Bifrost

XCM Actions

Smart Contracts Audit

Bifrost

Jul 5, 2023

Common Prefix

Overview

Introduction

Common Prefix was commissioned to perform a security audit on Bifrost's' XCM actions smart contracts, at commit hash 103d44154a3183acc8406c9aff876ccb97ae4d0c. The files inspected are the following:

```
├── AstarXcmAction.sol
├── MoonbeamXcmAction.sol
└── utils
    ├── AddressToAccount.sol
    ├── Blake2b.sol
    └── BuildCallData.sol
```

Note: After the completion of the audit, the Bifrost team changed the names of the contracts as follows:

`AstarXcmAction.sol -> AstarSlpx`

`MoonbeamXcmAction.sol -> MoonbeamSlpx`

We confirmed that the code at commit hash 28c701a314ffb24b885db113b9950e9ae60d2dfb except for the renaming is the same compared to the code of the audit and also includes all the fixes suggested in this report.

Protocol Description

The XCM Actions contracts allow users of the Astar and Moonbeam Polkadot's parachains to interact with Bifrost, executing actions such as minting vTokens, redeeming vTokens and swapping tokens. The contracts use the precompiles XCM (of Astar) and XTokens, XcmTransactorV2 (of Moonbeam) which incorporate the Polkadot XCM pallet to send messages and tokens to other parachains.

The cross-chain communication is possible only if the Substrate address corresponding to an EVM address format is known. The procedure of transforming one address format to the other requires the use of the Blake2b hash function. This hash function is described in RFC 7963. The XCM contracts use the Blake2b function without a key and for this case the Blake2b.sol contract follows closely the RFC. Moreover, we tested the contracts, using them to transform a list of EVM addresses to Substrate accounts and using the following as a standard https://snow-address-converter.web.app/, and confirmed that the address transformation works as expected. But for a non-empty key the Bifrost team's implementation of the Blake2b hash function does not follow closely the RFC. We mention these deviations below and we strongly advise that the Blake2b.sol, if not reworked for the non-empty key case, should not be used outside of the scope of the XCM Actions contracts.

Disclaimer

Note that this audit does not give any warranties on the bug-free status of the given smart contracts, i.e. the evaluation result does not guarantee the nonexistence of any further findings of security issues. This audit report is intended to be used for discussion purposes only. Functional correctness should not rely on human inspection but be verified through thorough testing. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the project.

Findings Severity Breakdown

The findings are classified under the following severity categories according to the impact and the likelihood of an attack.

Level	Description
Critical	Logical errors or implementation bugs that are easily exploited and maylead to any kind of loss of funds
High	Logical errors or implementation bugs that are likely to be exploited and may have disadvantageous economic impact or contract failure
Medium	Issues that may break the intended contract logic or lead to DoS attacks
Low	Issues harder to exploit (exploitable with low probability), issues that lead to poor contract performance, clumsy logic or seriously error-prone implementation
Informational	Advisory comments and recommendations that could help make the codebase clearer, more readable and easier to maintain

Findings

Critical

No critical issues found.

High

HIGH-1	When the output length is 16 the contract does not return the correct 16 bytes
Contract(s)	`Blake2b.sol`
Status	**Resolved**

Description

In Blake2b::finalize, when the output length is 16, the following assembly code is executed:

```
assembly {
    mstore(add(output, 16), mload(add(state, 20)));
    mstore(output, 16);
           }
```

The first mstore stores the mload(add(state, 20)) value i.e. the value stored in the word which starts at the address 20 bytes after the state address, but this word is part of the first 32 bytes word of state where the length of the state bytes array is stored. These are not the first 16 bytes of the part of state describing h, which should start at add(state,36). Moreover, this value is stored in add(output,16) which is again part of the first 32 bytes word of the output, where the length of the output bytes array should be stored, and not in the correct place add(output,32).

Recommendation

We suggest fixing this bug correcting the offsets in the mload and mstore opcodes. To keep only 16 bytes instead of a whole 32 bytes word a bitwise and with an appropriate bitmask can be used.

Alleviation

The team fixed this issue at commit hash 1fa5ea69aaf9559a6cc2a2b24cdc54c322f953e9, modifying the contract to only support 32 bytes output length, which is the output length needed in the XCM contracts.

Medium

MEDIUM-1	It is not confirmed that the messages corresponding to each action were successfully sent
Contract(s)	`AstarXcmAction.sol`
Status	**Resolved**

Description

Every action is executed in two steps:

1. Transfer an amount in Bifrost.
2. Send a message using the XCM pallet describing what action should be performed on the Biforst side i.e. minting, redeeming or swapping tokens.

The functions executing these steps are implemented in the XCM precompiled contract and each one returns a boolean (true if the message was successfully sent, false otherwise). But the contract does not check these booleans and the tx will be always completed even if they are false and the message was not even sent. If the messages are not successfully sent all the tokens are being sent back to the user and that's why we mark this issue just as medium, but it will definitely look like an inconsistency for the users if the action seems to be executed and the messages were not sent.

Note that even if the message was successfully sent, it does not mean that it will necessarily be interpreted and executed in the target chain (if this happens then again all the actions are reverted and the user does not lose any tokens), but nothing can be done about this in the XCM Actions contracts.

Recommendation

We suggest checking the value of these booleans and reverting if they are not both true.

Alleviation

The team added all the required checks and fixed this issue at commit hash 7b44ef2c7de6b0bdd705240456c1380df0b39f74.

MEDIUM-2	The implementation does not follow closely the Blake2b specs when the key is not empty
Contract(s)	`Blake2b.sol`
Status	**Resolved**

Description

There are several points where the behavior of the Blake2b.sol is not the expected one as it is described in the RFC about the Blake2b hash function. All these problems arise when a non-empty key is being used, therefore they do not affect the XCM contracts where the Blake2b function is used with no key only and most of these problems are known to the team and there are comments in the code mentioning them, but for completeness, we present a list with issues we have identified:

- The out_len variable of the Instance struct (denoted as nn in the RFC) and used as an argument of the init and reset functions, should be checked that it is less than or equal to 64 (bytes).
- The key and the input data are not padded with 0s to form words of 128 bytes.
- There is no distinction in the last round for the cases zero length and non-zero length key.
- The RFC says that the output of the hash function can be any number from 1 to 64 bytes, but the contract can have only outputs of lengths 16, 32 or 64.
- The last part of the reset function, the if (key_len>0) branch, will always revert since the conditions require(key_len == 64) and assert(key.length == 128) are contradictory, as long as the key is not padded with 0s after the require statement. Moreover this require allows keys with 64 bytes only (if non-empty), but the RFC allows any key length between 0 and 64 bytes.

Recommendation

We suggest fixing all these issues and making the contract behave as expected by the RFC, if this contract is going to be used outside of the XCM Actions protocol.

Alleviation

The team fixed this issue at commit hashes 1fa5ea69aaf9559a6cc2a2b24cdc54c322f953e9 and a4ab4e38e21364c15acdc4600aa6adbad69a7d51. The new Blake2b contract supports only output length of 32 bytes and empty keys.

Low

LOW-1	The condition about the minimum amount in xcmTransferAsset() and xcmTransferNativeAsset() is also satisfied if the assetAddressToMinimumValue[asset] has not been set yet
Contract(s)	`AstarXcmAction.sol, MoonBeamXcmAction.sol`
Status	**Resolved**

Description

All the actions have two parts: first transferring the asset to Biforst, using `xcmTransferAsset()` or `xcmTransferNativeAsset()` and then using XCM to send a message to Bifrost describing the action which should be executed. The `xcmTransferAsset()` and `xcmTranserNativeAsset()` functions check if the amount the user wants to send is greater than the minimum amount set by the owner for this asset. But if the user tries to transfer an asset and the owner has not set yet its minimum amount, this condition will be trivially satisfied and the transaction will proceed.

For `xcmTransferAsset()` the fix we suggest in INFO-1 will solve this problem, since all the functions related to non-native assets check that the `addressToCurrencyId` is not 0. For native assets, this extra check should be also added in mintNativeAsset where it is missing.

Recommendation

We suggest either implementing the INFO-1 fix or another fix that will prevent users from executing transactions before the minimum amount is set.

Alleviation

The team fixed the issue at commit hash 668190e3edbbbd17438734cb89861b8172828431, adding a check that a non-zero minimum amount has been set.

LOW-2	Anyone can initialize the contracts
Contract(s)	`AstarXcmAction.sol, MoonBeamXcmAction.sol`
Status	**Acknowledged**

Description

The contracts are upgradeable and for that reason, they use initializers and not constructors. The problem is that the initializers set the msg.sender as the owner of the contract, therefore if an attacker acts fast enough and calls the initialize function first, he will be the owner of the contracts and will be able to set important parameters and pause the contracts. Therefore the contracts should be immediately initialized after deployment. The initialization should happen through the proxy contract. But in general, it is also a good practice also to initialize the implementation contract. That way an attacker cannot become the owner of the implementation contract.

Recommendation

We suggest being really careful with initializations, initializing the contracts immediately after deployment and do not leave uninitialized not even the implementation contracts.

Alleviation

The team is going to use the OpenZeppelin Transparent Proxy pattern, therefore there is not any security issue if the implementation contract is not initialized. The team is also aware that the initialization through the proxy should be executed immediately after deployment.

Informational/Suggestions

INFO-1	`setAssetAddressToCurrencyId()` could call `setAssetAddressToMinimumValue()`
Contract(s)	`AstarXcmAction.sol, MoonBeamXcmAction.sol`
Status	**Resolved**

Description

The function setAssetAddressToCurrencyId() could also call setAssetAddressToMinimumValue(). That way all the details about the corresponding asset would be set simultaneously and some edge cases i.e. the user calls for an action involving asset A, the Id of A has been set, therefore the action will be executed, but its minimum value has not been set yet, therefore the corresponding condition will be redundant. This issue is also related to issue LOW-1.

Recommendation

We suggest adding a call of setAddressToMinimumValue in setAssetAddressToCurrencyId to avoid these edge cases.

Alleviation

The team fixed this issue at commit hash f4b8d61ab13a0e7b015736c4eb09c05129c1e76c , merging these two methods.

INFO-2	Some variables could be constants
Contract(s)	`MoonbeamXcmAction.sol`
Status	**Dismissed**

Description

The variables BNCAddress, bifrostParaId and nativeCurrencyId are set only once during initialization and they cannot be later updated, therefore they could be denoted as constants or as immutables making more gas efficient their access.

Recommendation

We suggest making these variables constants or immutable for gas efficiency.

Alleviation

The team informed us that will not fix this issue because: *"This problem has not been fixed, because we need to be compatible with the two networks of Moonriver and Moonbeam, so the information can only be determined during initialization"*.

INFO-3	`getXcmTransactorDestination() always returns` the same value, therefore it is not necessary to call it repeatedly
Contract(s)	`MoonbeamXcmAction.sol`
Status	**Resolved**

Description

This function hasn't any arguments and its output depends only on the bifrostParaId variable, which is set once upon initializing the contract, therefore the outcome of this function will always be the same

Recommendation

We suggest calling only once this function upon initialization and storing its value to a variable that can be used whenever it is needed.

Alleviation

The team implemented the fix at commit hash 79280d87799816d726fffdd4eb26ae09e41f5483.

INFO-4	`Typo`
Contract(s)	`BuildCalldata.sol`
Status	**Resolved**

Description

The third argument of the `buildSwapCallBytes()` function named `currency_in_out` should be renamed to `currency_out.`

Recommendation

We suggest fixing this typo to improve the readability of the code.

Alleviation

The team fixed the typo at commit hash e085e9fe8315e937f6ef49dd05632dad6dc6db79.

INFO-5	Redundant `super.*` keyword
Contract(s)	`AstarXcmAction.sol, MoonBeamXcmAction.sol`
Status	**Resolved**

Description

The `initialize()` function of AstarXcmAction.sol and MoonbeamxcmAction.sol calls the functions `_Ownable_init` and `_Pausable_init` of their parent contracts `OwnableUpgradeable` and `PausableUpgradeable` respectively using the keyword super.* although this is not necessary since there are no other functions with the same name in the child contracts.

Alleviation

The team fixed the issue at commit hash 3ee794471a9b3d27b9e7685fead2e8840024a4c9.

INFO-6	Non-used function concat
Contract(s)	`Blake2b.sol`
Status	**Resolved**

Description

The concat() function of the Blake2b.sol contract is not used anywhere and moreover, it is not related to the Blake2b hash function.

Recommendation

We suggest removing this function.

Alleviation

The team fixed the issue at commit hash bf1b83f3ec7e8a8dbc5fd08f15bb158abf7abc17.

INFO-7	Floating and experimental pragma
Contract(s)	`all contracts`
Status	**Partially resolved**

Description

Consider using a common fixed pragma for all contracts instead of floating pragmas as best practice.

Also, the ABI coder v2 is not considered experimental anymore, it can be selected via `pragma abicoder v2` since Solidity 0.7.4.

Recommendation

We suggest fixing a specific pragma for all contracts.

Alleviation

The team partially resolved the issue at commit hash d6c5a6faf90d1a9a95d409590d3fdfc0e437584a.

About Common Prefix

Common Prefix is a blockchain research, development, and consulting company consisting of a small number of scientists and engineers specializing in many aspects of blockchain science. We work with industry partners who are looking to advance the state-of-the-art in our field to help them analyze and design simple but rigorous protocols from first principles, with provable security in mind.

Our consulting and audits pertain to theoretical cryptographic protocol analyses as well as the pragmatic auditing of implementations in both core consensus technologies and application layer smart contracts.