CERTIK

Stafi Protocol

rBridge

Security Assessment

January 15th, 2021

[Final Report]

By:
Georgios Delkos @ CertiK
georgios.delkos@certik.org

Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Stafi Protocol - rBridge
Description	The Stafi rBridge
Platform	Substrate; Rust
Codebase	GitHub Repository
Commits	1. 1a5344a1a2ef1ad169f89be9ab987ff929040d60

Audit Summary

Delivery Date	Jan. 12, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 12, 2020 - Jan. 15, 2021

Vulnerability Summary

Total Issues	2
Total Critical	0
Total Major	0
Total Medium	1
Total Minor	0
Total Informational	1

Executive Summary

Stafi protocol requested for CertiK to perform an audit in their new rBridge swap module based on Substrate. The auditing team conducted the audit in the timeframe between January 12, 2020, and January 15, 2021, with 2 engineers.
The auditing process evaluated code implementation against provided specifications, examining language-specific issues, and ensure proper framework usage.

The system in scope is currently in beta and represents a one-way bridge. The module introduces privileged functionality for the administrator and the users of the system.

The system store is based on a simple state that handles the chain nonces, a collection of chain ids that are whitelisted from the system's administrator, and nonce values that are bumped up after every operation. The systems store also holds proxy accounts that can modify the chain fees and be added only by the systems administrator.
Finally, the system holds a fees recipient account and functionality to pause the system that are both controlled by the systems administrator.

The module single user available functionality is the transfer_native function that enables a user to swap coins from the substrate chain to the ERC20 token on the ethereum chain by paying a fee on the native chain. The user provides a recipient account on the ethereum network and the amount that he wants to swap. Currently, the system provides only a oneway functionality from native chain tokens to ethereum chain ERC20 tokens, and the arguments are frontend controlled. The checks should be expanded and updated to ensure security at the node level and not be dependant on the frontend validation. This is more important as the system will enable more chain swaps where conflicting conditions may arise due to structural similarities on inputs.

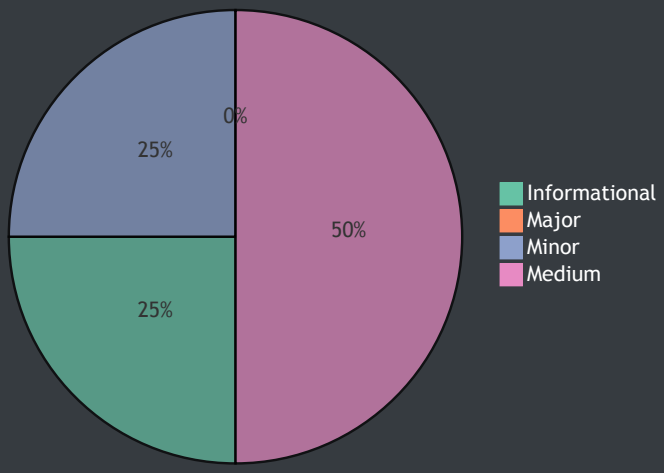
The code examined had no panicking macros usage, one unwrap_or default that is completely safe, and no redundant allocations or out-of-bounds indexing—no unhandled errors, and finally, no arithmetic problems.

To summarize, the system implementation is well constructed regarding the language and framework usage concerning best practices with no critical or major findings. The code is well written with documentation and commenting on code that helps the readability of the codebase, and the testing is extensive with sufficient edge cases provided.

Files In Scope

ID	File
CMN	node/pallets/bridge/common/src/lib.rs
SWP	node/pallets/bridge/swap/src/lib.rs

Findings



ID	Title	Type	Severity	Resolved
CMN-01	No Remove Whitelist Functionality	Implementation	Informational	ⓘ
SWP-01	Inefficient Check	Logical Issue	Medium	ⓘ

CMN-01: No Remove Whitelist Functionality.

Type	Severity	Location
Implementation	Informational	node/pallets/bridge/common/src/lib.rs L119

Description:

The code contains functionality to whitelist chain ids but does not contain functionality to remove from the whitelisting list.

```
/// Enables a chain ID as a source or destination for a bridge transfer.
///
/// # <weight>
/// - O(1) lookup and insert
/// # </weight>
#[weight = 195_000_000]
pub fn whitelist_chain(origin, id: ChainId) -> DispatchResult {
  Self::ensure_admin(origin)?;
  Self::whitelist(id)
}
```

Recommendation:

Introduce a new one function to remove whitelisted chain ids.

Alleviation:

The team has intruduced a remove whitelisted chain in commit `b495b5c4e746a89e2420a477c00ebcde1e5d27a9` .

SWP-01: Ineffiecient Check.

Type	Severity	Location
Volatile Code	● **Medium**	node/pallets/bridge/swap/src/lib.rs L65

Description:

The code checks if the request is for the ethereum chain and performs a check against the validity of the address.

The validity check just checks if the vector of bytes is of length 20.

```
    if dest_id == ETH_CHAIN_ID {
      Self::check_eth_recipient(recipient.clone())?;
    }

    ...

    impl<T: Trait> Module<T> {
        pub fn check_eth_recipient(recipient: Vec<u8>) -> DispatchResult {
            ensure!(recipient.len() == 20, Error::
    <T>::InvalidEthereumAddress);

            Ok(())
        }
    }
```

Recommendation:

Since the chain id for the moment is controlled by the front end, we can consider it safe. The issue here would be as soon as more chains are supported, and chain id is user-controlled. Cosmos addresses are also 20 bytes long, that would enable a cosmos chain id to slip through as a ethereum one with the current check.

There is validation going on on the front end on all fields, but security should not be based on this.

We do believe that the validation can also include a checksum check for the address.

Alleviation:

The team has acknowledged the issue and has already planned a fix.

Appendix

Finding Categories

Arithmetic

Arithmetic exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.