_none() {
    ...
}
```

Mixing these conditions makes it harder to understand the code. Moreover, it increases the risk of introducing errors in future changes: inverting the order of the 2 conditions would make token deployments fail, because calling `token_manager_invalid_token_identifier` would fail if `token_manager.is_zero()`, i.e. if there is no Token Manager deployed yet.

Expected behavior: The deployment steps in the endpoint `deploy_interchain_token` should be clearly separated, to make the code as simple as possible and to reduce the risk of introducing errors in future code changes.

Recommendation

In the endpoint `deploy_interchain_token`, we recommend explicitly separating the distinct deployment steps:

```
// 1st transaction - deploy Token Manager
if token_manager.is_zero() {
    ...
    return token_id;
}

// 2nd transaction - deploy token
if token_manager_invalid_token_identifier(token_manager).is_none() {
    ...
    return token_id;
}

// 3rd transaction - mint token if needed
...
```

C8: Function "ascii_to_u8" is unnecessarily complex

Severity: Minor

Status: Fixed

Location

interchain-token-factory/src/constants.rs
ascii_to_u8

Description

Current behavior: The function `ascii_to_u8` converts a string representing a token's decimals into an actual `u8` integer. However, the conversion is done in an unnecessarily complex way, i.e. by loading batches of 32 bytes:

```
fn ascii_to_u8(&self) -> u8 {  
    let mut result: u8 = 0;  
    self.for_each_batch::<32, _>(|batch| {  
        for &byte in batch {  
            if byte == 0 {  
                break;  
            }  
            result *= 10;  
            result += (byte as char).to_digit(16).unwrap() as u8;  
        }  
    });  
    result  
}
```

Namely, the decimals of a token on MultiversX must be between 0 and 18, hence the decimals' string representation is encoded on at most 2 bytes. In turn, it is unnecessary to parse the string by loading batches of 32 bytes as if the length of the string could be arbitrarily big.

Expected behavior: The function `ascii_to_u8` can be simplified and should be simplified.

Recommendation

We recommend simplifying the method `ascii_to_u8` as follows:

```
fn ascii_to_u8(&self) -> u8 {  
    let mut result: u8 = 0;  
    let mut byte_array = [0u8; 2];  
    let _ = self.load_slice(0, &mut byte_array);  
    for byte in byte_array {  
        result *= 10;  
        result += (byte as char).to_digit(16).unwrap() as u8;  
    }  
    result  
}
```

C9: Unnecessarily complex logic to extract ticker from token identifier

Severity: **Minor**

Status: **Fixed**

Location

interchain-token-factory/src/lib.rs
deploy_remote_interchain_token_raw

Description

In the method `deploy_remote_interchain_token_raw`, the ticker of the token is extracted by performing unnecessarily complex byte operations:

```
let token_identifier_name = token_identifier.into_name();  
let token_symbol = token_identifier_name  
    .copy_slice(0, token_identifier_name.len() - 7)  
    .unwrap();
```

Indeed, there is a Rust helper `ticker` that can be used directly to retrieve the ticker from an ESDT token.

Recommendation

In the method `deploy_remote_interchain_token_raw`, we recommend extracting the ticker of the ESDT token by using the dedicated helper from the Rust framework:

```
let token_symbol = token_identifier.unwrap_esdt().ticker();
```

C10: Inconsistent and misleading way to not specify address argument

Severity: Minor

Status: Fixed

Location

interchain-token-factory/src/lib.rs

Description

Current behavior: In several endpoints of the ITS Factory, an address can be optionally provided. However, if the user does not wish to specify an address, the way to do so is inconsistent over endpoints and sometimes misleading:

- In the endpoint `deploy_remote_interchain_token_with_minter`, the argument `destination_minter` is of type `OptionalValue<ManagedBuffer>`, therefore the user can provide a `None` argument.
- In the endpoint `register_custom_token`, the argument `operator` is mandatory, i.e. is of type `ManagedAddress`, and if the user does not wish to specify any operator, he would need to provide the zero address as argument. Similarly for the argument `minter` of the endpoint `deploy_interchain_token`.

In particular, the latter approach might be confusing since the user might not know that providing the zero address is the way to specify no address.

Expected behavior: The way of making an address argument optional should be consistent across endpoints, and should be as intuitive as possible.

Recommendation

We recommend changing the type of all address arguments which are optional to `OptionalValue<ManagedAddress>`:

- The argument `operator` of the endpoint `register_custom_token`,
- The argument `minter` of the endpoint `deploy_interchain_token`.

C11: No sanity check that a token is registered in Token Manager before registering on other blockchains

Severity: **Minor**

Status: **Fixed**

Location

```
interchain-token-factory/src/lib.rs  
    deploy_remote_interchain_token_raw
```

Description

Current behavior: The registration of a token on a destination blockchain can only be performed if the registration is complete on MultiversX, but this is for a subtle, non-explicit reason.

Namely, although it is explicitly checked that the Token Manager is already deployed, there is no explicit check that a token is set in the Token Manager. Fortunately, if the token is not set, then the remote registration fails, because in the method `deploy_remote_interchain_token_raw` :

- We read an empty `Egld0rEsdtTokenIdentifier` storage when calling `its_registered_token_identifier` ,
- Then, this token identifier is not recognized as EGLD when calling the method `is_egld` , since this helper from the Rust framework returns `true` only if the `data` field of the struct `Egld0rEsdtTokenIdentifier` is "EGLD-000000".
- The extraction of the ticker from an empty array of bytes fails:

```
let token_symbol = token_identifier_name  
    .copy_slice(0, token_identifier_name.len() - 7)  
    .unwrap();
```

However, this reasoning is subtle, and increases the risk of introducing errors in future changes:

- If the framework version is changed and in the new framework version, the helper `is_egld` returns `true` when the `data` field of the struct `Egld0rEsdtTokenIdentifier` is empty, then the registration to a destination blockchain wouldn't fail any longer, and would record the EGLD

information (name, symbol and decimals) instead of the correct information, on the destination blockchain.

- If the ticker is not extracted in the same way, maybe the registration to a destination blockchain would not fail any longer, but would be performed with erroneous information, such as an empty name or symbol.

Expected behavior: There should be an explicit check that the registration is finalized on MultiversX, in particular that a token is set in the Token Manager, before we can perform the registration on other blockchains. This is to make the code clearer, and reduce the risk of introducing errors in future changes.

Recommendation

In the method `deploy_remote_interchain_token_raw`, we recommend verifying that the local registration of the token is finalized, by explicitly checking that the token returned by the Token Manager view `registered_token_identifier` is a valid token, i.e. by calling the helper `is_valid` from the Rust framework.

C12: Unused functions "its_deployed_token_manager" and "token_manager_token_identifier"

Severity: Minor

Status: Fixed

Location

interchain-token-factory/src/proxy.rs

Description

The methods `its_deployed_token_manager` and `token_manager_token_identifier` are unused.

Recommendation

We recommend deleting the unused methods `its_deployed_token_manager` and `token_manager_token_identifier`.

