CERTIK

Stafi Protocol

Security Assessment

November 13th, 2020

For :

Stafi Bridge Solidity Contracts

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 indeed 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
Description	Solidity smart contracts to enable transfers to and from EVM compatible chains. These contracts consist of a core bridge contract (Bridge.sol) and a set of handler contracts (ERC20Handler.sol, and GenericHandler.sol). The bridge contract is responsible for initiating, voting on, and executing proposed transfers. The handlers are used by the bridge contract to interact with other existing contracts.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 169f826708b8cb9abf387f761f5456e7f5e33dd1 2. 357d641f944a0251517206b6ca5f1ccab6eb391f

Audit Summary

Delivery Date	Nov. 13, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Nov. 03, 2020 - Nov. 08 2020

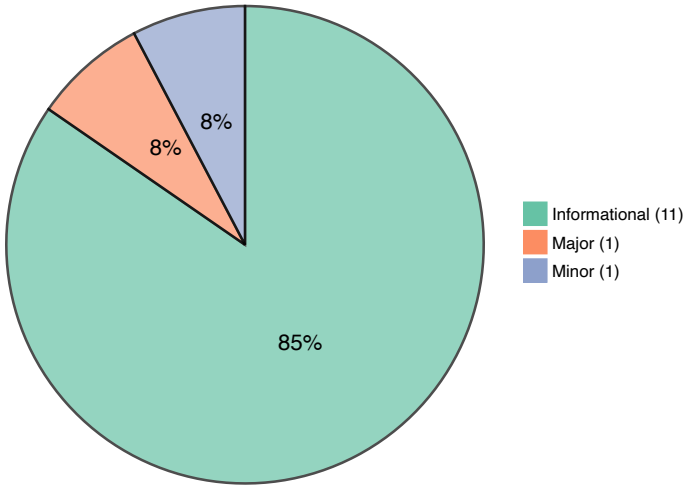
Vulnerability Summary

Total Issues	13	(13 Resolved)
Total Critical	-	
Total Major	1	(1 Resolved)
Total Minor	1	(1 Resolved)
Total Informational	11	(11 Resolved)

Executive Summary

- No deployment or development configuration or documentation was included in repository, and OpenZeppelin contracts were explicitly included in the codebase.
- The codebase was found to contain multiple contracts named `Pausable`. While no deployment or development configuration was supplied in the repository to suggest the compilation scheme, if all contracts in the codebase were compiled in a single pass, there would have been name collisions for the `Pausable` contract. Compilation may have succeed, but only the `Pausable` contract which was compiled first was chosen for the placement of both contracts, which could have resulted in undefined behavior or crashing.
- Access control was found to be properly implemented on all public and externally-visible functions within the `Bridge`, `ERC20Handler` and `HandlerHelper` contracts.
- Calling private implementation functions for a modifier is inefficient, so we recommended placing the code within each modifier directly.
- We pointed out that when iterating over an array, it is more performant to store the length of the array in a local variable than to retrieve the length over each iteration.
- The `fundERC20` function in the `ERC20Safe` contract did not implement access restriction and takes an arbitrary `owner` address parameter instead of referencing `msg.sender`.
- We noticed that proposal identifiers may have the potential to collide, as they are calculated from `depositNonce` and `chainID` `uint64` parameters which are packed into a `uint72`, but the `depositNonce` value is only shifted left by 8 bits and the `chainID` value is not clamped before performing the bitwise-OR, which makes determining a difference between the proposal identifiers impossible for `(depositNonce: 1, chainID: 512)`, `(depositNonce: 2, chainID: 256)` and `(depositNonce: 3, chainID: 0)`, among many other possible collisions. After communicating with the Stafi team about the issue, they responded with the following points:
    1. Only relayers added by admin can call the functions related to the proposal.
    2. There won't be too many chains, may be only a dozen at most.
    3. In addition to nonceAndID, there is datahash, which consists of recipientAddress and amount.
    4. A proposal has an expiration time too.
- All of the issues were resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

Findings



Prefix	File
STB	General
BRI	contracts/Bridge.sol
ERS	contracts/ERC20Safe.sol
ERH	contracts/handlers/ERC20Handler.sol

ID	Title	Type	Severity	Resolved
STB-01	Lack of deployment or development configuration	Implementation	Informational	✓
STB-02	Multiple `Pausable` contract implementations	Implementation	Minor	✓
BRI-01	Unnecessary private function `_onlyAdmin`	Implementation	Informational	✓
BRI-02	Unnecessary private function `_onlyAdminOrRelayer`	Implementation	Informational	✓
BRI-03	Unnecessary private function `_onlyRelayers`	Implementation	Informational	✓
BRI-04	Inefficient loop over `initialRelayers` memory array	Performance	Informational	✓
BRI-05	Potential proposal ID collisions in `getProposal`	Implementation	Informational	✓
BRI-06	Potential proposal ID collisions in `voteProposal`	Implementation	Informational	✓
BRI-07	Potential proposal ID collisions in `cancelProposal`	Implementation	Informational	✓
BRI-08	Potential proposal ID collisions in `executeProposal`	Implementation	Informational	✓
BRI-09	Inefficient loop over `addrs` memory array	Performance	Informational	✓
ERS-01	Arbitrary `owner` address in unrestricted `fundERC20`	Implementation	Major	✓
ERH-01	Inefficient loop over memory arrays	Performance	Informational	✓

STB-01: Lack of deployment or development configuration

Type	Severity	Location
Implementation	Informational	General

Description:

No deployment or development configuration or documentation was included in the repository, and OpenZeppelin contracts were explicitly included in the codebase.

Recommendation:

We recommended utilizing `npm` and `truffle` or `buidler`, as well as importing the official `@openzeppelin/contracts` npm module over including the contracts directly.

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

STB-02: Multiple `Pausable` contract implementations

Type	Severity	Location
Implementation	Minor	General

Description:

There were multiple implementations of contracts named `Pausable` in the project. When all of the contracts were compiled in a single pass, compilation may have succeeded, but only the `Pausable` contract which was compiled first would be chosen for the placement of both contracts, which could have resulted in undefined behavior or crashing.

Recommendation:

We recommended removing the `utils/Pausable.sol` file in favor of the `openzeppelin/Pausable.sol` file, or renaming the `Pausable` contract in the `utils/Pausable.sol` file to something unique.

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

BRI-01: Unnecessary private function `_onlyAdmin`

Type	Severity	Location
Implementation	Informational	Bridge.sol L71, L90-L92

Description:

The `onlyAdmin` modifier in the `Bridge` contract made a call to the private `_onlyAdmin` function at line 71:

```
_onlyAdmin();
```

But the private `_onlyAdmin` function was not utilized anywhere else from within the `Bridge` contract:

```
function _onlyAdmin() private view {
  require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "sender doesn't have admin role");
}
```

Recommendation:

We recommended removing the private `_onlyAdmin` function at lines 90-92 and moving the requirement from its implementation directly into the `onlyAdmin` modifier:

```
modifier onlyAdmin() {
  require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "sender doesn't have admin role");
  _;
}
```

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

BRI-02: Unnecessary private function `_onlyAdminOrRelayer`

Type	Severity	Location
Implementation	Informational	Bridge.sol L76, L85-L88

Description:

The `onlyAdminOrRelayer` modifier in the `Bridge` contract made a call to the private `_onlyAdminOrRelayer` function at line 76:

```
_onlyAdminOrRelayer();
```

But the private `_onlyAdminOrRelayer` function was not utilized anywhere else from within the `Bridge` contract:

```
function _onlyAdminOrRelayer() private view {
  require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender) || hasRole(RELAYER_ROLE, msg.sender),
    "sender is not relayer or admin");
}
```

Recommendation:

We recommended removing the private `_onlyAdminOrRelayer` function at lines 85-88 and moving the requirement from its implementation directly into the `onlyAdminOrRelayer` modifier:

```
modifier onlyAdminOrRelayer() {
  require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender) || hasRole(RELAYER_ROLE, msg.sender),
    "sender is not relayer or admin");
  _;
}
```

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

BRI-03: Unnecessary private function `_onlyRelayers`

Type	Severity	Location
Implementation	Informational	Bridge.sol L81, L94-L96

Description:

The `onlyRelayers` modifier in the `Bridge` contract made a call to the private `_onlyRelayers` function at line 81:

```
_onlyRelayers();
```

But the private `_onlyRelayers` function was not utilized anywhere else from within the `Bridge` contract:

```
function _onlyRelayers() private view {
  require(hasRole(RELAYER_ROLE, msg.sender), "sender doesn't have relayer role");
}
```

Recommendation:

We recommended removing the private `_onlyRelayers` function at lines 94-96 and moving the requirement from its implementation directly into the `onlyRelayers` modifier:

```
modifier onlyRelayers() {
  require(hasRole(RELAYER_ROLE, msg.sender), "sender doesn't have relayer role");
  _;
}
```

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

BRI-04: Inefficient loop over `initialRelayers` memory array

Type	Severity	Location
Performance	Informational	Bridge.sol L113

Description:

The `constructor` of the `Bridge` contract performed a loop over its supplied `initialRelayers` memory array while retrieving the length of the array over each iteration, which was inefficient:

```
for (uint i; i < initialRelayers.length; i++) {
```

Recommendation:

We recommended storing the length of the `initialRelayers` array in a local variable in order to save on the overall cost of gas:

```
uint256 initialRelayerCount = initialRelayers.length;

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

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

BRI-05: Potential proposal ID collisions in `getProposal`

Type	Severity	Location
Implementation	Informational	Bridge.sol L248

Description:

The `getProposal` function in the `Bridge` contract has the potential for proposal identifiers to collide, as they are calculated from `depositNonce` and `originChainID` `uint64` parameters which are packed into a `uint72`, but the `depositNonce` value is only shifted left by 8 bits and the `originChainID` value is not clamped before performing the bitwise-OR, which makes determining a difference between the proposal identifiers impossible for `(depositNonce: 1, originChainID: 512)`, `(depositNonce: 2, originChainID: 256)` and `(depositNonce: 3, originChainID: 0)`, among many other possible collisions:

```
uint72 nonceAndID = (uint72(depositNonce) << 8) | uint72(originChainID);
```

Recommendation:

We recommended either clamping the value of the `originChainID` parameter to the maximum value of a `uint8` or refactoring the proposal identifier structure to utilize a `uint128` instead of a `uint72`, then shift the `depositNonce` left by 64 in order to protect against collisions.

Alleviation:

The issue was dropped from major to informational and is considered resolved after communicating with the client to come to the following conclusions:

1. Only relayers added by admin can call the functions related to the proposal.
2. There won't be too many chains, may be only a dozen at most.
3. In addition to nonceAndID, there is datahash, which consists of recipientAddress and amount.
4. A proposal has an expiration time too.

BRI-06: Potential proposal ID collisions in `voteProposal`

Type	Severity	Location
Implementation	Informational	Bridge.sol L333

Description:

The `voteProposal` function in the `Bridge` contract has the potential for proposal identifiers to collide, as they are calculated from `depositNonce` and `chainID` `uint64` parameters which are packed into a `uint72`, but the `depositNonce` value is only shifted left by 8 bits and the `chainID` value is not clamped before performing the bitwise-OR, which makes determining a difference between the proposal identifiers impossible for `(depositNonce: 1, chainID: 512)`, `(depositNonce: 2, chainID: 256)` and `(depositNonce: 3, chainID: 0)`, among many other possible collisions:

```
uint72 nonceAndID = (uint72(depositNonce) << 8) | uint72(chainID);
```

Recommendation:

We recommended either clamping the value of the `chainID` parameter to the maximum value of a `uint8` or refactoring the proposal identifier structure to utilize a `uint128` instead of a `uint72`, then shift the `depositNonce` left by 64 in order to protect against collisions.

Alleviation:

The issue was dropped from major to informational and is considered resolved after communicating with the client to come to the following conclusions:

1. Only relayers added by admin can call the functions related to the proposal.
2. There won't be too many chains, may be only a dozen at most.
3. In addition to nonceAndID, there is datahash, which consists of recipientAddress and amount.
4. A proposal has an expiration time too.

BRI-07: Potential proposal ID collisions in `cancelProposal`

Type	Severity	Location
Implementation	Informational	Bridge.sol L391

Description:

The `cancelProposal` function in the `Bridge` contract has the potential for proposal identifiers to collide, as they are calculated from `depositNonce` and `chainID` `uint64` parameters which are packed into a `uint72`, but the `depositNonce` value is only shifted left by 8 bits and the `chainID` value is not clamped before performing the bitwise-OR, which makes determining a difference between the proposal identifiers impossible for `(depositNonce: 1, chainID: 512)`, `(depositNonce: 2, chainID: 256)` and `(depositNonce: 3, chainID: 0)`, among many other possible collisions:

```
uint72 nonceAndID = (uint72(depositNonce) << 8) | uint72(chainID);
```

Recommendation:

We recommended either clamping the value of the `chainID` parameter to the maximum value of a `uint8` or refactoring the proposal identifier structure to utilize a `uint128` instead of a `uint72`, then shift the `depositNonce` left by 64 in order to protect against collisions.

Alleviation:

The issue was dropped from major to informational and is considered resolved after communicating with the client to come to the following conclusions:

1. Only relayers added by admin can call the functions related to the proposal.
2. There won't be too many chains, may be only a dozen at most.
3. In addition to nonceAndID, there is datahash, which consists of recipientAddress and amount.
4. A proposal has an expiration time too.

BRI-08: Potential proposal ID collisions in `executeProposal`

Type	Severity	Location
Implementation	Informational	Bridge.sol L416

Description:

The `executeProposal` function in the `Bridge` contract has the potential for proposal identifiers to collide, as they are calculated from `depositNonce` and `chainID` `uint64` parameters which are packed into a `uint72`, but the `depositNonce` value is only shifted left by 8 bits and the `chainID` value is not clamped before performing the bitwise-OR, which makes determining a difference between the proposal identifiers impossible for `(depositNonce: 1, chainID: 512)`, `(depositNonce: 2, chainID: 256)` and `(depositNonce: 3, chainID: 0)`, among many other possible collisions:

```
uint72 nonceAndID = (uint72(depositNonce) << 8) | uint72(chainID);
```

Recommendation:

We recommended either clamping the value of the `chainID` parameter to the maximum value of a `uint8` or refactoring the proposal identifier structure to utilize a `uint128` instead of a `uint72`, then shift the `depositNonce` left by 64 in order to protect against collisions.

Alleviation:

The issue was dropped from major to informational and is considered resolved after communicating with the client to come to the following conclusions:

1. Only relayers added by admin can call the functions related to the proposal.
2. There won't be too many chains, may be only a dozen at most.
3. In addition to nonceAndID, there is datahash, which consists of recipientAddress and amount.
4. A proposal has an expiration time too.

BRI-09: Inefficient loop over `addrs` memory array

Type	Severity	Location
Performance	Informational	Bridge.sol L438

Description:

The `transferFunds` function in the `Bridge` contract performed a loop over its supplied `addrs` memory array while retrieving the length of the array over each iteration, which was inefficient:

```
for (uint i = 0; i < addrs.length; i++) {
```

Recommendation:

We recommended storing the length of the `initialRelayers` array in a local variable in order to save on the overall cost of gas:

```
uint256 addrCount = addrs.length;

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

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

ERS-01: Arbitrary `owner` address in unrestricted `fundERC20`

Type	Severity	Location
Implementation	Major	ERC20Safe.sol L22-L25

Description:

The `fundERC20` function in the `ERC20Safe` contract did not implement access restriction and took an arbitrary `owner` address parameter instead of referencing `msg.sender`:

```
function fundERC20(address tokenAddress, address owner, uint256 amount) public {
  IERC20 erc20 = IERC20(tokenAddress);
  _safeTransferFrom(erc20, owner, address(this), amount);
}
```

Recommendation:

We recommended determining if the `fundERC20` function should be unrestricted:

- If not, implement proper access restriction for the `funcERC20` function.
- If so, consider replacing the usage of the arbitrary `owner` address parameter with `msg.sender`

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

ERH-01: Inefficient loop over memory arrays

Type	Severity	Location
Performance	Informational	handlers/ERC20Handler.sol L44-L55

Description:

The `constructor` of the `ERC20Handler` contract performed a loops over its supplied `initialResourceIDs` and `burnableContractAddresses` memory array parameters while retrieving the length of the arrays over each iteration, which was inefficient:

```
for (uint256 i = 0; i < initialResourceIDs.length; i++) {
  _setResource(initialResourceIDs[i], initialContractAddresses[i]);
}

for (uint256 i = 0; i < burnableContractAddresses.length; i++) {
  _setBurnable(burnableContractAddresses[i]);
}
```

Recommendation:

We recommended refactoring the `constructor` of the `ERC20Handler` contract to store the length of the `initialResourceIDs` and `burnableContractAddresses` memory array parameters in local variables in order to save on the overall cost of gas:

```
uint256 initialResourceIDsLength = initialResourceIDs.length;
uint256 burnableContractAddressesLength = burnableContractAddresses.length;

require(initialResourceIDsLength == initialContractAddresses.length,
  "initialResourceIDs and initialContractAddresses len mismatch");

_bridgeAddress = bridgeAddress;

for (uint256 i = 0; i < initialResourceIDsLength; i++) {
  _setResource(initialResourceIDs[i], initialContractAddresses[i]);
}

for (uint256 i = 0; i < burnableContractAddressesLength; i++) {
  _setBurnable(burnableContractAddresses[i]);
}
```

Alleviation:

The issue was resolved with commit 357d641f944a0251517206b6ca5f1ccab6eb391f.

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation 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.