Squid

Date	September 2022
Auditors	Tejaswa Rastogi, Christian Goll

1 Executive Summary

This report presents the results of our engagement with **Squid** to review SquidRouter, their cross-chain swap and liquidity routing protocol.

The review was conducted over two weeks, from **September 19, 2022** to **September 27, 2022**. A total of 2x7 person-days were spent.

It should be noted that instances of concentrated risk with the current design were found particularly reentrancy and loss of funds thanks to reliance on generalised low-level calls. Given the generalised nature of the calls, there are no guard rails in place to prevent the latter and other scenarios such as calls to non-existent contracts in `SquidMulticall` both on the source chain and the destination chain or calls to non-EVM chains.

A re-design of the contracts should be considered to add more guard rails. The current concept of sending funds to a contract, from which anyone can call arbitrary code, might be too liberal and generic to be properly secured in it's current state.

2 Scope

Our review focused on the following repositories:

- github/0xsquid/squid-core@1a9a9ff71b6f86f45b907b0d5906d8595bf5a0e4

The following files were explicitly included in the review:

- `packages/squidswap-contracts/contracts/SquidRouter.sol`
- `packages/squidswap-contracts/contracts/SquidMulticall.sol`
- `packages/squidswap-contracts/contracts/SquidProxy.sol`

The following documentation was provided: Notion: Squid Smart Contract Docs, docs.0xsquid.com

Over review for audit fixes focused on the following repositories:
github/0xsquid/squid-core@124176215b746fa367859e0b2555296e0d67b82e

2.1 Objectives

Together with the Squid team, we identified the following priorities for our review:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.
3. Reentrancy protection
4. Calls with arbitrary data

3 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

3.1 `SquidMulticall` - Possible reentrancy Major Partially Addressed

Resolution

The squid team has implemented a reentrancy guard to SquidMulticall's `run` function. It can help in preventing reentrant calls that can emerge from cases like receiver hooks of tokens. However, it is still insufficient to prevent cases where an attacker may try to steal funds using token approvals from the contract to itself as mentioned in Para. 3 in Description. The audit team recommends verifying the operational logic and redesigning the logic.

Description

The function `run` allows the execution of generalized low-level calls, which in some cases, may lead to reentrancy, allowing an attacker to steal funds from the victim.

For instance, a token transfer with a *before/after* transfer hook enables the recipient to make a reentrant call to the `run` function and extract available funds at that point in time. The stolen funds may be from a token swap that a user is still left to process in subsequent calls.

The funds can be stolen even without making a reentrant call. An attacker can make an `approve` call to its address with maximum allowance and wait for a *before/after* transfer hook to trigger. After which, an attacker can just transfer available funds at that point in time from the `SquidMulticall` .

Examples

code/packages/squidswap-contracts/contracts/SquidMulticall.sol:L26-L27

```
(bool success, ) = call.target.call{value: call.value}(call.callData);
if (!success) revert CallFailed(i);
```

Recommendation

Some Possible measures that may be taken: 1- Using ReentrancyGuard for the `run` function 2- Restricting calls to the `run` function to `SquidRouter` only. 3- Adding ReentrancyGuard to `SquidRouter` if, under normal cases, no call is expected to reenter `SquidRouter` .

To tackle scenarios of `approve` calls, there is a need to redefine the function logic.

3.2 `SquidMulticall` - no isContract check in `SquidMulticall.run()` Medium Acknowledged

Resolution

Comment from Squid Team: "We consider this as a marginal error resulting from protocol misusage rather than a security flaw. Squid does not expect the `target` to ever intentionally be an EOA, and hence including a boolean flag to the struct `call` is not required."

Also, the squid team suggests that Apps building on top of Squid should be careful to validate `calldata` encoded into contract calls.

However, the audit team still believes that it's a security flaw and suggests working on the recommendations. The `Multicall` contract is already validating the target address to be `contract` for `CallType.FullTokenBalance` , and thus the logic can be adjusted for Default callType as well.

https://github.com/ConsenSysDiligence/squid-audit-2022-09/blob/bb15b3c6a6d82d6a193753f04309f100c0f77bfa/code/packages/squidswap-contracts/contracts/SquidMulticall.sol#L41

Another strategy could be, to validate the `call.target` to be a contract if there exists a non-empty `call.callData` .

Description

There is no functionality to check whether `call.target` has code at the address. If a user accidentally inputs the wrong address as a target i.e, one with no code deployed, the return value will always be true even though the transaction wasn't completed. See here

Examples

code/packages/squidswap-contracts/contracts/SquidMulticall.sol:L26

```
(bool success, ) = call.target.call{value: call.value}(call.callData);
```

Recommendation

Add a boolean flag to the struct `Call` indicating whether the target is meant to be a contract or an EOA and checking if there is code deployed at the address if it is indeed meant to be a contract call.

3.3 Piggybacking/Extracting Tokens Medium Acknowledged

Resolution

Comment from Squid Team: "We consider it as a marginal error resulting from protocol misusage rather that security flaw. Implementing such logic on chain would significantly increase gas cost to address something like .001% of cases. Also if there is no call to execute, user should rather use callBridge instead of callBridgeCall, or directly use axelar bridge instead of bridgeCall."

However, the audit team still believes adding the required logic may save users funds in case they get stuck accidentally and recommends the same.

Description

The `SquidRouter` executes user-defined calls on `SquidMulticall`. Under normal circumstances, the contracts are not supposed to hold any tokens. However, for any reason, if the tokens get stuck in either of the contracts, then any user can extract/piggyback them in an arbitrary call.

Some scenarios where the tokens may get stuck are:

1. If there are no calls to execute on the destination chain(`_executeWithToken`), i.e `calls.length == 0`.
2. If the user didn't craft an actionable call to transfer the received funds in `SquidMulticall` to the desired address.

As `SquidMulticall.run()` can be called by any external actor, anyone can extract funds or piggyback them in any arbitrary call from the `SquidMulticall` or event fetch from `SquidRouter` as it gives max allowance for token transfers.

Examples

code/packages/squidswap-contracts/contracts/SquidMulticall.sol:L26-L27

```
(bool success, ) = call.target.call{value: call.value}(call.callData);
if (!success) revert CallFailed(i);
```

code/packages/squidswap-contracts/contracts/SquidRouter.sol:L162-L171

```
function _approve(
    address tokenAddress,
    address spender,
    uint256 amount
) private {
    if (IERC20(tokenAddress).allowance(address(this), spender) < amount) {
        // Not a security issue since the contract doesn't store tokens
        IERC20(tokenAddress).approve(spender, type(uint256).max);
    }
}
```

Recommendation

An approach to handle this may be:

1. Transfer funds to `refundRecipient`, if there are no calls to execute on `SquidMulticall`
2. Transfer funds to `refundRecipient`, if, after executing all the calls, there exists any unclaimed balance in the contract.

Also, the allowance should be limited to the actual balance being transferred to an address.

3.4 `SquidRouter` - Execution may fail due to blacklisted `refundRecipient` Minor Acknowledged

Resolution

Comment from Squid Team: "As this type of blacklisting is not standardized, we won't be able to check for it consistently and leave this at the responsibility of the user to not use a blacklisted address."

Description

Contract implements a safety fallback mechanism to deal with cases where a call execution may fail by transferring the funds to a user-chosen address as `refundRecipient`. However, it may happen, that the chosen `refundRecipient` is a blacklisted address for a particular token, thus reverting execution on the destination chain.

Examples

code/packages/squidswap-contracts/contracts/SquidRouter.sol:L124-L130

```
try squidMulticall.run(calls) {
    emit CrossMulticallExecuted(keccak256(payload));
} catch (bytes memory reason) {
    // Refund tokens to refund recepient if swap fails
    _safeTransfer(bridgedTokenAddress, refundRecipient, amount);
    emit CrossMulticallFailed(keccak256(payload), reason, refundRecipient);
}
```

Recommendation

Consider verifying the destination chain's `refundRecipient` prior to making a bridge call.

3.5 `SquidRouter` - No `destinationChain` validation. Minor Acknowledged

Resolution

Squid team believes that validating `destinationChain` on-chain will be inefficient and not a favorable smart contract design. Also, the team comments that the transaction will fail on Axelar network if an incorrect chain is provided, and may be retried in some cases. However, refunds from the Axelar gateway for messages which contain invalid destination chain transactions are not currently supported.

Description

In the SquidRouter, several functions take `destinationChain` as an input parameter; however, they are never validated. The Axelar docs state that general message passing is only supported on EVM chains, with only token bridging being supported on non-EVM chains. This opens up scenarios where multicall user transactions fail once execution is passed to the bridge. This wastes unnecessary gas on the user's part and makes debugging challenging.

Examples

code/packages/squidswap-contracts/contracts/SquidRouter.sol:L71-L95

```
function callBridgeCall(
    address token,
    uint256 amount,
    string calldata destinationChain,
    string calldata bridgedTokenSymbol,
    ISquidMulticall.Call[] calldata sourceCalls,
    ISquidMulticall.Call[] calldata destinationCalls,
    address refundRecipient,
    bool enableForecall
) external payable whenNotPaused {
    _fundAndRunMulticall(token, amount, sourceCalls);

    address bridgedTokenAddress = gateway.tokenAddresses(bridgedTokenSymbol);
    uint256 bridgedTokenAmount = IERC20(bridgedTokenAddress).balanceOf(address(this));

    _bridgeCall(
        destinationChain,
        bridgedTokenSymbol,
        bridgedTokenAddress,
        bridgedTokenAmount,
        destinationCalls,
        refundRecipient,
        enableForecall
    );
}
```

Recommendation

Add functionality to check whether the `destinationChain` is supported for general message parsing or not and log errors to ease debugging.

3.6 `SquidRouter` - Unforeseen Risks due to Axelar Bridge Failure Minor Acknowledged

Resolution

The squid team acknowledges the risk. Comment from Squid Team: "We have no power over this risk. Using Squid implies accepting the inherent risk of the underlying Axelar protocol."

Description

The contract allows users to perform cross-chain transactions via Axelar Network. The first step is to emit a `ContractCallWithToken` / `TokenSent` event from the gateway. However, there may exist some unforeseen risks due to bridge failure as:

1. Axelar Network, is unable to capture/confirm a valid bridge request, thus causing loss to the end user.
2. Axelar Network, is unable to relay the original transaction which has been already forecalled at the destination chain, thus causing loss to forecallService.

Examples

code/packages/squidswap-contracts/contracts/SquidRouter.sol:L68

```
gateway.sendToken(destinationChain, destinationAddress, bridgedTokenSymbol, bridgedTokenAmount);
```

code/packages/squidswap-contracts/contracts/SquidRouter.sol:L159

```
gateway.callContractWithToken(destinationChain, stringAddress, payload, bridgedTokenSymbol, amount);
```

Recommendation

It is recommended to implement the required security schemes to tackle these scenarios.

3.7 `SquidMulticall` - Inadequate Error Handling ✓ Fixed

Resolution

The function now reverts with return data from the low-level call with the call number.

Description

The contract allows the execution of multiple user-defined low-level calls. If for any reason, any call fails, the function reverts to a custom error and the number of the specific call that caused the revert. However, it is inadequate and makes debugging quite difficult.

Examples

code/packages/squidswap-contracts/contracts/SquidMulticall.sol:L26-L27

```
(bool success, ) = call.target.call{value: call.value}(call.callData);
if (!success) revert CallFailed(i);
```

Recommendation

As most of the well-known protocols return error statements while reverting a transaction. A better approach could be to capture this revert reason and attach it to the custom error with the specific call number that failed.

3.8 Structured Storage may lead to Storage Collision ✓ Fixed

Resolution

The unstructured layout is now being followed for the state variables with the help of RoledPausable.

Description

`SquidRouter` inherits OpenZeppelin's `Pausable` contract, which stores the `_paused` boolean variable at storage slot 0. Currently, `SquidProxy` doesn't define structured storage for variables. However, if the team decides to add some variables in a future version, it may lead to storage collision with `_paused` and produce erroneous results.

Examples

```
bool private _paused;
```

Recommendation

It is best practice to define unstructured storage for all the storage variables, i.e., storing data in specific storage slots, like the way contract stores values for owner and implementation address, to avoid any unintended storage collision.

3.9 `SquidMulticall` - Gas critiques ✓ Fixed

Resolution

Squid team claimed that while testing the recommended optimizations are making operations more time and gas expensive and hence decided to not go with them. The audit team however has not verified these costs.

Description

Given this is meant to enable multiple transactions, gas efficiency is a concern. With that in mind, in `SquidMulticall.run()`, the current code is written, so the compiler reads the array length on every iteration resulting in extra gas usage i.e. an extra `calldataload` which is 3 gas on every iteration except the first.

Moving on, the compiler enforces overflow checks on the value of `i`, which is unneeded because the value of `i` is always strictly less than `length <= 2**256 - 1`, so the maximum value of `i` to enter the for-loop body is `2**256 - 2`. Regardless, the compiler still checks for overflow, using up more gas than necessary.

Finally, consider

Examples

code/packages/squidswap-contracts/contracts/SquidMulticall.sol:L10-L12

```
function run(Call[] calldata calls) external payable {
    for (uint256 i = 0; i < calls.length; i++) {
        Call memory call = calls[i];
```

Recommendation

Cache the array length in the stack and make the increment unchecked.

```
- for (uint i = 0; i < array.length; i++) {
-     // do something that doesn't change arr.length
-}

+ uint length = array.length;
+ for (uint i = 0; i < length; ) {
+     // do something that doesn't change array.length
+.    unchecked {
+         ++i
+     }
+}
```

3.10 `.env` is uploaded to shared repo exposing private keys ✓ Fixed

Resolution

The file has now been removed

Description

The `.env` file is added to the shared repo. Luckily, it's just a burner; however it is highly advised to add `.env` and other config files with sensitive information to a `.gitignore` file.

Examples

code/packages/squidswap-contracts/.env:L1

```
PRIVATE_KEY="0xa0aa5ccccc27c64427d53213144ad0ec84257980e7f26bf77a2932332203bab9"
```

Recommendation

Add `.env` to `.gitignore`.

3.11 `SquidRouter` - Lack of input validation ✓ Fixed

Resolution

The recommended checks have now been added

Description

Considering the value of several immutable variables are set at construction, it is recommended to perform input validation to avoid loss of functionality or other unexpected behaviour including extra gas expenditure due to redeployments.

Examples

code/packages/squidswap-contracts/contracts/SquidRouter.sol:L17-L30

```
IAxelarGasService private immutable gasService;
IAxelarGasService private immutable forecallGasService;
ISquidMulticall private immutable squidMulticall;

constructor(
    address _gateway,
    address _gasService,
    address _forecallGasService,
    address _multicall
) AxelarForecallable(_gateway) {
    gasService = IAxelarGasService(_gasService);
    forecallGasService = IAxelarGasService(_forecallGasService);
    squidMulticall = ISquidMulticall(_multicall);
}
```

Recommendation

Add zero address checks and validate the values set in the deploy scripts ahead of time.

3.12 `SquidRouter` - Missing 2-step approach for changing privileged role Acknowledged

Resolution

Squid Team doesn't consider it relevant to write their own version to accompany this logic as Axelar's contract is audited and well-tested. However, we recommend handling ownership transfers with utmost care.

Description

`SquidRouter` uses a single-step approach to transfer ownership to a new address, which imposes a severe risk, as accidental transfers may leave the contract with a malicious owner or none at all leading to loss of functionality.

Examples

```
    function transferOwnership(address newOwner) external virtual onlyOwner {
        if (newOwner == address(0)) revert InvalidOwner();

        emit OwnershipTransferred(newOwner);
        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(_OWNER_SLOT, newOwner)
        }
    }
```

Recommendation

It is recommended to use a 2-step approach for transferring ownership.

An example approach may be: 1 - Current Owner proposes ownership, which will instantiate a Pending Owner. 2 - The Pending Owner may then accept the ownership proposal to become the new owner in a separate transaction.

3.13 Inadequate NatSpec Comments Acknowledged

Resolution

Comment from Squid Team: “We don’t consider full NatSpec commented contracts relevant as they are mostly redundant with the code namings. We believe they clutter the code base and are prone to becoming stale and misleading as new developers update code, but sometimes fail to update the NatSpec.”

However, we still believe that NatSpec Comments can increase readability and suggest adding them wherever necessary.

Description

The contracts in scope lack NatSpec comments which provide rich documentation for the functions, expected input, and return values. They also help developers to get an expected logical breakdown of a function.

Examples

code/packages/squidswap-contracts/contracts/SquidRouter.sol:L32-L39

```
function bridgeCall(
    string calldata destinationChain,
    string calldata bridgedTokenSymbol,
    uint256 amount,
    ISquidMulticall.Call[] calldata calls,
    address refundRecipient,
    bool enableForecall
) external payable whenNotPaused {
```

Recommendation

Add NatSpec comments to keep the code well documented and increase readability.

Appendix 1 - Disclosure