Axelar Network contest Findings & Analysis Report

2022-06-29

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 smart contract system written in Solidity. The audit contest took place between April 7—April 11 2022.

Wardens

24 Wardens contributed reports to the Axelar Network contest:

1. CertoraInc (danb, egjlmn1, OriDabush, ItayG, and shakedwinder)
2. Chom
3. sseefried
4. rayn
5. cccz
6. IIIIIII
7. ilan
8. Dravee
9. Funen
10. dirk_y
11. delfin454000
12. rishabh
13. 0xkatana
14. 0xNazgul
15. rfa
16. Tomio
17. 0v3rf10w
18. Hawkeye (0xwags and 0xmint)
19. nahnah

This contest was judged by 0xean.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 5 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 4 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 10 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 14 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 contest repository, and is composed of 19 smart contracts written in the Solidity programming language and includes 1,605 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.

High Risk Findings (1)

[H-01] Cross-chain smart contract calls can revert but source chain tokens remain burnt and are not refunded

Submitted by sseefried, also found by Chom

Smart contract calls often revert. In such cases any ether sent along with the transaction is returned and sometimes the remaining gas (depending on whether an `assert` caused the reversion or not).

For contracts involving ERC20 tokens it is also expected that, should a contract call fail, one's tokens are not lost/transferred elsewhere.

The callContractWithToken function does not appear to take contract call failure on the destination chain into account, even though this could be quite a common occurrence.

Tokens are burned on line 105 but there is no mechanism in the code base to return these burned tokens in the case that the contract call fails on the destination chain.

The impact is that users of the Axelar Network can lose funds.

Proof of Concept

I have put together an executable Proof of Concept in a fork of the repo. File DestinationChainContractCallFails.js implements a test that attempts to call a token swap function on the destination chain. The swap function was provided as part of the competition repo. Given a certain amount of token A it returns twice as much of token B.

In the test I have provided the contract call on the destination chain fails because there is simply not enough of token B in the `TokenSwapper` contract to transfer to the user. This might be rare in practice — since adequate liquidity would generally be provided by the contract — but cross-chain contract calls are unlikely to be limited to token swaps only! I specifically chose this example to show that cross-chain contract calls can fail *even in the cases that Axelar have already considered* in their test suite.

In the **unit test** you will find:

- Lines of note have been prefixed with `sseefried:`
- The test is a little strange in that it *succeeds* because it expects a `revert`. This happens on **line 380**
- I took the liberty of modifying the `TokenSwapper` contract slightly **here**, in order to show that the contract call reverts because of a lack of token B.
- The amount of token A on **line 201** can be modified to be a smaller value. Doing so, and re-running the test, will result in a *test failure* which means that the contract call did *not* revert i.e. the contract call on the destination chain succeeded. This shows that, before the change, the revert was due to a lack of token B in the `TokenSwapper` contract.
- **Lines 388-389** show that, in the case of a revert on the destination chain, the tokens remain burnt on the source chain.

Recommended Mitigation Steps

When making a credit card purchase it is common for transactions to remain in a "pending" state until eventually finalised. Often one's *available* bank balance will decrease the moment the purchase has been approved. Then one of two things happens:

- the transaction is finalised and the balance becomes the same as the available balance
- the transaction fails and the amount is refunded

I suggest a similar design for cross-chain contract calls, with one major difference: the token should still be burned on the source chain but it should be re-minted and refunded in case of a contract call failure on the destination chain. The steps would be roughly this:

- User calls `AxelarGateway.callContractWithToken()` and tokens are burned
- Steps 3 - 8 from the **competition page** occur as normal.
- However, the call to `executeWithToken` in step 8 now fails. This is monitored by the Axelar Network and a new event e.g. `ContractCalledFailed` is emitted on the *destination chain*.

- One the *source chain* the Axelar Network emits a new event e.g. `ContractCallFailedWithRefund`. This causes a re-minting of the tokens and a refund to the user to occur. The event should also be observable by the user. It should contain a reason for the contract call failure so that they are informed as to why it failed

deluca-mike (Axelar) acknowledged and commented:

> In this situation, the validators can still mint/transfer the user back their tokens on the source chain, so there is no real loss. There does lack an "official" way to alert the validators of this, but it can be handled entirely by off-chain micro-services and whatnot. In the future, as ERC20 transfers are pushed out of the contract as handled as separate application on top of the generic cross-chain calls, a mechanism can be implemented to send a message back to the source chain to release/mint the tokens back to the user.

0xean (judge) increased severity to High and commented:

> Upgrading this issue from Medium to High Severity

> ```
> 3 — High: Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).
> ```

> While the sponsor does mention there is a possible way that this could be handled in the future, currently this risk exists in the system today and doesn't have a proper or official mitigation in place.

Medium Risk Findings (4)

[M-01] Low level call returns true if the address doesn't exist

Submitted by CertoraInc

AxelarGateway.sol#L545-L548
AxelarGatewayProxy.sol#L16-L24

As written in the [solidity documentation](), the low-level functions call, delegatecall and staticcall return true as their first return value if the account called is non-existent, as part of the design of the EVM. Account existence must be checked prior to calling if needed.

Proof of Concept

The low-level functions `call` and `delegatecall` are used in some places in the code and it can be problematic. For example, in the `_callERC20Token` of the `AxelarGateway` contract there is a low level call in order to call the ERC20 functions, but if the given `tokenAddress` doesn't exist `success` will be equal to true and the function will return true and the code execution will be continued like the call was successful.

```
function _callERC20Token(address tokenAddress, bytes memory call
    (bool success, bytes memory returnData) = tokenAddress.call(
    return success && (returnData.length == uint256(0) || abi.de
}
```

Another place that this can happen is in `AxelarGatewayProxy`'s constructor

```
constructor(address gatewayImplementation, bytes memory params)
    _setAddress(KEY_IMPLEMENTATION, gatewayImplementation);

   (bool success, ) = gatewayImplementation.delegatecall(
       abi.encodeWithSelector(IAxelarGateway.setup.selector, par
   );

    if (!success) revert SetupFailed();
}
```

If the `gatewayImplementation` address doesn't exist, the delegate call will return true and the function won't revert.

Tools Used

Remix, VS Code

Recommended Mitigation Steps

Check before any low-level call that the address actually exists, for example before the low level call in the callERC20 function you can check that the address is a contract by checking its code size.

deluca-mike (Axelar) confirmed, but disagreed with severity and commented:

> `_callERC20Token` is only called with a token address that has been set and was already validated by the gateway, and thus they do already exist.

> See:
>
> - AxelarGateway.sol#L291
> - AxelarGateway.sol#L379

> As for `gatewayImplementation.delegatecall`, this is a valid find, but there isn't much risk because it would just mean that the gateway deployed would be a dud, and it would need to be redeployed correctly before the Axelar network can start interacting with it. In any case, we will fix this by checking that there is code at that address.

0xean (judge) commented:

> `2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

> Given that this *would* affect the protocols availability (the gateway being a "dud") the finding does qualify as med risk.

[M-02] User's funds can get lost when transferring to other chain

Submitted by CertoraInc

AxelarGateway.sol#L384-L389

When transferring tokens to other chain, the tokens in the source chain are burned - if they are external they will be transferred to the AxelarGateway, otherwise they will be burned. In the target chain the same amount of tokens will be minted for the user - if it is external it will be transferred to him from the AxelarGateway, otherwise it will be minted to him.
But there is a problem - if the AxelarGateway doesn't have the needed amount of token for some reason, the `_callERC20Token` with the `transfer` function selector will fail and return false, which will make the `_mintToken` function revert. Because it reverted, the user won't get his funds on the destination chain, although he payed the needed amount in the source chain.

Tools Used

VS Code and Remix

Recommended Mitigation Steps

Instead of reverting when the transfer is not successful, simply call the `callContractWithToken` with the source chain as the destination chain in order to return the user his funds.

deluca-mike (Axelar) acknowledged and commented:

> While true, the only way the destination gateway cannot mint or transfer token, in response to a burn or transferFrom on the source chain, is foul play on the validators' part. Once we assume foul play, then really there is no protection here, since even with the recommended mitigation steps, you'd still need cooperation from the validators to sign the mint/transfer refund command for the source gateway.

> Because of this, while the issue is acknowledged, it's not really something that can be solved.

> Further, well-behaving validators can still sign a refund mint/transfer for the source gateway if they see that a mint/transfer on the destination gateway failed, without needing to do what is suggested in the Recommended Mitigation Steps.

[M-03] `_execute` can potentially reorder a batch of commands while executing, breaking any assumptions on command orders.

Submitted by rayn

AxelarGatewayMultisig.sol#L484
AxelarGatewayMultisig.sol#L490
AxelarGatewayMultisig.sol#L529

Since this is important, we quote it again instead of referring to our other bug report on a different, yet related bug. The context within which a command is executed is extremely important.

> AxelarGatewayMultisig.execute() takes a signed batch of commands. Each command has a corresponding commandID. This is guaranteed to be unique from the Axelar network. execute intentionally allows retrying a commandID if the command failed to be processed; this is because commands are state dependent, and someone might submit command 2 before command 1 causing it to fail.

Thus if an attacker manages to rearrange execution order of commands within a batch, it should probably be treated seriously. This is exactly what might happen here due to reentrancy. A malicious player that managed to gain reentrancy over execute can easily execute later commands in a batch before earlier commands are fully executed, effectively breaking all assumptions on command executed context.

Proof of Concept

The `_execute` function and its wrapper `execute` are both reentrant.

```
function execute(bytes calldata input) external override;
function _execute(bytes memory data, bytes[] memory signatur
```

Thus if an attacker manages to reenter the `_execute` function with the same batch of commands and signatures, previously successfully executed and ongoing commands will be skipped due to premature marking of the success flag.

```
        if (isCommandExecuted(commandId)) continue; /* Ignore if
```

This allows later commands to be executed before the current ongoing command is finished. The reentrant attack can be nested to perform further reordering of commands.

Generally speaking, other unrelated batches of signed commands can only be executed, but since the assumption of ordering is most likely stronger within a single batch, we focus on illustrating the single batch scenario above.

Tools Used

vim, ganache-cli

Recommended Mitigation Steps

Make execute nonReentrant

Add an ever increasing nonce to signatures to prevent replay

```
function execute(bytes calldata input) nonReentrant external
    ...
}
```

deluca-mike (Axelar) acknowledged, but disagreed with High severity and commented:

> Axelar and the gateways make no concrete guarantees about command execution order, so the idea of "reordering" is moot. Currently, the only commands that call out are:
>
> - deploy a token (so its order is irrelevant)
> - minting a token (and thus a call to the token contract will not re-enter, since it is either one the gateway deployed, or the gateway onboarded a malicious external ERC2, which still has no effect on the other tokens)
> - burn a token (same as above)

> As with the other issue, while we do acknowledge that the contestant has correctly pointed out how the gateway handles (and can re-handle) commands, this is a low-risk (or no-risk) issue, since no actual risk has been demonstrated apart from “maybe, somehow, possibly, it could be abused”. We’d need at least some feasible and concrete hypothetical or example.

Oxean (judge) decreased severity to Medium and commented:

> I am going to side with the warden on this but downgrade to medium severity. The transaction ordering is not the high risk issue, but the ability for re-entrancy and thus replay presents a risk that should be mitigated.

> Looking at other recent hacks (for example - https://rekt.news/agave-hundred-rekt/), new environments pose new risks for re-entrancy to appear and with the point of this protocol being to extend across many chains, added additional measures to avoid re-entrancy and the replay that could occur as a result seems like a very logical choice.
>
> ```
> 2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.
> ```

[M-04] Unsupported fee-on-transfer tokens

Submitted by cccz

When tokenAddress is fee-on-transfer tokens, in the _burnTokenFrom function, the actual amount of tokens received by the contract will be less than the amount.

Proof of Concept

AxelarGateway.sol#L284-L334

Recommended Mitigation Steps

Consider getting the received amount by calculating the difference of token balance (using balanceOf) before and after the transferFrom.

deluca-mike (Axelar) confirmed and commented:

> Valid for `TokenType.External`, since it is a token implementation that is not ours, and thus could actually transfer us less than expected due to fees.

> Keep in mind that, in the case of a malicious token contract, it could also lie about the `balanceOf`.

> In any case, if and when we wanted to accept fee-on-transfer tokens in the gateway, we *might* need to implement the recommended mitigation steps; however, it is not that simple because the is not link (on-chain) here that ensure the amount the gateway burns to be equal to the amount the gateway/validators mint elsewhere. Knowing the actual amount burned is not critical to the source gateway, but rather to the validators that will need to create the mint command elsewhere.

Low Risk and Non-Critical Issues

For this contest, 10 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **IIIIIII** received the top score from the judge.

The following wardens also submitted reports: **rayn**, **ilan**, **CertoraInc**, **dirk_y**, **delfin454000**, **cccz**, **Funen**, **Dravee**, *and* **rishabh**.

[L-01] Cross-chain replay attacks

Storing the `block.chainid` is not safe. See **this** issue from a prior contest for details.

1. File: src/ERC20Permit.sol (lines **23-28**)

```
            DOMAIN_SEPARATOR = keccak256(
                abi.encode(
                    DOMAIN_TYPE_SIGNATURE_HASH,
                    keccak256(bytes(name)),
                    keccak256(bytes('1')),
                    block.chainid,
```

[L-02] Incorrect EIP-2612 deadline check

> allow `transferFrom` to occur while `expiry >= block.timestamp`.

https://github.com/ethereum/EIPs/blob/1473025f064929bfab405eb00b8cd16dd741f269/EIPS/eip-2612.md?plain=1#L172
The current code should change to use `<=`

1. File: src/ERC20Permit.sol (line 43)

```
        require(block.timestamp < deadline, 'EXPIRED');
```

[L-03] Missing checks for `address(0x0)` when assigning values to `address` state variables

1. File: src/AxelarGateway.sol (line 67)

```
        TOKEN_DEPLOYER_IMPLEMENTATION = tokenDeployerImplementat
```

[L-04] Open TODOs

Code architecture, incentives, and error handling/reporting questions/issues should be resolved before deployment

1. File: src/MintableCappedERC20.sol (line 30)

```
        // TODO move burnFrom into a separate BurnableERC20 contract
```

[L-05] Missing contract-existence checks before low-level calls

Low-level calls return success if there is no code present at the specified address. In addition to the zero-address checks, add a check to verify that
`<address>.code.length > 0`

1. File: src/AxelarGateway.sol (lines 398-415)

```
if (tokenAddress == address(0)) revert TokenDoesNotExist

if (_getTokenType(symbol) == TokenType.External) {
    _checkTokenStatus(symbol);

    DepositHandler depositHandler = new DepositHandler{

    (bool success, bytes memory returnData) = depositHar
        tokenAddress,
        abi.encodeWithSelector(
            IERC20.transfer.selector,
            address(this),
            IERC20(tokenAddress).balanceOf(address(depos
        )
    );

    if (!success || (returnData.length != uint256(0) &&
        revert BurnFailed(symbol);
```

2. File: src/AxelarGatewayProxy.sol (line 19)

```
(bool success, ) = gatewayImplementation.delegatecall(
```

3. File: src/AxelarGatewayProxy.sol (line 34)

```
let result := delegatecall(gas(), implementation, 0,
```

4. File: src/AxelarGateway.sol (line 350)

```
(bool success, bytes memory data) = TOKEN_DEPLOYER_I
```

[N-01] `require()` / `revert()` statements should have descriptive reason strings

1. File: src/DepositHandler.sol (line 12)

```
        require(_lockedStatus == IS_NOT_LOCKED);
```

[N-02] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

1. File: src/BurnableMintableCappedERC20.sol (line 48)

```
    function burn(bytes32 salt) public onlyOwner {
```

2. File: src/MintableCappedERC20.sol (line 23)

```
    function mint(address account, uint256 amount) public onlyOw
```

[N-03] `constant` s should be defined rather than using magic numbers

1. File: src/BurnableMintableCappedERC20.sol (line 37)

```
                                bytes1(0xff),
```

2. File: src/ECDSA.sol (line 33)

```
        if (signature.length != 65) revert InvalidSignatureLengt
```

3. File: src/ECDSA.sol (line 58)

```
if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576
```

4. File: src/ECDSA.sol (line 60)

```
if (v != 27 && v != 28) revert InvalidV();
```

5. File: src/ECDSA.sol (line 60)

```
if (v != 27 && v != 28) revert InvalidV();
```

6. File: src/ERC20Permit.sol (line 44)

```
require(uint256(s) <= 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

7. File: src/ERC20Permit.sol (line 45)

```
require(v == 27 || v == 28, 'INV_V');
```

8. File: src/ERC20Permit.sol (line 45)

```
require(v == 27 || v == 28, 'INV_V');
```

[N-04] Use a more recent version of solidity

Use a solidity version of at least 0.8.12 to get `string.concat()` to be used instead of `abi.encodePacked(<str>,<str>)`

1. File: src/AxelarGateway.sol (line 3)

```
pragma solidity 0.8.9;
```

2. File: src/AdminMultisigBase.sol (line 3)

```
pragma solidity 0.8.9;
```

3. File: src/AxelarGatewayMultisig.sol (line 3)

```
pragma solidity 0.8.9;
```

4. File: src/BurnableMintableCappedERC20.sol (line 3)

```
pragma solidity 0.8.9;
```

5. File: src/ECDSA.sol (line 3)

```
pragma solidity 0.8.9;
```

6. File: src/ERC20Permit.sol (line 3)

```
pragma solidity 0.8.9;
```

[N-05] Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate

1. File: src/ERC20.sol (lines 34-36)

```
    mapping(address => uint256) public override balanceOf;

    mapping(address => mapping(address => uint256)) public overr
```

[N-06] Variable names that consist of all capital letters should be reserved for `const` / `immutable` variables

If the variable needs to be different based on which class it comes from, a `view` / `pure` *function* should be used instead (e.g. like this).

1. File: src/ERC20Permit.sol (line 8)

```
    bytes32 public DOMAIN_SEPARATOR;
```

[N-07] File is missing NatSpec

1. File: src/AdminMultisigBase.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

2. File: src/DepositHandler.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

3. File: src/BurnableMintableCappedERC20.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

4. File: src/Ownable.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

5. File: src/TokenDeployer.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

6. File: src/interfaces/IAxelarGateway.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

7. File: src/interfaces/IAxelarGatewayMultisig.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

8. File: src/interfaces/IERC20BurnFrom.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

9. File: src/interfaces/IAxelarExecutable.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

10. File: src/MintableCappedERC20.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

11. File: src/ERC20Permit.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

[N-08] Event is missing `indexed` fields

Each `event` should use three `indexed` fields if there are three or more fields

1. File: src/interfaces/IAxelarGateway.sol (lines 10-16)

```
    event TokenSent(
```

```
        address indexed sender,
        string destinationChain,
        string destinationAddress,
        string symbol,
        uint256 amount
    );
```

2. File: src/interfaces/IAxelarGateway.sol (lines 18-24)

```
    event ContractCall(
        address indexed sender,
        string destinationChain,
        string destinationContractAddress,
        bytes32 indexed payloadHash,
        bytes payload
    );
```

3. File: src/interfaces/IAxelarGateway.sol (lines 26-34)

```
    event ContractCallWithToken(
        address indexed sender,
        string destinationChain,
        string destinationContractAddress,
        bytes32 indexed payloadHash,
        bytes payload,
        string symbol,
        uint256 amount
    );
```

4. File: src/interfaces/IAxelarGateway.sol (line 38)

```
    event TokenDeployed(string symbol, address tokenAddresses);
```

5. File: src/interfaces/IAxelarGateway.sol (line 62)

```
    event TokenFrozen(string symbol);
```

6. File: src/interfaces/IAxelarGateway.sol (line 64)

```
    event TokenUnfrozen(string symbol);
```

7. File: src/interfaces/IAxelarGatewayMultisig.sol (line 8)

```
    event OwnershipTransferred(address[] preOwners, uint256 prev
```

8. File: src/interfaces/IAxelarGatewayMultisig.sol (lines 10-15)

```
    event OperatorshipTransferred(
        address[] preOperators,
        uint256 prevThreshold,
        address[] newOperators,
        uint256 newThreshold
    );
```

9. File: src/interfaces/IERC20.sol (line 74)

```
    event Transfer(address indexed from, address indexed to, uir
```

10. File: src/interfaces/IERC20.sol (line 80)

```
    event Approval(address indexed owner, address indexed spende
```

[N-09] Unsafe early return from a modifier

If the modifier is used with a function that has a named return, the default value will be returned which may lead to confusing behavior. Consider using a function instead of a modifier

1. File: src/AdminMultisigBase.sol (lines 42-44)

```
if (adminVoteCount < _getAdminThreshold(adminEpoch)) ret

_;
```

[N-10] Consider allowing infinite approval

Doing what OpenZeppelin does and considering `type(uint256).max` to be infinite approval may help users to create smaller transactions.

1. File: src/ERC20.sol (line 105)

```
_approve(sender, _msgSender(), allowance[sender][_msgSer
```

[N-11] Consider using a two-step-transfer of ownership

The current owner would nominate a new owner, and to become the new owner, the nominated account would have to approve the change, so that the address is proven to be valid

1. File: src/Ownable.sol (line 20)

```
function transferOwnership(address newOwner) public virtual
```

[N-12] Consider adding a comment saying that EIP-2098 short signatures are not supported

1. File: src/ECDSA.sol (line 33)

```
if (signature.length != 65) revert InvalidSignatureLengt
```

deluca-mike (Axelar) commented:

> **Cross-chain replay attacks**
> Acknowledged, since just as that linked finding suggested, if Ethereum forks we

will not use any minority fork, since we are dealing with (bridging) assets that cannot be split.

Incorrect EIP-2612 deadline check
Confirmed.

Missing checks for address(0x0) when assigning values to address state variables
Confirmed. Will correct by checking code length of the address, which will also help with a delegatcall elsewhere in the contract.

Open TODOs
Confirmed, will remove TODO and move the function as noted.

Missing contract-existence checks before low-level calls

1. File: src/AxelarGateway.sol (lines 398-415) Disputed since that is not a low-level call, despite looking similar to one.
2. File: src/AxelarGatewayProxy.sol (line 19) Acknowledged, but given that it is a highly-used delegatecall as part of the proxy pattern, there is no point checking if the contract exists. If it does not, we have much much larger problems.
3. File: src/AxelarGatewayProxy.sol (line 34) As above, acknowledged, but given that it is a highly-used delegatecall as part of the proxy pattern, there is no point checking if the contract exists. If it does not, we have much much larger problems.
4. File: src/AxelarGateway.sol (line 350) Confirmed, but we will solve this but doing a `if (tokenDeployerImplementation.code.length == 0) revert InvalidTokenDeployer();` in the gateway's constructor, to prevent it form ever being initialized with an invalid TokenDeployer.

require()/revert() statements should have descriptive reason strings
Confirmed, and will be fixed with adopting if-revert with custom error messages in place of all requires.

public functions not called by the contract should be declared external instead
Confirmed.

constants should be defined rather than using magic numbers
Acknowledged, but these were intentional so they could be compared to Solidity Docs and standard ECDSA implementations.

Use a more recent version of solidity
Acknowledged, but these were all tested on 0.8.9, and this is locked in for release. The next release will take latest Solidity available at the time testing is started.

Multiple address mappings can be combined into a single mapping of an address to a struct, where appropriate
Disputed. While true, and would save gas, this would be a deviation from the ERC20 standard, and add complexity (in terms of standards and readability) for minimal gain.

Variable names that consist of all capital letters should be reserved for const/immutable variables
Confirmed. That variable should have been, and will be, made immutable.

File is missing NatSpec
Acknowledged, but more documentation will come in subsequent releases.

Each event should use three indexed fields if there are three or more fields
Acknowledged. Indexed fields actually cost a bit more gas, and should only be reserved for data that can be known ahead of time. Further, it is not yet clear which fields should be indexed, so they will only be indexed at a later release when we gather requirements. However, one should never index fields whose possible values are not part of some reasonable set (i.e. amount).

Unsafe early return from a modifier
Confirmed, however since all functions that currently use this modifier do not return data, we will just leave a comment above the modifier noting that it should be used with care.

Consider allowing infinite approval
Confirmed.

Consider adding a comment saying that EIP-2098 short signatures are not supported

> Acknowledged, but such a comment would be misplaced here, and would be better in normal documentation.

0xean (judge) commented:

> Severities proposed by warden seem correct.

Gas Optimizations

For this contest, 14 reports were submitted by wardens detailing gas optimizations. The **report highlighted below** by **IIIIIII** received the top score from the judge.

The following wardens also submitted reports: **Dravee**, **CertoraInc**, **0xkatana**, **Funen**, **0xNazgul**, **ilan**, **rfa**, **rayn**, **Tomio**, **Chom**, **0v3rf10w**, **Hawkeye**, *and* **nahnah**.

[G-01] Using `1` and `2` rather than `0` and `1` saves gas

See **this** issue from a prior contest for details

1. File: src/DepositHandler.sol (lines **6-7**)

```
uint256 internal constant IS_NOT_LOCKED = uint256(0);
uint256 internal constant IS_LOCKED = uint256(1);
```

[G-02] Not using the named return variables when a function returns, wastes deployment gas

1. File: src/Context.sol (line **17**)

```
return payable(msg.sender);
```

[G-03] Use a more recent version of solidity

Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

See original submission for list of instances.

[G-04] Using `bool`s for storage incurs overhead

```
// Booleans are more expensive than uint256 or any type that
// word because each write operation emits an extra SLOAD to
// slot's contents, replace the bits taken up by the boolean
// back. This is the compiler's defense against contract upg
// pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27

1. File: src/EternalStorage.sol (line 14)

```
mapping(bytes32 => bool) private _boolStorage;
```

[G-05] `<array>.length` should not be looked up in every loop of a `for`-loop

Even memory arrays incur the overhead of bit tests and bit shifts to calculate the array length

1. File: src/AxelarGatewayMultisig.sol (line 118)

```
for (uint256 i; i < accounts.length; i++) {
```

2. File: src/AxelarGatewayMultisig.sol (line 271)

```
for (uint256 i; i < accounts.length; i++) {
```

[G-06] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

See original submission for list of instances.

[G-07] `++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

1. File: src/AxelarGateway.sol (line 225)

```
        for (uint256 i; i < adminCount; i++) {
```

2. File: src/AdminMultisigBase.sol (line 51)

```
        for (uint256 i; i < adminCount; i++) {
```

3. File: src/AdminMultisigBase.sol (line 158)

```
        for (uint256 i; i < adminLength; i++) {
```

4. File: src/AxelarGatewayMultisig.sol (line 42)

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

5. File: src/AxelarGatewayMultisig.sol (line 118)

```
        for (uint256 i; i < accounts.length; i++) {
```

6. File: src/AxelarGatewayMultisig.sol (line 140)

```
for (uint256 i; i < ownerCount; i++) {
```

7. File: src/AxelarGatewayMultisig.sol (line 181)

```
for (uint256 i; i < accountLength; i++) {
```

8. File: src/AxelarGatewayMultisig.sol (line 271)

```
for (uint256 i; i < accounts.length; i++) {
```

9. File: src/AxelarGatewayMultisig.sol (line 293)

```
for (uint256 i; i < operatorCount; i++) {
```

10. File: src/AxelarGatewayMultisig.sol (line 332)

```
for (uint256 i; i < accountLength; i++) {
```

11. File: src/AxelarGatewayMultisig.sol (line 495)

```
for (uint256 i; i < signatureCount; i++) {
```

12. File: src/AxelarGatewayMultisig.sol (line 526)

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

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

1. File: src/AxelarGateway.sol (line 225)

```
for (uint256 i; i < adminCount; i++) {
```

2. File: src/AdminMultisigBase.sol (line 51)

```
for (uint256 i; i < adminCount; i++) {
```

3. File: src/AdminMultisigBase.sol (line 158)

```
for (uint256 i; i < adminLength; i++) {
```

4. File: src/AxelarGatewayMultisig.sol (line 118)

```
for (uint256 i; i < accounts.length; i++) {
```

5. File: src/AxelarGatewayMultisig.sol (line 140)

```
for (uint256 i; i < ownerCount; i++) {
```

6. File: src/AxelarGatewayMultisig.sol (line 181)

```
for (uint256 i; i < accountLength; i++) {
```

7. File: src/AxelarGatewayMultisig.sol (line 271)

```
for (uint256 i; i < accounts.length; i++) {
```

8. File: src/AxelarGatewayMultisig.sol (line 293)

```
for (uint256 i; i < operatorCount; i++) {
```

9. File: src/AxelarGatewayMultisig.sol (line 332)

```
        for (uint256 i; i < accountLength; i++) {
```

10. File: src/AxelarGatewayMultisig.sol (line 495)

```
        for (uint256 i; i < signatureCount; i++) {
```

11. File: src/AxelarGatewayMultisig.sol (line 526)

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

[G-09] Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead

> When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.

https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html
Use a larger size then downcast where needed

1. File: src/AxelarGateway.sol (line 62)

```
    uint8 internal constant OLD_KEY_RETENTION = 16;
```

2. File: src/AxelarGateway.sol (line 339)

```
        uint8 decimals,
```

3. File: src/ERC20.sol (line 43)

```
        uint8 public immutable decimals;
```

4. File: src/ERC20.sol (line 54)

```
        uint8 decimals_
```

5. File: src/AxelarGatewayMultisig.sol (line 363)

```
        (string memory name, string memory symbol, uint8 decimal
```

6. File: src/BurnableMintableCappedERC20.sol (line 24)

```
        uint8 decimals,
```

7. File: src/TokenDeployer.sol (line 11)

```
        uint8 decimals,
```

8. File: src/ECDSA.sol (line 38)

```
        uint8 v;
```

9. File: src/MintableCappedERC20.sol (line 17)

```
        uint8 decimals,
```

10. File: src/ERC20Permit.sol (line 39)

```
        uint8 v,
```

[G-10] Expressions for constant values such as a call to `keccak256()`, should use `immutable` rather than `constant`

See this issue for a detail description of the issue.

See original submission for list of instances.

[G-11] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

1. File: src/ERC20.sol (line 205)

```
        require(account != address(0), 'ZERO_ADDR');
```

[G-12] State variables only set in the constructor should be declared `immutable`

1. File: src/ERC20.sol (line 40)

```
    string public name;
```

2. File: src/ERC20.sol (line 41)

```
    string public symbol;
```

3. File: src/MintableCappedERC20.sol (line 12)

```
    uint256 public cap;
```

[G-13] Use custom errors rather than `revert()` / `require()` strings to save deployment gas

1. File: src/ERC20.sol (Various lines throughout the file)

2. File: src/DepositHandler.sol (Various lines throughout the file)
3. File: src/BurnableMintableCappedERC20.sol (Various lines throughout the file)
4. File: src/Ownable.sol (Various lines throughout the file)
5. File: src/AxelarGatewayProxy.sol (Various lines throughout the file)
6. File: src/MintableCappedERC20.sol (Various lines throughout the file)
7. File: src/ERC20Permit.sol (Various lines throughout the file)

[G-14] 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.

1. File: src/AxelarGateway.sol (line 234)

```
function freezeToken(string memory symbol) external override
```

2. File: src/AxelarGateway.sol (line 240)

```
function unfreezeToken(string memory symbol) external overri
```

3. File: src/AxelarGateway.sol (line 246)

```
function freezeAllTokens() external override onlyAdmin {
```

4. File: src/AxelarGateway.sol (line 252)

```
function unfreezeAllTokens() external override onlyAdmin {
```

5. File: src/AxelarGateway.sol (lines 258-262)

```
function upgrade(
    address newImplementation,
    bytes32 newImplementationCodeHash,
    bytes calldata setupParams
) external override onlyAdmin {
```

6. File: src/AxelarGatewayMultisig.sol (line 362)

```
function deployToken(bytes calldata params, bytes32) externa
```

7. File: src/AxelarGatewayMultisig.sol (line 371)

```
function mintToken(bytes calldata params, bytes32) external
```

8. File: src/AxelarGatewayMultisig.sol (line 377)

```
function burnToken(bytes calldata params, bytes32) external
```

9. File: src/AxelarGatewayMultisig.sol (line 383)

```
function approveContractCall(bytes calldata params, bytes32
```

10. File: src/AxelarGatewayMultisig.sol (line 404)

```
function approveContractCallWithMint(bytes calldata params,
```

11. File: src/AxelarGatewayMultisig.sol (line 429)

```
function transferOwnership(bytes calldata params, bytes32) e
```

12. File: src/AxelarGatewayMultisig.sol (line 440)

```
function transferOperatorship(bytes calldata params, bytes32
```

13. File: src/BurnableMintableCappedERC20.sol (line 48)

```
function burn(bytes32 salt) public onlyOwner {
```

14. File: src/Ownable.sol (line 20)

```
function transferOwnership(address newOwner) public virtual
```

15. File: src/MintableCappedERC20.sol (line 23)

```
function mint(address account, uint256 amount) public onlyOw
```

16. File: src/MintableCappedERC20.sol (line 31)

```
function burnFrom(address account, uint256 amount) external
```

deluca-mike (Axelar) commented:

> **Using 1 and 2 rather than 0 and 1 saves gas**
> Confirmed.

> **Not using the named return variables when a function returns, wastes deployment gas**
> Disputed, since in most cases it does not actually cost more gas, however, we are removing the Context contract anyway.

> **Use a more recent version of solidity**
> Acknowledged, but these were all tested on 0.8.9, and this is locked in for release. The next release will take latest Solidity available at the time testing is started.

Using bools for storage incurs overhead
Acknowledged, but it's better to use bools for readability in many cases.

.length should not be looked up in every loop of a for-loop
Disputed, since recent versions of solidity optimize for this if the array has a fixed length.

Using calldata instead of memory for read-only arguments in external functions saves gas
Confirmed.

++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
Confirmed for using ++i instead of i++, but acknowledged for the unchecked suggestion, which we will forgo for readability, for now.

Usage of uints/ints smaller than 32 bytes (256 bits) incurs overhead
Acknowledged, but will not change since the overhead is at worst minimal, and much of this functionality/code is shared in the space. Further, it helps with readability.

Expressions for constant values such as a call to keccak256(), should use immutable rather than constant
Disputed, since the issue being referred to is 2 years old, and since then, Solidity computes the literals at compile-time.

Duplicated require()/revert() checks should be refactored to a modifier or function
Acknowledged, but will not change as this is a design preference and the compiler can optimize out duplicate code.

State variables only set in the constructor should be declared immutable
While true, and thus confirmed, for cap, this is not true, and thus disputed for strings, since non-value types cannot be immutable in Solidity (yet).

Use custom errors rather than revert()/require() strings to save deployment gas
Confirmed.

> **Functions guaranteed to revert when called by normal users can be marked payable**
> Acknowledged, but will not do since it is gas savings at the expense of confusion, readability issues, and the possibilities for ETH to be received.

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