SquidRouter Security Review

Auditors
0xKaden, Security Researcher

May 1, 2024

1 Executive Summary

Over the course of 5 days in total, Squid engaged with 0xKaden to review SquidRouter.

Metadata

Repository	Commit
squid-contracts	6fcdd2f

Summary

Type of Project	Cross-Chain Messaging
Timeline	April 25th, 2024 - May 1st, 2024
Methods	Manual Review

Total Issues

Critical Risk	1
High Risk	0
Medium Risk	2
Low Risk	4
Gas Optimizations	0

Contents

2 Introduction

Squid is a liquidity and messaging router on Axelar.

The focus of the security review was on the following contracts:

1. SquidRouter.sol
2. SquidMulticall.sol

Disclaimer: This review does not make any warranties or guarantees regarding the discovery of all vulnerabilities or issues within the audited smart contracts. The auditor shall not be liable for any damages, claims, or losses incurred from the use of the audited smart contracts.

3 Findings

3.1 Critical Risk

3.1.1 Incorrectly encoded metadata results in lost funds

Severity: Critical

Context: SquidRouter.sol#L283

Description:

In `SquidRouter.itsBridgeCall`, we create the `metadata` to provide by encoding the `enableExpress` boolean alongside the provided `payload` before passing it to the `InterchainTokenService.interchainTransfer` call:

```
bytes memory metadata = abi.encode(enableExpress ? uint32(1) : uint32(0), payload);
IInterchainTokenService(interchainTokenService).interchainTransfer(
    bridgedTokenId,
    destinationChain,
    destinationAddress,
    amount,
    metadata,
    msg.value
);
```

In `InterchainTokenService.interchainTransfer`, we decode the `metadataVersion` from the payload with `_decodeMetadata`:

```
(MetadataVersion metadataVersion, bytes memory data) = _decodeMetadata(metadata);
```

`_decodeMetadata` works by retrieving the first 4 bytes of the metadata as the `MetadataVersion` and returning the remaining data excluding the version:

```
function _decodeMetadata(bytes calldata metadata) internal pure returns
↪  (MetadataVersion version, bytes memory data) {
    if (metadata.length < 4) return (MetadataVersion.CONTRACT_CALL, data);

    uint32 versionUint = uint32(bytes4(metadata[:4]));
    if (versionUint > LATEST_METADATA_VERSION) revert
    ↪  InvalidMetadataVersion(versionUint);

    version = MetadataVersion(versionUint);

    if (metadata.length == 4) return (version, data);

    data = metadata[4:];
}
```

The problem with all this lies in the fact that the ABI encoded metadata version in `itsBridgeCall` is actually encoded to a 32 byte value contrary to the expected 4 byte value. We can test the result of the ABI encoding in `chisel`:

```
 bytes memory a = abi.encode(uint32(1), uint256(1))
 a
Type: dynamic bytes
 Hex (Memory):
 Length ([0x00:0x20]):
↪  0x0000000000000000000000000000000000000000000000000000000000000040
 Contents ([0x20:..]):
↪  0x0000000000000000000000000000000000000000000000000000000000000001
    0000000000000000000000000000000000000000000000000000000000000001
```

We can see above that even though we encoded a `uint32`, the encoded value is 32 bytes as opposed to the intended 4 bytes.

The result of this is that our `payload` that gets received in `_executeWithInterchainToken` on the destination chain will start with an unexpected 28 bytes. Processing this `payload` will thus fail, causing the transaction to get reverted. Since we cannot modify the `payload`, we will never be able to successfully execute the transaction and the funds will be permanently lost.

Recommendation:

Concatenating fixed length byte arrays in Solidity is notoriously difficult. The safest solution appears to be to compute `payload` with the 4 byte version prefix off-chain and removing the `enableExpress` boolean accordingly since it is not otherwise used.

3.2 High Risk

No high risk findings were discovered.

3.3 Medium Risk

3.3.1 Unsafe paused functions

Severity: Medium

Context:

- SquidRouter.sol#L368
- SquidRouter.sol#L483

Description:

Part of this audit was to validate that usage of the new pausable mechanism of upgrading the contract to the SquidRouterPaused contract is secure. This logic works by simply upgrading the entire SquidRouter contract to SquidRouterPaused which simply reverts for nearly every function call. This pattern has been deemed to be unsafe for two functions: cfReceive and rescueFunds.

cfReceive is a function called by Chainflip on the destination chain to transfer tokens to the router and execute arbitrary calls via SquidMulticall. As documented, "The receiver on the destination chain needs to ensure that the receiving logic won't revert." Using the SquidRouterPaused pattern fails to comply with this documented expectation, causing cfReceive to revert unexpectedly.

As a fallback, Chainflip allows the user to sign the payload later on to re-broadcast the transaction. However, the payload is only, "valid until two key rotations have taken place."

As for rescueFunds, this is an emergency method which is used to withdraw any funds which have been accidentally left in the contract. Causing this function to revert during the paused state is likely undesirable.

Recommendation:

Use a regular Pausable implementation as was used previously, including the whenNotPaused modifier on functions which are safe to pause and not on functions which are unsafe to pause such as cfReceive and rescueFunds.

3.3.2 SquidMulticall is unsafe to be used by contracts or EOA's which hold funds non-atomically

Severity: Medium

Context:

- SquidMulticall.sol#L39
- SquidMulticall.sol#L51

Description:

SquidMulticall.run and SquidMulticall.collectAndRun are functions that can be called by anyone and execute an array of arbitrary calls. Both functions include a safeTransferFrom with the msg.sender as the from address, implying that usage of the contract generally requires

approving the contract to spend tokens on behalf of the user. Since anyone can execute arbitrary calls, any token approvals applied to `SquidMulticall` can be stolen by attackers by simply calling `transferFrom` on the token contract to transfer the funds to themselves.

As indicated by the Squid team, `SquidMulticall` is not intended to be used by contracts which hold funds non-atomically. However, there is a significant lack of documentation indicating this. Furthermore, along with the lack of documentation, the ability for anyone to execute these functions implies that the contract can be used arbitrarily for similar use cases as used in the system.

Recommendation:

It should be clearly documented that `SquidMulticall` should only be used by contracts which are never expected to hold funds non-atomically. Additionally, it's recommended that `run` and `collectAndRun` are restricted to only be callable by the intended contract(s). Note that restricting who can call these functions does not prevent accidental approvals from being stolen, as they can still be executed via the router, but it does clarify the expectation as to how the contract should be used.

3.4 Low Risk

3.4.1 Format inconsistent with documentation

Severity: Low

Context: SquidRouter.sol#L500-L503

Description:

In `SquidRouter._setup`, we document the expected format for the `data` parameter as:

```
abi.encode(address pauser, address approver)
```

However, the decoding logic used is inconsistent with the documented format:

```
address _approver = abi.decode(data, (address));
```

We can see above that the `pauser` is not included in the `data` and only the `approver` is decoded. Following the documented format when performing an upgrade may result in the address intended as the `pauser` being set as the `approver`.

Recommendation:

Consider whether the documented format or implemented logic is correct. Assmuing the implemented logic is correct, update the documented expected format to be consistent with the logic used:

```
-/// Expected format is: abi.encode(address pauser, address approver).
+/// Expected format is: abi.encode(address approver).
```

3.4.2 Unexpected reverts on bridge receivers result in loss of funds

Severity: Low

Context: SquidRouter.sol#L367-L425

Description:

SquidRouter contains functions which are called by bridge protocols to manage bridged funds on the destination contract as defined by execution on the source contract: cfReceive, _executeWithToken* and _executeWithInterchainToken*. Each of these functions expects the proceeding execution to not revert, else the token transfers will not be completable, resulting in lost funds.

An unexpected revert can occur due to a couple possible circumstances:

- Improperly encoded payload can cause abi.decode to revert.
- Any reverting calls in the multicall execution cause the entire transaction to revert.

The latter can occur due to non-deterministic function execution, i.e. a function that may succeed some times but will revert other times. A common example of this is a swap with a defined slippage limit. A price change between source chain initiation and destination chain reception may reasonably cause the slippage limit to be exceeded, causing execution to revert. Unexpected reverts may also occur due to a number of other circumstances, including multicall ordering and transaction timing.

*These functions are actually internal, with their external counterparts that are being called by the bridge protocols being present in inherited contracts. For the sake of simplicity, these functions are referenced instead since they are relevant to the proceeding execution as discussed in this finding.

Recommendation:

A few procedural steps can be taken to avoid unexpected reverts on the bridge receivers:

- Audit the off-chain encoding logic used to ensure that it never incorrectly encodes the payload.
- Avoid non-deterministic function execution on the destination chain. Documentation should be added accordingly which explains this.
- Use an off-chain transaction simulation tool prior to execution to validate that the execution is expected to succeed under current conditions.

3.4.3 Typos and misleading comments

Severity: Low

Context:

- SquidRouter.sol#L212
- SquidRouter.sol#L429-L437

- SquidRouter.sol#L427-L428
- ISquidMulticall.sol#L15

Description:

There are a few instances in the codebase where comments are either misspelled, incorrect, or misleading. Fixing these instances can improve contract readability and maintainability.

1. Incorrect payload format

In several places, the documented `payload` format is indicated as containing a `bytes32 salt`, while recent changes have modified the used `salt` to be a `bytes8` value. Note that this has already been fixed in a commit following the commit hash in review.

2. Contradictory comment

In the `_processPayload` NatSpec documentation, we note that the function, "Does not work with native token." Yet we later indicate that the amount provided, "Must match msg.value if native tokens." Presumably, we should remove the latter to avoid confusion.

3. Spelling and grammar mistakes

Also in the `_processPayload` NatSpec documentation, we state, "Check size of payload and processes is accordingly. If there is no calls, send tokens directly to user. If there are calls, run them." This should be fixed to instead say, "Checks size of payload and processes it accordingly. If there are no calls, send tokens directly to user. If there are calls, run them."

4. Incorrectly documented CallType

We document the `CallType.FullNativeBalance` as updating the calldata parameter similarly to `CallType.FullTokenBalance`. However, this is not the intended or actual functionality. Instead, we should update this comment to indicate that `CallType.FullNativeBalance` will set the value of the call as the native token balance of the multicall contract.

Recommendation:

Apply the recommendations as described above.

3.4.4 Setting the calldata parameter is not possible for nested calls

Severity: Low

Context: SquidMulticall.sol#L101

Description:

The `SquidMulticall._setCalldataParameter` logic works by taking a given parameter index and updating 32 bytes of memory corresponding to the offset which that index points to:

```
assembly {
    // 36 bytes shift because 32 for prefix + 4 for selector
    mstore(add(callData, add(36, mul(parameterPosition, 32))), value)
}
```

The problem with this logic is that it assumes that any possible parameter to set must have a byte offset with a multiple of 32 and remainder of 4. This is generally the case due to ABI encoding, but there are some circumstances where this is not the case. For example, if the calldata contains a nested call and it's desirable to set one of the parameters within this nested call, it will not be possible because it will be shifted by 4 bytes to support the function selector.

Recommendation:

Rather than providing a `parameterPosition` which is multiplied by 32, we can instead provide a `parameterOffset` which points directly to the start of the parameter under all circumstances.

3.5 Gas Optimizations

No gas optimizations were discovered.