Axelar Network v2 contest
Findings & Analysis Report

2022-10-06

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Axelar Network v2 smart contract system written in Solidity. The audit contest took place between July 29—August 3 2022.

Wardens

80 Wardens contributed reports to the Axelar Network v2 contest:

1. Chom
2. xiaoming90
3. __141345__
4. Lambda
5. Ruhum
6. Respx
7. 0x52
8. cryptphi
9. oyc_109
10. IllIIII
11. rbserver
12. JC
13. Bnke0x0
14. defsec
15. Dravee
16. 0x1f8b
17. fatherOfBlocks

18. Deivitto
19. ajtra
20. robee
21. mics
22. Aymen0909
23. TomJ
24. horsefacts
25. Sm4rty
26. lucacez
27. c3phas
28. kyteg
29. Rolezn
30. simon135
31. 0xNazgul
32. benbaessler
33. RedOneN
34. Rohan16
35. Waze
36. apostle0x01
37. djxploit
38. gogo
39. tofunmi
40. NoamYakov
41. ReyAdmirado
42. bharg4v
43. asutorufos
44. bulej93
45. CodingNameKiki
46. durianSausage

47. sashik_eth
48. 8olidity
49. ElKu
50. Noah3o6
51. berndartmueller
52. hansfriese
53. cccz
54. CertoraInc (egjlmn1, OriDabush, ItayG, shakedwinder, and RoiEvenHaim)
55. sseefried
56. cryptonue
57. 0xf15ers (remora and twojoy)
58. 0xSmartContract
59. ashiq0x01
60. bardamu
61. codexploder
62. ignacio
63. Twpony
64. ch13fd357r0y3r
65. Yiko
66. MiloTruck
67. 0xsam
68. gerdusx
69. medikko
70. Tomio
71. owenthurm
72. a12jmx
73. Fitraldys
74. ak1
75. erictee

This contest was judged by Alex the Entreprenerd.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 6 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 6 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 65 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 56 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Axelar Network v2 contest repository, and is composed of 15 smart contracts written in the Solidity programming language and includes 1,813 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

Medium Risk Findings (6)

[M-01] `removeWrapping` can be called when there are still wrapped tokens

Submitted by Lambda, also found by 0x52 and cryptphi

XC20Wrapper.sol#L66

An owner can call `removeWrapping`, even if there are still circulating wrapped tokens. This will cause the unwrapping of those tokens to fail, as `unwrapped[wrappedToken]` will be `addres(0)`.

Recommended Mitigation Steps

Track how many wrapped tokens are in circulation, only allow the removal of a wrapped tokens when there are 0 to ensure for users that they will always be able to unwrap.

re1ro (Axelar) confirmed and commented:

> Valid observation. We will consider a different approach.

> **Mitigation**
> `removeWrapping` method was removed
> https://github.com/axelarnetwork/axelar-xc20-wrapper/pull/4

Alex the Entreprenerd (judge) commented:

> The warden has shown how the Admin can remove the mapping that allows to redeem bridged tokens, because this will cause the inability to unwrap, and can be operated by the admin, I agree with Medium Severity.

> The sponsor has confirmed and they have mitigated by removing the function.

[M-02] `XC20Wrapper` may lose received token forever if `LocalAsset(xc20).mint` is reverted indefinitely

Submitted by Chom

XC20Wrapper.sol#L124-L126

XC20Wrapper may lose received token forever if LocalAsset(xc20).mint is reverted indefinitely.

Similar to ERC20, the spec said that if mint returns false it means minting is failed. But it is commonly revert instead of returning false which is also a minting failure. XC20 may revert on minting as well and common sense also guiding programmers to use the revert pattern instead of returning false.

This case is not handled if SC20 minting is reverted indefinitely. No matter how hard you retry the GMP message execution, it always fail thus the token get locked forever.

Proof of Concept

```
        function _executeWithToken(
            string calldata,
            string calldata,
            bytes calldata payload,
            string calldata tokenSymbol,
            uint256 amount
        ) internal override {
            address receiver = abi.decode(payload, (address));
            address tokenAddress = gateway().tokenAddresses(tokenSym
            address xc20 = wrapped[tokenAddress];
            if (xc20 == address(0) || !LocalAsset(xc20).mint(receive
                _safeTransfer(tokenAddress, receiver, amount);
            }
        }
```

- Token is sent to gateway before executing the message on the destination chain.

- If `_executeWithToken` fail, the token remain inside gateway. The only way to use that token is to execute the `_executeWithToken` succesfully.
- Assume LocalAsset(xc20).mint(...) revert indefinitely, _executeWithToken also revert indefinitely.
- As a result, `_executeWithToken` never success thus the tokens remain inside gateway forever.

Recommended Mitigation Steps

Use try catch

```
    function _executeWithToken(
        string calldata,
        string calldata,
        bytes calldata payload,
        string calldata tokenSymbol,
        uint256 amount
    ) internal override {
        address receiver = abi.decode(payload, (address));
        address tokenAddress = gateway().tokenAddresses(tokenSym
        address xc20 = wrapped[tokenAddress];
        if (xc20 == address(0)) {
            _safeTransfer(tokenAddress, receiver, amount);
        }

        try LocalAsset(xc20).mint(receiver, amount) returns (boo
            if (!success) _safeTransfer(tokenAddress, receiver,
        } catch { _safeTransfer(tokenAddress, receiver, amount);
    }
```

[re1ro (Axelar) acknowledged and commented]():

> **Mitigation**
> We addressed the issue with introducing `_safeMint` function
> https://github.com/axelarnetwork/axelar-xc20-wrapper/pull/4

[Alex the Entreprenerd (judge) commented]():

> The warden states that `mint()` may fail and cause a revert instead of returning false.

> With the code in scope we can check the used ERC20 implementation and we find:

> [ERC20.sol#L187-L188]()

```
        if (account == address(0)) revert InvalidAccount();
```

> Because a revert can happen, the scenario, which hypothetically would brick the functionality can actually happen.

> We may also have reverts due to overflow and underflow.

> Because the code is built to assume that no revert can happen, but the warden demonstrated how a revert could factually happen, I do agree with Medium Severity.

> The sponsor has mitigated by using `_safeMint`.

[M-03] System will not work anymore after EIP-4758

Submitted by Lambda, also found by Chom

[DepositReceiver.sol#L25]()

After [EIP-4758](), the `SELFDESTRUCT` op code will no longer be available. According to the EIP, "The only use that breaks is where a contract is re-created at the same address using CREATE2 (after a SELFDESTRUCT)". Axelar is exactly such an application, the current deposit system will no longer work.

Recommended Mitigation Steps

To avoid that Axelar simply stops working one day, the architecture should be changed. Instead of generating addresses for every user, the user could directly interact with the deposit service and the deposit service would need to keep track of funds and provide refunds directly.

[re1ro (Axelar) commented]():

> Very good spot. We will address this.

re1ro (Axelar) acknowledged

Alex the Entreprenerd (judge) commented:

> The warden has shown a plausible upgrade path for Ethereum that will remove the `SELFDESTRUCT` opcode, bricking the `DepositReceiver` functionality.

> If the fork was in place today, the code would be broken, and the finding should be of high severity.

> Because the fork is not in place, and no clear timeline is defined for “The Purge”, I think Medium Severity to be correct.

[M-04] Previous {Operators/Weights/Threshold} Are Still Able To Sign Off New Commands After Operatorship Is Transferred

Submitted by xiaoming90

The administrator will call `AxelarAuthWeighted.transferOperatorship` function to transfer the operatorship to a new set of {Operators/Weights/Threshold}.

However, it was observed that after transferring the operatorship to a new set of {Operators/Weights/Threshold}, the previous sets of {Operators/Weights/Threshold} are still able to generate a valid proof, and subsequently execute the command.

The following piece of code shows that as long as valid proof is submitted, the commands will be executed by the system.

AxelarGateway.sol#L262

```
    function execute(bytes calldata input) external override {
        (bytes memory data, bytes memory proof) = abi.decode(input,

        bytes32 messageHash = ECDSA.toEthSignedMessageHash(keccak25
        // TEST auth and getaway separately
        bool currentOperators = IAxelarAuth(AUTH_MODULE).validatePro
            ..SNIP..
    }
```

The following piece of code shows that the past 16 sets of {Operators/Weights/Threshold} are considered valid and can be used within the `AxelarAuthWeighted._validateSignatures` function. Thus, the past 16 sets of {Operators/Weights/Threshold} are able to sign and submit a valid proof, and the proof will be accepted by the `AxelarAuthWeighted.validateProof` that allows them to execute the commands.

AxelarAuthWeighted.sol#L36

```
    uint8 internal constant OLD_KEY_RETENTION = 16;

    function validateProof(bytes32 messageHash, bytes calldata proof
            (address[] memory operators, uint256[] memory weights, u
                    proof,
                    (address[], uint256[], uint256, bytes[])
            );

            bytes32 operatorsHash = keccak256(abi.encode(operators,
            uint256 operatorsEpoch = epochForHash[operatorsHash];
            uint256 epoch = currentEpoch;

            if (operatorsEpoch == 0 || epoch - operatorsEpoch >= OLD

            _validateSignatures(messageHash, operators, weights, thr

            currentOperators = operatorsEpoch == epoch;
    }
```

Understood from the team that the reason for allowing past 16 sets of {Operators/Weights/Threshold} is that after a transfer of operatorship, the commands that were signed recently but have not been executed yet will not become invalid. Further understood from the team the operatorship transfer is performed when there is a significant change in stake distribution on the Axelar network.

It makes sense for commands that were already signed recently by past operators before the operatorship transfer to be executable. However, based on the current design, it is also possible for the past 16 sets of {Operators/Weights/Threshold} to submit a new valid proof/signature for new commands to be executed after the operatorship transfer, and the `AxelarAuthWeighted._validateSignatures` function will happily accept the proof/signature, which should not be allowed.

It was understood that the operatorship transfer is performed when there is a significant change in stake distribution on the Axelar network, therefore, it does not make sense for all the past 16 sets of {Operators/Weights/Threshold} to be still able to sign and execute new commands after the operatorship transfer, because they follow the old stake distribution that is no longer considered as valid.

Only the current set of operators and its stake distribution should be used to verify any new command signed and issued after the operatorship transfer.

Proof-of-Concept

Assuming that there are 3 validators (Alice, Bob and Charles)

Operation at Time 1 (T1)

At T1, the following is the state:

> currentEpoch = 1

> hashForEpoch[Epoch 1] = {operators: [Alice, Bob, Charles], weights: [0.5, 0.25, 0.25], threshold: 0.5} convert to hash

At T1, Alice could submit the following input to `AxelarGateway.execute(bytes calldata input)` function to execute the commands:

> input = {

> bytes memory data = commands to be executed

> bytes memory proof = {operators: [Alice, Bob, Charles], weights: [0.5, 0.25, 0.25], threshold: 0.5, signatures: [Alice’s signature]}

> }

Since Alice’s signature weight is 0.5, having Alice’s signature alone is sufficient to meet the threshold of 0.5 and the commands will be executed.

Operation at Time 2 (T2)

At T2, the Axelar administrator decided to change the stake distribution. The admin called the `AxelarAuthWeighted.transferOperatorship` and change the state to as follows:

> currentEpoch = 2

> hashForEpoch[Epoch 2] = {operators: [Alice, Bob, Charles], weights: [0.25, 0.4, 0.4], threshold: 0.5} convert to hash <== newly added

> hashForEpoch[Epoch 1] = {operators: [Alice, Bob, Charles], weights: [0.5, 0.25, 0.25], threshold: 0.5} convert to hash

At T2, Alice’s weight has reduced from `0.5` to `0.25`. As per the current stake distribution, Alice’s signature alone is not sufficient to meet the threshold of `0.5`. Thus, she is not able to execute any new command without an additional signature from Bob or Charles.

However, note that the past 16 sets of {operators/weights/threshold} are considered valid by the system, so in another word, all the past 16 stake distributions are considered valid too.

Thus, Alice simply needs to re-use back to the previous set of {operators/weights/threshold} in Epoch 1 and she can continue to execute new commands without the signature of Bob or Charles, thus bypassing the current stake distribution.

At T2, Alice could still submit the following input to `AxelarGateway.execute(bytes calldata input)` function with only Alice’s signature to execute the command:

> input = {

> bytes memory data = commands to be executed

> bytes memory proof = {operators: [Alice, Bob, Charles], weights: [0.5, 0.25, 0.25], threshold: 0.5, signatures: [Alice's signature]}

> }

No additional signature from Bob or Charles is needed.

Following is from Epoch 1

> {operators: [Alice, Bob, Charles], weights: [0.5, 0.25, 0.25], threshold: 0.5

Operator Address Changed After Operatorship Is Being Transferred

Noted from the discord channel the following clarification from the team.

> Based on couple of questions I have received, I'd like to clarify one assumption we are making for the contracts (which is enforced at the axelar proof of stake network): Operators correspond to validators on the Axelar network. However, the operator address for a given epoch is derived from the validator key along with a nonce that is unique for each operator epoch. i.e Whenever operatorship is being transferred, an honest validator will always generate a new operator address (and not reuse their old one) due to a nonce.

With this control in place, even if the validator has generated a new operator address after the operatorship has been transferred, it is still possible for the validator to re-use back the old operator address and sign the command as the validator is aware of the private key needed to sign on behalf of the old operator address. Thus, the above issue still exists.

Additionally, there is always a risk of a "dishonest" validator not generating a new operator address after operatorship is being transferred if the new stake distribution does not benefit them. In the above example, Alice who has its weightage reduced from 0.5 to 0.25 do not see the benefit of the new stake distribution can decide not to generate a new operator address and continue to use the old operator address that allowed her to sign and execute any command without an additional signature from Bob or Charles.

Impact

Current stake distribution can be bypassed.

Recommended Mitigation Steps

Consider updating the system to ensure that the following requirements are followed:

- Command signed by the past 16 sets of {operators/weights/threshold} AFTER the operatorship transfer should not be executable and should be rejected. Only commands signed by the current set of {operators/weights/threshold} AFTER the operatorship transfer should be accepted and executable.
- Commands signed by the past 16 sets of {operators/weights/threshold} BEFORE the operatorship transfer should be accepted and executable.

re1ro (Axelar) commented:

> Good spot.
> I think we could include the timestamps to prevent old operators to sing any new commands.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> Per the Warden POC - Stake Distribution from past epochs is not changed, meaning a `transferOperatorship` called by the `owner` with the goal of reducing weights for a specific operator will be circumventable.

> This implies:

- Ability to sidestep coded logic and code intent -> Broken Invariants
- Inability to kick a malicious operator (unless you use the 16 times transferOperatorship exploit shown from other reports)

> The "need to remove a bad operator" is definitely contingent on setup, so Medium Severity is definitely fair.

> I'll think about raising or keeping as Med.

Alex the Entreprenerd (judge) commented:

> In contrast to other reports, this submission shows a reasonable path forward to invalidate old operators, while allowing them to re-try old commands.

> For this reason I think this is distinct from #19 etc.

[M-05] Change of operators possible from old operators

Submitted by Lambda, also found by Respx and Ruhum

AxelarGateway.sol#L268
AxelarGateway.sol#L311

According to the specifications, only the current operators should be able to transfer operatorship. However, there is one way to circumvent this. Because currentOperators is not updated in the loop, when multiple `transferOperatorship` commands are submitted in the same `execute` call, all will succeed. After the first one, the operators that signed these commands are no longer the current operators, but the call will still succeed.

This also means that one set of operators could submit so many `transferOperatorship` commands in one `execute` call that `OLD_KEY_RETENTION` is reached for all other ones, meaning they would control complete set of currently valid operators.

Recommended Mitigation Steps

Set `currentOperators` to `false` when the operators were changed.

re1ro (Axelar) confirmed, but disagreed with severity and commented:

> This case would never occur in practice because our command batches are produced and signed by the Axelar nerwork. So there would be never 2 `transferOperatorship` commands in the same batch.

> In general if the recent operators turn malicious they can overtake the gateway disregarding this finding.

> **Mitigation**
> We still have added the sanity check to set `currentOperators` to `false`.
> https://github.com/axelarnetwork/axelar-cgp-solidity/pull/138

milapsheth (Axelar) commented:

> For more context, the current operators could just easily transfer the operatorship in multiple txs instead. We heavily rely on the assumption the majority of the operators by weight are not malicious.

Alex the Entreprenerd (judge) commented:

> Very interesting find.

> Per the warden submission: the operators can sign a `transferOperatorship` and then still act as if they are the current operator while their calls are being executed.

> This can be further extended to perform more than `OLD_KEY_RETENTION` to invalidate all old keys, which may be desirable or a malicious attack depending on context.

> The sponsor disagrees with severity, citing that the main assumption of the code is that operators by weight are non malicious

> Personally I think the finding:

- Breaks an assumption of the code (current operators exclusively can `transferOperatorship`)
- Allows the operators to kick old operators in one tx instead of `OLD_KEY_RETENTION` txs

Alex the Entreprenerd (judge) commented:

> With the information that I have, considering that:

- Breaks an assumption of the code (current operators exclusively can transferOperatorship)
- Allows the operators to kick old operators in one tx instead of OLD*KEY*RETENTION txs

> Because this is contingent on a malicious majority, and considering that a malicious majority can perform even worse attacks (DOS, TX Censoring, Shutting down the chain)

> I believe that Medium Severity is correct.

[M-06] Add cancel and refund option for Transaction Recovery

Submitted by __141345__

AxelarGateway.sol#L262
AxelarGasService.sol#L98
AxelarGasService.sol#L110

Transactions could fail or get stuck, according to the documentation:

> Occasionally, transactions can get “stuck” in the pipeline from a source to destination chain (e.g. due to one-off issues that arise with relayers that operate on top of the network).

> Transactions have typically gotten “stuck” in the pipeline due to: (A) The transaction failing to relay from the source chain into the Axelar network for processing. (B) The transaction failing to get executed on the destination chain.

And there are several options provided:

- manual approve
- manual execute
- add gas

However, some transactions’ execution depend on the time or certain condition. For example, some transaction has a deadline, it the deadline is passed, the transaction will be invalid. Or some conditions may be temporary, for example, some certain price difference for some token pair. In this case, the failed transactions will be meaningless to redo, the appropriate method is to cancel the transaction and refund. If no such option is provided, users’ fund for this transaction would be lock or loss.

Proof of Concept

```
contracts/AxelarGateway.sol
    function approveContractCall(bytes calldata params, bytes32
    function execute(bytes calldata input) external override {}

contracts/gas-service/AxelarGasService.sol
    function addGas() external override {}
    function addNativeGas() external payable override {}
```

The options are `approveContractCall()` , `execute` , `addGas()` and `addNativeGas()` are available, but no cancel and refund option.

Recommended Mitigation Steps

Provide a cancel option if the transaction failed, from the source chain or destination chain, and allow the user to get the gas refund.

re1ro (Axelar) acknowledged and commented:

> At this point this functionality can be implemented by the user in their Executable contract by the application. Axelar is providing a ground level cross-chain communication protocol.

> Refunds and deadline based cancel are very application specific cases and shouldn't be implemented on the protocol level. Some refunds could require manual intervention and it won't be scaleable for us to provide such support of all the applications built on top of Axelar. Especially considering that data related to expiration or price difference will be encoded inside of the payload and Axelar is not aware of the payload encoding.

> It shouldn't be too difficult to implement. In this example we will send it back to the original chain:

```
    function _executeWithToken(
        string memory sourceChain,
        string memory sourceAddress,
        bytes calldata payload,
        string memory tokenSymbol,
        uint256 amount
    ) internal override {
        if (price difference for some token pair > limit) {
            IERC20(token).approve(address(gateway), amount);
            gateway.sendToken(sourceChain, sourceAddress, tokenS
            return;
        }
       . . .
   }
```

> We will consider adding basic implementation of such methods to our `AxelarExecutable` so it can be adapted by the applications. We will have a better idea of the requirements when there will be more applications built on top. Good spot.

Alex the Entreprenerd (judge) commented:

> The warden has shown that the system in scope has no way to "cancel and refund" a transaction, while the details for impact are implementation dependent, allowing canceling tx that are failing to be relayed will help integrators in the worst case.

> While impact is hard to quantify because the expected value of the operation by the caller should be higher than the gas paid, the actual loss in the stated case is that of the gas cost of the transaction.

> While minor, given the fact that it is not recoverable, given the value of the submission and the acknowledgment by the sponsor, I think Medium Severity to be appropriate.

re1ro (Axelar) disagreed with severity and commented:

> We disagree with severity.
> Transactions are recoverable/refundable. There is nothing preventing it to be recovered from our protocol perspective. It's just that we don't suggest any default mechanism for this.

> Refund and cancel methods are up to the cross-chain app developer to implement. It very application specific and it's not up for us to decide how and

> what should be recovered/refunded. For some application execution deadline could be a trigger to refund, for others - price slippage. Even if transaction reverts it will restore the gateway approval and can be retried or refunded.

> Again it is not responsibility of the protocol but rather an application specific logic. We marked it as acknowledged because we agree we should provide some guidelines and examples for this in our docs. But there is no outstanding issue in this regard.

Please note: the following took place after judging and awarding were finalized.

Alex the Entreprenerd (judge) commented:

> I believe the Sponsor's counterargument to be valid and invite end users to make up their own opinion.

> Ultimately the presence or absence of an app-specific refund is dependent on the implementation.

> I chose to give Medium Severity in view of the risk for end-users, however, I could have rated with QA given a different context.

> I invite end-users to make up their own opinion and thank the sponsor for their insight.

Low Risk and Non-Critical Issues

For this contest, 65 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **oyc_109** received the top score from the judge.

The following wardens also submitted reports: rbserver, IllIllI, Bnke0x0, defsec, xiaoming90, horsefacts, JC, robee, 0x52, Dravee, mics, Chom, 0x1f8b, berndartmueller, fatherOfBlocks, Sm4rty, hansfriese, Aymen0909, c3phas, Deivitto, lucacez, __141345__, Rohan16, Waze, kyteg, Rolezn, cccz, CertoraInc, Lambda, simon135, Respx, 0xNazgul, ajtra, Ruhum, benbaessler, sseefried, bharg4v, cryptonue, RedOneN, 0xf15ers, 0xSmartContract, 8olidity, apostle0x01, ashiq0x01, bardamu, bulej93, codexploder, CodingNameKiki, cryptphi, djxploit, durianSausage, ElKu, gogo, ignacio, Noah3o6, sashik_eth, tofunmi, TomJ, Twpony, ch13fd357r0y3r, NoamYakov, ReyAdmirado, asutorufos, *and* Yiko.

[L-01] Unused receive() function

If the intention is for the Ether to be used, the function should call another function, otherwise it should revert

```
    AxelarDepositServiceProxy.sol::13 => receive() external payable
    DepositReceiver.sol::29 => receive() external payable {}
```

[L-02] decimals() not part of ERC20 standard

decimals() is not part of the official ERC20 standard and might fail for tokens that do not implement it. While in practice it is very unlikely, as usually most of the tokens implement it, this should still be considered as a potential issue.

```
    XC20Wrapper.sol::62 => if (!LocalAsset(xc20Token).set_metadata(r
```

[L-03] Unsafe use of transfer()/transferFrom() with IERC20

Some tokens do not implement the ERC20 standard properly but are still accepted by most code that accepts ERC20 tokens. For example Tether (USDT)'s transfer() and transferFrom() functions do not return booleans as the specification requires, and instead have no return value. When these sorts of tokens are cast to IERC20, their function signatures do not match and therefore the calls made, revert. Use OpenZeppelin's SafeERC20's safeTransfer()/safeTransferFrom() instead

```
    AxelarGasService.sol::128 => if (amount > 0) receiver.transfer(a
    AxelarGasService.sol::144 => receiver.transfer(amount);
    ReceiverImplementation.sol::23 => if (address(this).balance > 0)
    ReceiverImplementation.sol::51 => if (address(this).balance > 0)
    ReceiverImplementation.sol::71 => if (address(this).balance > 0)
    ReceiverImplementation.sol::86 => recipient.transfer(amount);
```

[L-04] Missing checks for zero address

Checking addresses against zero-address during initialization or during setting is a security best-practice. However, such checks are missing in address variable initializations/changes in many places.

Impact: Allowing zero-addresses will lead to contract reverts and force redeployments if there are no setters for such address variables.

```
229 https:
 19 https:
```

[N-01] Use a more recent version of solidity

Use a solidity version of at least 0.8.4 to get bytes.concat() instead of abi.encodePacked(,) Use a solidity version of at least 0.8.12 to get string.concat() instead of abi.encodePacked(,) Use a solidity version of at least 0.8.13 to get the ability to use using for with a list of free functions

```
AxelarAuthWeighted.sol::3 => pragma solidity 0.8.9;
AxelarDepositServiceProxy.sol::3 => pragma solidity 0.8.9;
AxelarDepositService.sol::3 => pragma solidity 0.8.9;
AxelarGasServiceProxy.sol::3 => pragma solidity 0.8.9;
AxelarGasService.sol::3 => pragma solidity 0.8.9;
AxelarGateway.sol::3 => pragma solidity 0.8.9;
DepositBase.sol::3 => pragma solidity 0.8.9;
DepositReceiver.sol::3 => pragma solidity 0.8.9;
IAxelarAuth.sol::3 => pragma solidity ^0.8.9;
IAxelarAuthWeighted.sol::3 => pragma solidity ^0.8.9;
IAxelarDepositService.sol::3 => pragma solidity ^0.8.9;
IAxelarGasService.sol::3 => pragma solidity ^0.8.9;
IDepositBase.sol::3 => pragma solidity ^0.8.9;
ReceiverImplementation.sol::3 => pragma solidity 0.8.9;
XC20Wrapper.sol::3 => pragma solidity 0.8.9;
```

[N-02] Unspecific Compiler Version Pragma

Avoid floating pragmas for non-library contracts.

While floating pragmas make sense for libraries to allow them to be included with multiple different versions of applications, it may be a security risk for application implementations.

A known vulnerable compiler version may accidentally be selected or security tools might fall-back to an older compiler version ending up checking a different EVM compilation that is ultimately deployed on the blockchain.

It is recommended to pin to a concrete compiler version.

```
IAxelarAuth.sol::3 => pragma solidity ^0.8.9;
IAxelarAuthWeighted.sol::3 => pragma solidity ^0.8.9;
IAxelarDepositService.sol::3 => pragma solidity ^0.8.9;
IAxelarGasService.sol::3 => pragma solidity ^0.8.9;
IDepositBase.sol::3 => pragma solidity ^0.8.9;
```

re1ro (Axelar) acknowledged and commented:

> [L-01]
> Not applicable. We need `receive` to receive ether from `WETH` contract.

> [L-02]
> Not applicable. `axelarToken` is our own implementation in this context and it implements `decimals`

> [L-03]
> Nope.

> [L-04]
> Yes.

> [N-01]
> We allow Unspecific Compiler version for our interfaces, so they can be imported by other projects

Alex the Entreprenerd (judge) commented:

[L-01] Unused receive() function
For the proxy
Low

[L-02] decimals() not part of ERC20 standard
Low

[L-03] Unsafe use of transfer()/transferFrom() with IERC20
Low

[L-04] Missing checks for zero address
Low

[N-01] Use a more recent version of solidity
Non-critical

[N-02] Unspecific Compiler Version Pragma
Non-critical

Gas Optimizations

For this contest, 56 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IllIllI received the top score from the judge.

The following wardens also submitted reports: JC, Dravee, 0x1f8b, ajtra, Bnke0x0, defsec, Deivitto, fatherOfBlocks, oyc_109, Aymen0909, MiloTruck, TomJ, Ruhum, __141345__, 0xNazgul, 0xsam, apostle0x01, benbaessler, djxploit, gerdusx, gogo, kyteg, Lambda, lucacez, medikko, NoamYakov, rbserver, RedOneN, ReyAdmirado, robee, Rolezn, simon135, tofunmi, Tomio, Respx, asutorufos, bharg4v, bulej93, Chom, CodingNameKiki, durianSausage, mics, owenthurm, Rohan16, sashik_eth, Sm4rty, Waze, a12jmx, Fitraldys, 8olidity, ak1, c3phas, ElKu, erictee, *and* Noah3o6.

Summary

	Issue	Instances
[G-01]	Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas	7
[G-02]	Avoid contract existence checks by using solidity version 0.8.10 or later	25
[G-03]	`internal` functions only called once can be inlined to save gas	7
[G-04]	`<array>.length` should not be looked up in every loop of a `for` -loop	7
[G-05]	`++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops	12
[G-06]	`keccak256()` should only need to be called on a specific string literal once	4
[G-07]	Optimize names to save gas	10
[G-08]	`++i` costs less gas than `i++` , especially when it's used in `for` -loops (`--i` / `i--` too)	5
[G-09]	Empty blocks should be removed or emit something	2
[G-10]	Functions guaranteed to revert when called by normal users can be marked `payable`	11

Total: 90 instances over 10 issues

[G-01] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. **Each iteration of this for-loop costs at least 60 gas** (i.e. `60 * <mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution. Note that even if an interface defines a function as having `memory` arguments, it's still valid for implementation contracs to use `calldata` arguments instead.

If the array is passed to an `internal` function which passes the array to another internal function where the array is modified and therefore `memory` is used in the `external` call, it's still more gass-efficient to use `calldata` when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one

Note that I've also flagged instances where the function is `public` but can be marked as `external` since it's not called by the contract, and cases where a constructor is involved

There are 7 instances of this issue:

```
File: contracts/auth/AxelarAuthWeighted.sol

/// @audit recentOperators
16:       constructor(bytes[] memory recentOperators) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/auth/AxelarAuthWeighted.sol#L16

```
File: contracts/AxelarGateway.sol

172:      function tokenFrozen(string memory) external pure over

/// @audit executeData
447       function _unpackLegacyCommands(bytes memory executeDat
448           external
449           pure
450           returns (
451               uint256 chainId,
452               bytes32[] memory commandIds,
453               string[] memory commands,
454:              bytes[] memory params
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/AxelarGateway.sol#L172

```
File: contracts/deposit-service/AxelarDepositService.sol

/// @audit wrappedSymbol
18:        constructor(address gateway, string memory wrappedSym
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/AxelarDepositService.sol#L18

```
File: contracts/deposit-service/DepositReceiver.sol

/// @audit delegateData
8:        constructor(bytes memory delegateData) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/DepositReceiver.sol#L8

```
File: contracts/deposit-service/ReceiverImplementation.sol

/// @audit wrappedSymbol
12:        constructor(address gateway, string memory wrappedSym
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/ReceiverImplementation.sol#L12

```
File: contracts/gas-service/AxelarGasService.sol

/// @audit symbol
35        function payGasForContractCallWithToken(
36            address sender,
37            string calldata destinationChain,
38            string calldata destinationAddress,
39            bytes calldata payload,
40            string memory symbol,
41            uint256 amount,
42            address gasToken,
43              uint256 gasFeeAmount,
44:             address refundAddress
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasService.sol#L35-L44

[G-02] Avoid contract existence checks by using solidity version 0.8.10 or later

Prior to 0.8.10 the compiler inserted extra code, including `EXTCODESIZE` (100 gas), to check for contract existence for external calls. In more recent solidity versions, the compiler will not insert these checks if the external call has a return value

There are 25 instances of this issue:

```
File: contracts/AxelarGateway.sol

/// @audit validateProof()
268:            bool currentOperators = IAxelarAuth(AUTH_MODULE).v

/// @audit _unpackLegacyCommands()
275:            try AxelarGateway(this)._unpackLegacyCommands(data

/// @audit call()
320:                (bool success, ) = address(this).call(abi.enco

/// @audit balanceOf()
385:                    abi.encodeWithSelector(IERC20.transfer.sel

/// @audit burn()
393:                IBurnableMintableCappedERC20(tokenAddress).bur

/// @audit mint()
481:                IBurnableMintableCappedERC20(tokenAddress).mir

/// @audit depositAddress()
525:                    IBurnableMintableCappedERC20(tokenAddress)

/// @audit burn()
532:            IBurnableMintableCappedERC20(tokenAddress).burn(by
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/AxelarGateway.sol#L268

```
File: contracts/deposit-service/AxelarDepositService.sol

/// @audit approve()
30:            IERC20(wrappedTokenAddress).approve(gateway, amour

/// @audit tokenAddresses()
115:               address gatewayToken = IAxelarGateway(gateway)
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/AxelarDepositService.sol#L30

```
File: contracts/deposit-service/DepositReceiver.sol

/// @audit delegatecall()
/// @audit receiverImplementation()
12:            (bool success, ) = IAxelarDepositService(msg.sende
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/DepositReceiver.sol#L12

```
File: contracts/deposit-service/ReceiverImplementation.sol

/// @audit tokenAddresses()
25:            address tokenAddress = IAxelarGateway(gateway).tok

/// @audit refundToken()
27:            address refund = DepositBase(msg.sender).refundTok

/// @audit balanceOf()
29:                _safeTransfer(refund, refundAddress, IERC20(re

/// @audit balanceOf()
33:            uint256 amount = IERC20(tokenAddress).balanceOf(ac
/// @audit approve()
38:            IERC20(tokenAddress).approve(gateway, amount);

/// @audit refundToken()
49:            address refund = DepositBase(msg.sender).refundTok

/// @audit balanceOf()
53:                _safeTransfer(refund, refundAddress, IERC20(re

/// @audit approve()
64:            IERC20(wrappedTokenAddress).approve(gateway, amour

/// @audit refundToken()
74:            address refund = DepositBase(msg.sender).refundTok

/// @audit balanceOf()
76:                _safeTransfer(refund, refundAddress, IERC20(re

/// @audit balanceOf()
80:            uint256 amount = IERC20(wrappedTokenAddress).balar

/// @audit withdraw()
85:            IWETH9(wrappedTokenAddress).withdraw(amount);
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/ReceiverImplementation.sol#L25

```
File: contracts/gas-service/AxelarGasService.sol

/// @audit balanceOf()
130:                    uint256 amount = IERC20(token).balanceOf(a
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasService.sol#L130

[G-03] `internal` functions only called once can be inlined to save gas

Not inlining costs **20 to 40 gas** because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There are 7 instances of this issue:

```
File: contracts/auth/AxelarAuthWeighted.sol

86          function _validateSignatures(
87              bytes32 messageHash,
88              address[] memory operators,
89              uint256[] memory weights,
90              uint256 threshold,
91:             bytes[] memory signatures

115:        function _isSortedAscAndContainsNoDuplicate(address[]
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/auth/AxelarAuthWeighted.sol#L86-L91

```
File: contracts/AxelarGateway.sol

611:        function _setTokenDailyMintAmount(string memory symbol

622:        function _setTokenAddress(string memory symbol, addres

630         function _setContractCallApproved(
631             bytes32 commandId,
632             string memory sourceChain,
633             string memory sourceAddress,
634             address contractAddress,
635:            bytes32 payloadHash

640         function _setContractCallApprovedWithMint(
641             bytes32 commandId,
642             string memory sourceChain,
643             string memory sourceAddress,
644             address contractAddress,
645             bytes32 payloadHash,
646             string memory symbol,
647:            uint256 amount
```

```
655:        function _setImplementation(address newImplementation)
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/AxelarGateway.sol#L611

[G-04] `<array>.length` should not be looked up in every loop of a `for` -loop

The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (**100 gas**)
- memory arrays use `MLOAD` (**3 gas**)
- calldata arrays use `CALLDATALOAD` (**3 gas**)

Caching the length changes each of these to a `DUP<N>` (**3 gas**), and gets rid of the extra `DUP<N>` needed to store the stack offset

There are 7 instances of this issue:

```
File: contracts/auth/AxelarAuthWeighted.sol

17:            for (uint256 i; i < recentOperators.length; ++i) {

98:            for (uint256 i = 0; i < signatures.length; ++i) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/auth/AxelarAuthWeighted.sol#L17

```
File: contracts/AxelarGateway.sol

207:            for (uint256 i = 0; i < symbols.length; i++) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/AxelarGateway.sol#L207

```
File: contracts/deposit-service/AxelarDepositService.sol

114:            for (uint256 i; i < refundTokens.length; i++) {

168:            for (uint256 i; i < refundTokens.length; i++) {

204:            for (uint256 i; i < refundTokens.length; i++) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/AxelarDepositService.sol#L114

```
File: contracts/gas-service/AxelarGasService.sol

123:            for (uint256 i; i < tokens.length; i++) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasService.sol#L123

[G-05] `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves **30-40 gas** per loop

There are 12 instances of this issue:

```
File: contracts/auth/AxelarAuthWeighted.sol
```

```
17:            for (uint256 i; i < recentOperators.length; ++i) {

69:            for (uint256 i = 0; i < weightsLength; ++i) {

98:            for (uint256 i = 0; i < signatures.length; ++i) {

101:                for (; operatorIndex < operatorsLength && sigr

116:            for (uint256 i; i < accounts.length - 1; ++i) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/auth/AxelarAuthWeighted.sol#L17

```
File: contracts/AxelarGateway.sol

195:            for (uint256 i; i < adminCount; ++i) {

207:            for (uint256 i = 0; i < symbols.length; i++) {

292:            for (uint256 i; i < commandsLength; ++i) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/AxelarGateway.sol#L195

```
File: contracts/deposit-service/AxelarDepositService.sol

114:            for (uint256 i; i < refundTokens.length; i++) {

168:            for (uint256 i; i < refundTokens.length; i++) {

204:            for (uint256 i; i < refundTokens.length; i++) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/AxelarDepositService.sol#L114

```
File: contracts/gas-service/AxelarGasService.sol

123:          for (uint256 i; i < tokens.length; i++) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasService.sol#L123

[G-06] `keccak256()` should only need to be called on a specific string literal once

It should be saved to an immutable variable, and the variable used instead. If the hash is being used as a part of a function selector, the cast to `bytes4` should also only be done once

There are 4 instances of this issue:

```
File: contracts/deposit-service/AxelarDepositServiceProxy.sol

9:          return keccak256('axelar-deposit-service');
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/AxelarDepositServiceProxy.sol#L9

```
File: contracts/deposit-service/AxelarDepositService.sol

242:          return keccak256('axelar-deposit-service');
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/AxelarDepositService.sol#L242

```
File: contracts/gas-service/AxelarGasServiceProxy.sol
```

```
10:            return keccak256('axelar-gas-service');
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasServiceProxy.sol#L10

```
File: contracts/gas-service/AxelarGasService.sol

181:           return keccak256('axelar-gas-service');
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasService.sol#L181

[G-07] Optimize names to save gas

`public`/`external` function names and `public` member variable names can be optimized to save gas. See this link for an example of how it works. Below are the interfaces/abstract contracts that can be optimized so that the most frequently-called functions use the least amount of gas possible during method lookup. Method IDs that have two leading zero bytes can save **128 gas** each during deployment, and renaming functions to have lower method IDs will save **22 gas** per call, per sorted position shifted

There are 10 instances of this issue:

```
File: contracts/auth/AxelarAuthWeighted.sol

/// @audit validateProof(), transferOperatorship()
9:    contract AxelarAuthWeighted is Ownable, IAxelarAuthWeighte
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/auth/AxelarAuthWeighted.sol#L9

```
File: contracts/AxelarGateway.sol
```

```
/// @audit sendToken(), callContract(), callContractWithToken(),
15:   contract AxelarGateway is IAxelarGateway, AdminMultisigBas
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/AxelarGateway.sol#L15

```
File: contracts/deposit-service/AxelarDepositService.sol

/// @audit sendNative(), addressForTokenDeposit(), addressForNat
15:   contract AxelarDepositService is Upgradable, DepositBase,
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/AxelarDepositService.sol#L15

```
File: contracts/deposit-service/ReceiverImplementation.sol

/// @audit receiveAndSendToken(), receiveAndSendNative(), recei
11:   contract ReceiverImplementation is DepositBase {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/ReceiverImplementation.sol#L11

```
File: contracts/gas-service/AxelarGasService.sol

/// @audit collectFees(), refund(), contractId()
10:   contract AxelarGasService is Upgradable, IAxelarGasService
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasService.sol#L10

```
File: contracts/interfaces/IAxelarAuth.sol
```

```
/// @audit validateProof(), transferOperatorship()
7:    interface IAxelarAuth is IOwnable {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/interfaces/IAxelarAuth.sol#L7

```
File: contracts/interfaces/IAxelarAuthWeighted.sol

/// @audit currentEpoch(), hashForEpoch(), epochForHash()
7:    interface IAxelarAuthWeighted is IAxelarAuth {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/interfaces/IAxelarAuthWeighted.sol#L7

```
File: contracts/interfaces/IAxelarDepositService.sol

/// @audit sendNative(), addressForTokenDeposit(), addressForNat
9:    interface IAxelarDepositService is IUpgradable, IDepositBa
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/interfaces/IAxelarDepositService.sol#L9

```
File: contracts/interfaces/IAxelarExecutable.sol

/// @audit execute(), executeWithToken()
7:    abstract contract IAxelarExecutable {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/interfaces/IAxelarExecutable.sol#L7

```
File: contracts/interfaces/IAxelarGasService.sol
```

```
/// @audit payGasForContractCall(), payGasForContractCallWithTok
8:    interface IAxelarGasService is IUpgradable {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/interfaces/IAxelarGasService.sol#L8

[G-08] `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

Saves **5 gas per loop**

There are 5 instances of this issue:

```
File: contracts/AxelarGateway.sol

207:          for (uint256 i = 0; i < symbols.length; i++) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/AxelarGateway.sol#L207

```
File: contracts/deposit-service/AxelarDepositService.sol

114:          for (uint256 i; i < refundTokens.length; i++) {

168:          for (uint256 i; i < refundTokens.length; i++) {

204:          for (uint256 i; i < refundTokens.length; i++) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/deposit-service/AxelarDepositService.sol#L114

```
File: contracts/gas-service/AxelarGasService.sol
```

```
123:            for (uint256 i; i < tokens.length; i++) {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasService.sol#L123

[G-09] Empty blocks should be removed or emit something

The code should be refactored such that they no longer exist, or the block should do something useful, such as emitting an event or reverting. If the contract is meant to be extended, the contract should be `abstract` and the function signatures be added without any default implementation. If the block is an empty `if`-statement block to avoid doing subsequent checks in the else-if/else conditions, the else-if/else conditions should be nested under the negation of the if-statement, because they involve different classes of checks, which may lead to the introduction of errors when the code is later modified (`if(x){}else if(y){...}else{...}` => `if(!x){if(y){...}else{...}}`). Empty `receive()` / `fallback()` `payable` functions that are not used, can be removed to save deployment gas.

There are 2 instances of this issue:

```
File: contracts/interfaces/IAxelarExecutable.sol

46:        ) internal virtual {}

54:        ) internal virtual {}
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/interfaces/IAxelarExecutable.sol#L46

[G-10] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about **21 gas per call** to the function, in addition to the extra deployment cost

There are 11 instances of this issue:

```
File: contracts/auth/AxelarAuthWeighted.sol

47:        function transferOperatorship(bytes calldata params) e
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/auth/AxelarAuthWeighted.sol#L47

```
File: contracts/AxelarGateway.sol

204:       function setTokenDailyMintLimits(string[] calldata sym

217        function upgrade(
218            address newImplementation,
219            bytes32 newImplementationCodeHash,
220            bytes calldata setupParams
221:       ) external override onlyAdmin {

331:       function deployToken(bytes calldata params, bytes32) e

367:       function mintToken(bytes calldata params, bytes32) ext

373:       function burnToken(bytes calldata params, bytes32) ext

397:       function approveContractCall(bytes calldata params, by

411:       function approveContractCallWithMint(bytes calldata pa

437:       function transferOperatorship(bytes calldata newOperat
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/AxelarG ateway.sol#L204

```
File: contracts/gas-service/AxelarGasService.sol

120:      function collectFees(address payable receiver, address

136       function refund(
137           address payable receiver,
138           address token,
139           uint256 amount
140:      ) external onlyOwner {
```

https://github.com/code-423n4/2022-07-axelar/blob/9c4c44b94cddbd48b9baae30051a4e13cbe39539/contracts/gas-service/AxelarGasService.sol#L120

Alex the Entreprenerd (judge) commented:

> **[G-01] Using calldata instead of memory for read-only arguments in external functions saves gas**
> 60 for the array of bytes

> **[G-02] Avoid contract existence checks by using solidity version 0.8.10 or later**
> 100 gas per instance
> 2500

> **[G-03] internal functions only called once can be inlined to save gas**
> 20 per instance
> 140

> **[G-04] .length should not be looked up in every loop of a for-loop + [G-05]**
> Giving 300 consistently with rest of submissions

> **[G-06] keccak256() should only need to be called on a specific string literal once**
> 30 gas per instance
> 120

> Rest is too opinionated for me :P

> Great report as usual, would love to see a couple customized suggestion (packing or similar) and benchmarks, but still really good.

> 3120 gas saved

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top