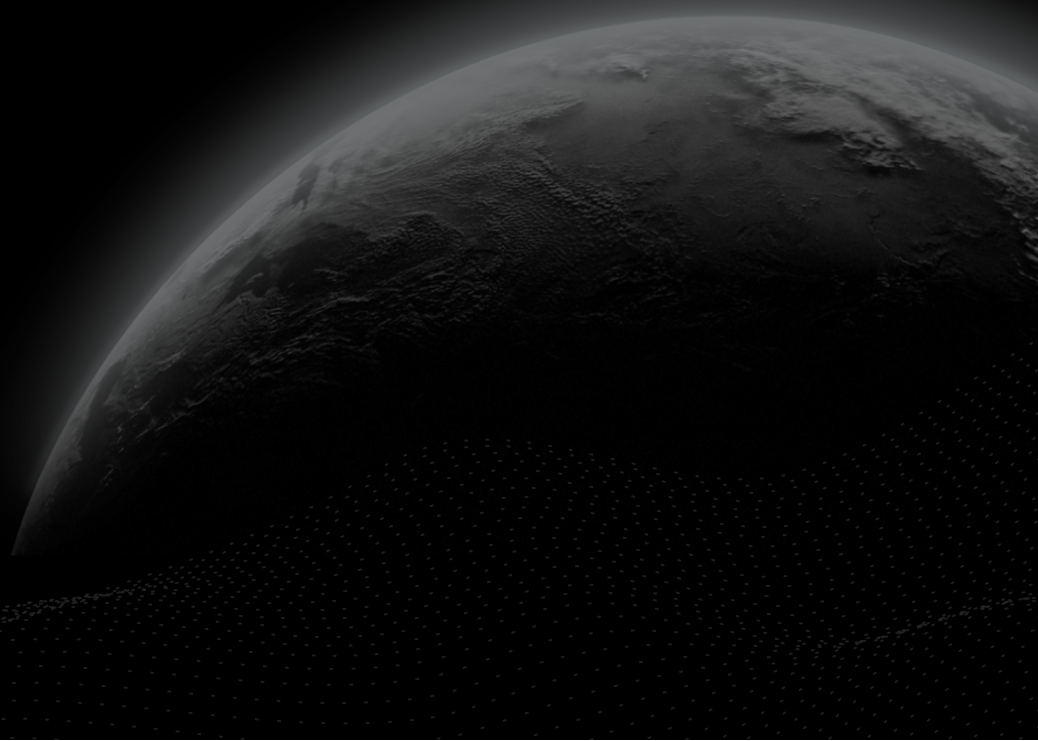
CERTIK

Security Assessment

Orakl Network

CertiK Assessed on May 11th, 2024



CertiK Assessed on May 11th, 2024

Orakl Network

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Oracle	EVM Compatible	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 05/11/2024	N/A

CODEBASE	COMMITS
update	a6f788882345cb870bbc07751e4ccc603f50d11f
base	6b33b7294a5348b17bd030d468952af4c34aa665
View All in Codebase Page	View All in Codebase Page

Vulnerability Summary

14	10	0	0	4	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

1 Critical	1 Resolved		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
7 Major	3 Resolved, 4 Acknowledged		Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
1 Medium	1 Resolved		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
4 Minor	4 Resolved		Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
1 Informational	1 Resolved		Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | ORAKL NETWORK

CODEBASE | ORAKL NETWORK

Repository

update

base

Commit

a6f788882345cb870bbc07751e4ccc603f50d11f

6b33b7294a5348b17bd030d468952af4c34aa665

AUDIT SCOPE | ORAKL NETWORK

8 files audited ● 4 files with Acknowledged findings ● 4 files without findings

ID	Repo	File	SHA256 Checksum
FBB	Bisonai/orakl	src/Feed.sol	19560836a3a9f0369ab378c17c5bb9f1d39b2dd2df596dcd485bea48b4d16937
FPB	Bisonai/orakl	src/FeedProxy.sol	9c57db923bbaae68cb4866b7b5f9d5d3ce6da2a2dc56f434631bc67d459b052b
FRB	Bisonai/orakl	src/FeedRouter.sol	edd2ad1c09389256ccd5196184575c828146eb750f45d3c6ffde4d866e3a409c
SPB	Bisonai/orakl	src/SubmissionProxy.sol	6f365dfeadd0828f4bf494610891588b389cb3773671161fa74a7e86c64441ac
IFB	Bisonai/orakl	src/interfaces/IFeed.sol	17e518fbbae11149e62d4dacb05a2dc40ac771d2ac98ba9c869f397d498c6870
IFP	Bisonai/orakl	src/interfaces/IFeedProxy.sol	d6547f87c4d28946fd90dc1b5b8ac957177dc5674aa572e7455c3e11ffcff398
IFR	Bisonai/orakl	src/interfaces/IFeedRouter.sol	fdab09b6e7c14d119178a190b47f73c50f8805897393592c0d20a9c1f586d96a
IFS	Bisonai/orakl	src/interfaces/IFeedSubmit.sol	200f2621fbb27647b07468b544fd50713041629bc61d4c8a7604371d37033891

APPROACH & METHODS | ORAKL NETWORK

This report has been prepared for Orakl to discover issues and vulnerabilities in the source code of the Orakl Network project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | ORAKL NETWORK

14	1	7	1	4	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Orakl Network. Through this audit, we have uncovered 14 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
SPB-02	Feed Is Not Included In The Signed Message	Volatile Code	Critical	Resolved
FBB-01	**Centralization Risks In Feed.Sol**	**Centralization**	**Major**	**Acknowledged**
FPB-01	**Centralization Risks In FeedProxy.Sol**	**Centralization**	**Major**	**Acknowledged**
FRB-02	**Centralization Risks In FeedRouter.Sol**	**Centralization**	**Major**	**Acknowledged**
SPB-03	**Centralization Risks In SubmissionProxy.Sol**	**Centralization**	**Major**	**Acknowledged**
SPB-04	`addOracle()` Assumes Not More Than 255 Oracles	Volatile Code	Major	Resolved
SPB-05	Oracle Indexes Can Duplicate	Volatile Code	Major	Resolved
SPB-06	`updateOracle()` Doesn't Set `index`	Volatile Code	Major	Resolved
SPU-02	`updateFeed()` Doesn't Update `feedAddresses`	Volatile Code	Medium	Resolved
FRB-03	`FeedRouter.twap()` Doesn't Check If `validFeed`	Inconsistency	Minor	Resolved
SPB-07	Inconsistent `setMaxSubmission()` Bounds	Inconsistency	Minor	Resolved

ID	Title	Category	Severity	Status
SPB-08	`removeOracle()` Doesn't Check If Oracle Exists	Volatile Code	Minor	Resolved
SPB-09	Lack Of Sanity Checks In `validateProof()`	Volatile Code	Minor	Resolved
GIT-01	Inaccurate Comments	Coding Issue	Informational	Resolved

SPB-02 | FEED IS NOT INCLUDED IN THE SIGNED MESSAGE

Category	Severity	Location	Status
Volatile Code	Critical	src/SubmissionProxy.sol (base): 276	Resolved

Description

```
276             bytes32 message_ = keccak256(abi.encodePacked(_answers[i],
_timestamps[i]));
277             if (validateProof(_feeds[i], message_, proofs_)) {
278                 IFeed(_feeds[i]).submit(_answers[i]);
279             }
```

The signed message in `submit()` includes `_answers[i]` and `_timestamps[i]`, `_feeds[i]` is not included. As a result, the attacker can use a valid proof set to submit the answers to a wrong feed. For example, the price of ETH-USDT to a feed BTC-USDT.

The signed message also doesn't include the destination address, the chain ID, and many other fields recommended by EIP-712: Typed structured data hashing and signing. As a result, the answer/timestamp signed in the testnet can be submitted in the mainnet, or the prices signed for one `SubmissionProxy` can be submitted to another.

`SubmissionProxy.submit()` also calls a `IFeed(_feeds[i]).submit()` where `_feeds[i]` is provided by the user. Calling the user-provided address can be dangerous.

Scenario

1. 3 Oracles prepare 3 valid proofs of the message consisting of the current ETH-USDT price (3000$) and the current timestamp.
2. The Oracle calls `SubmissionProxy.submit([ETH-USDT feed], [3000$], [now], [3 Oracle proofs])`
3. The attacker eavesdrops the submitted transaction in the mempool.
4. The attacker calls `SubmissionProxy.submit([BTC-USDT feed], [3000$], [now], [3 Oracle proofs])` with a higher gas price.
5. The attacker transaction is valid and `IFeed(BTC-USDT feed).submit(3000$)` will be executed and accepted by the feed.
6. The attacker gets benefits from the invalid price on the feed.

Recommendation

We recommend following the EIP-712 standard and remembering the submitted messages to avoid the replay attack.

Alleviation

Proofs are generated when price data is aggregated, independent of a specific chain or feed contract addresses, and are intended for use across multiple submission proxy contracts and chains.

The implemented fix is described here.

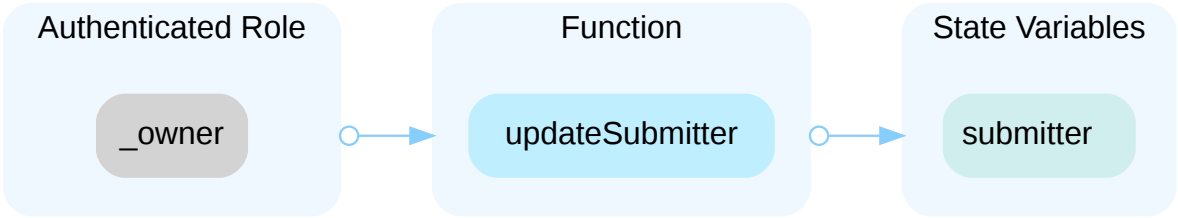
[CertiK]: After the fix the signed by Oracle message contains `keccak256(abi.encodePacked(answer, timestamp, feedHashes))` . For example, `3000$, 1715277134, BTC-USD` . The Oracle should be careful and never sign inaccurate data (for test purposes or similar).

FBB-01 | CENTRALIZATION RISKS IN FEED.SOL

Category	Severity	Location	Status
Centralization	**Major**	**src/Feed.sol (base): 68, 84**	**Acknowledged**

Description

In the contract `Feed` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and `updateSubmitter()`. Then `submit()` any `answer`.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

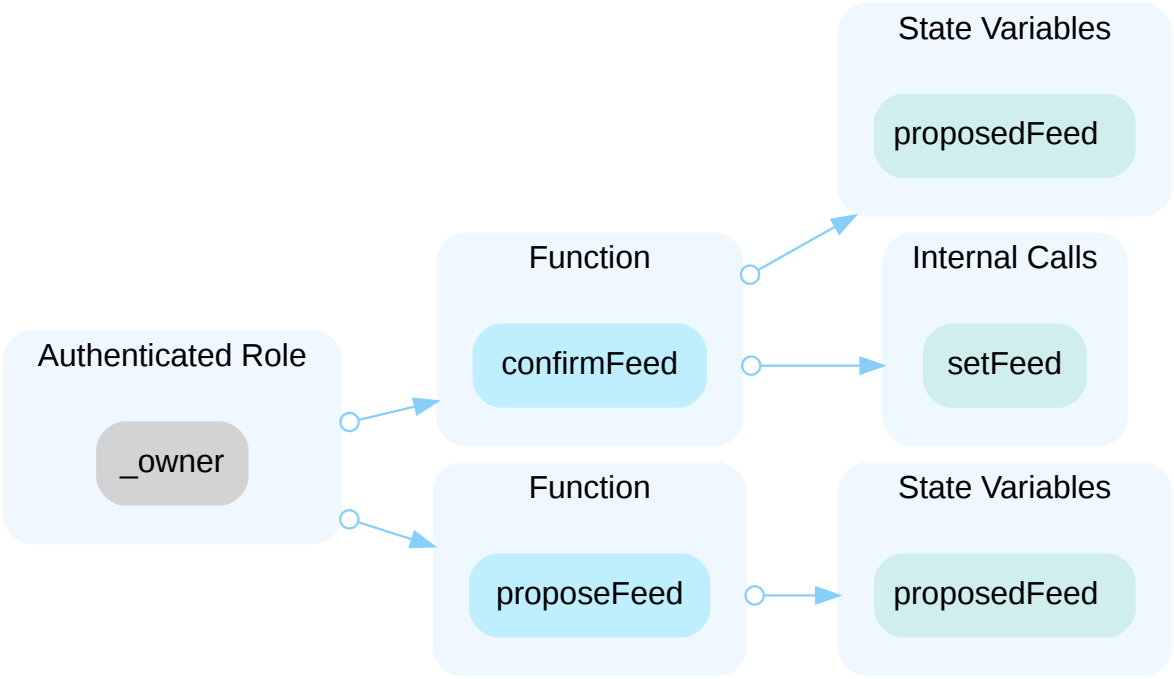
The project team position can be found here. Currently, there are no plans to implement multisig, but the direction of utilizing a multisig wallet is deemed acceptable and may be utilized within this year.

FPB-01 | CENTRALIZATION RISKS IN FEEDPROXY.SOL

Category	Severity	Location	Status
Centralization	**Major**	**src/FeedProxy.sol (base): 161, 177**	**Acknowledged**

Description

In the contract `FeedProxy` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and assign any underlying feed via `proposeFeed()` / `confirmFeed()` with any behavior.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

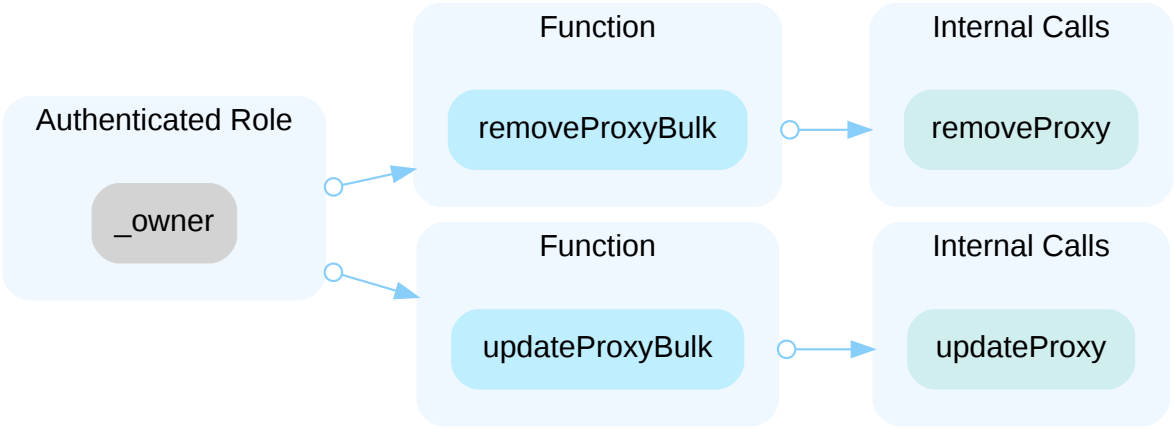
The project team position can be found here. Currently, there are no plans to implement multisig, but the direction of utilizing a multisig wallet is deemed acceptable and may be utilized within this year.

FRB-02 | CENTRALIZATION RISKS IN FEEDROUTER.SOL

Category	Severity	Location	Status
Centralization	**Major**	**src/FeedRouter.sol (base): 46, 59**	**Acknowledged**

Description

In the contract `FeedRouter` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set any feed to any `feedName`.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

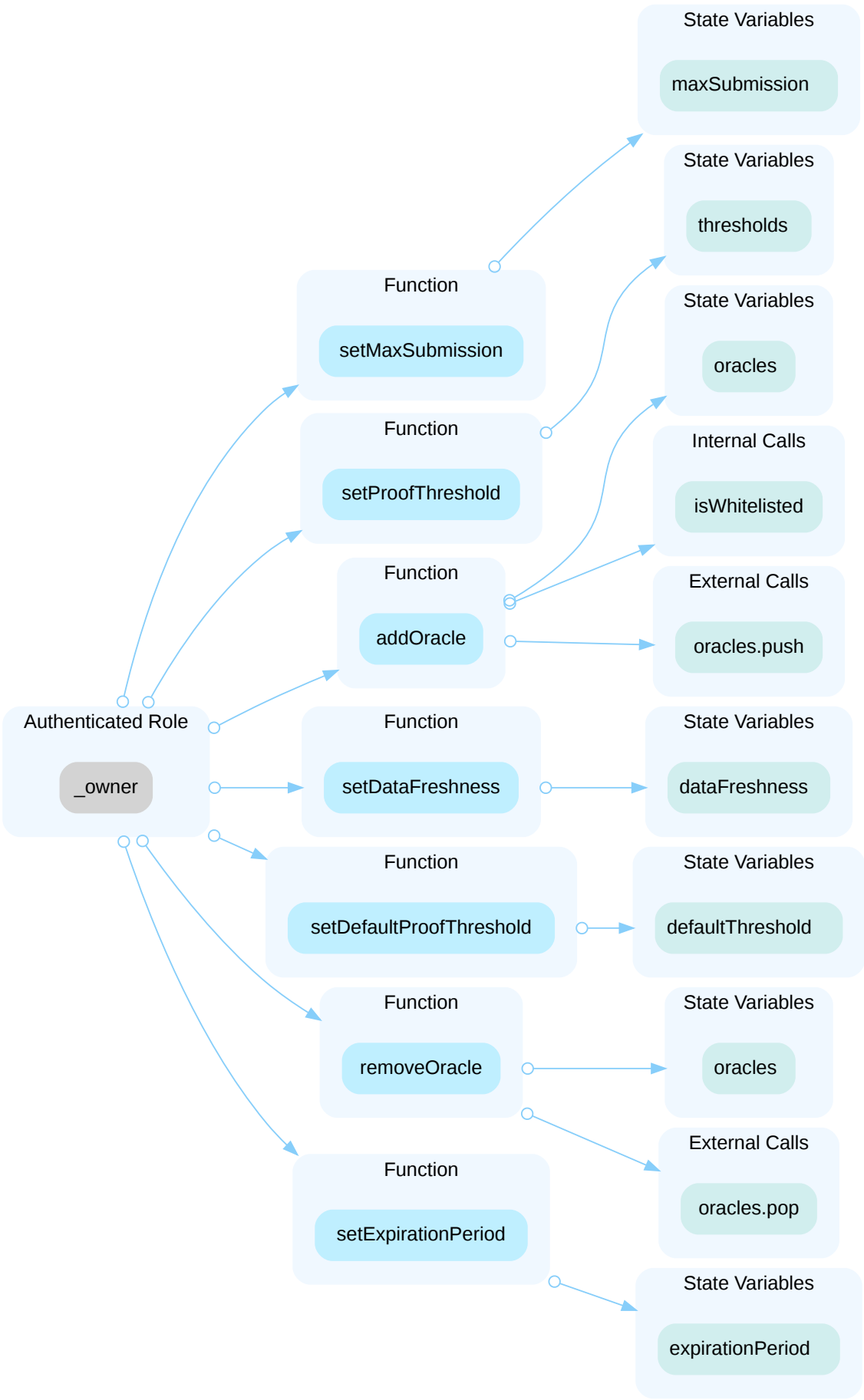
The project team position can be found here. Currently, there are no plans to implement multisig, but the direction of utilizing a multisig wallet is deemed acceptable and may be utilized within this year.

SPB-03 | CENTRALIZATION RISKS IN SUBMISSIONPROXY.SOL

Category	Severity	Location	Status
Centralization	**Major**	**src/SubmissionProxy.sol (base): 74, 86, 95, 109, 125, 146, 189, 212**	**Acknowledged**

Description

In the contract `SubmissionProxy` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and add/remove oracles and influence the feed output.


State Variables
maxSubmission
State Variables
thresholds
Function
setMaxSubmission
State Variables
oracles
Function
setProofThreshold
Internal Calls
isWhitelisted
Function
addOracle
External Calls
oracles.push
Authenticated Role
_owner
Function
setDataFreshness
State Variables
dataFreshness
Function
setDefaultProofThreshold
State Variables
defaultThreshold
Function
removeOracle
State Variables
oracles
External Calls
oracles.pop
Function
setExpirationPeriod
State Variables
expirationPeriod

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

The project team position can be found here. Currently, there are no plans to implement multisig, but the direction of utilizing a multisig wallet is deemed acceptable and may be utilized within this year.

SPB-04 | `addOracle()` ASSUMES NOT MORE THAN 255 ORACLES

Category	Severity	Location	Status
Volatile Code	Major	src/SubmissionProxy.sol (base): 155~157	Resolved

Description

```
155     uint8 oraclesLength_ = uint8(oracles.length);
156     for (uint8 i = 0; i < oraclesLength_; i++) {
```

`SubmissionProxy.addOracle()` converts `oracles.length` to `uint8`. It looks for expired `oracles` only in the first 255 elements. If not found, it adds a new element but assigns `index` converted to `uint8`.

`removeOracle()` and `updateOracle()` use `uint256` to enumerate the array. `OracleInfo.index` has `uint8` type.

Scenario

1. `onlyOwner` calls `addOracle()` with different arguments 300 times.
2. 300 elements will be added to `oracles` array.
3. The last element will get `info.index = 43`.
4. That prevents that oracle from a proof generation for `submit()` since the proofs should have different indexes in ascending order.

Recommendation

We recommend using `uint256` as an index and during enumeration, or explicitly checking that `oracles.length` is bounded and preventing new oracles adding.

SPB-05 | ORACLE INDEXES CAN DUPLICATE

Category	Severity	Location	Status
Volatile Code	Major	src/SubmissionProxy.sol (base): 192	Resolved

Description

`removeOracle()` moves the oracles in `oracles` array but doesn't update the indexes. As a result different oracles can have duplicating indexes. This breaks the proofs submission via `submit()`.

Scenario

1. `owner` calls `addOracle(A)`. `oracles[0]` is assigned `A`, `whitelist[A].index` is assigned 0.
2. `owner` calls `addOracle(B)`. `oracles[1]` is assigned `B`, `whitelist[B].index` is assigned 1.
3. `owner` calls `removeOracle(A)`. `oracles[0]` is assigned the last array element - `B`, `oracles[1]` is popped from the array, `whitelist[A].index` is assigned 0, `A` is marked as expired.
4. `B` is now stored at `oracles[0]` but has index 1.
5. `owner` calls `addOracle(C)`. `oracles[1]` is assigned `C`, `whitelist[C].index` is assigned 1.
6. As a result, `B` and `C` both have index 1 and their proofs can't be submitted in the same submission.

Recommendation

We recommend updating the indexes if the elements are moved inside `oracles` array.

SPB-06 | `updateOracle()` DOESN'T SET `index`

Category	Severity	Location	Status
Volatile Code	Major	src/SubmissionProxy.sol (base): 232	Resolved

Description

`updateOracle()` deactivates the old oracle:

```
219         whitelist[msg.sender].expirationTime = block.timestamp;
```

However, its `index` is not set to 0, like in `removeOracle()`.

The new Oracle index `info.index` is not set. As a result, the updated oracle will not be able to submit their proofs via `submit()`.

`addOracle()` also doesn't update the index of replaced expired Oracle.

Recommendation

We recommend updating the `index` in all cases.

SPU-02 | `updateFeed()` DOESN'T UPDATE `feedAddresses`

Category	Severity	Location	Status
Volatile Code	Medium	src/SubmissionProxy.sol (update1): 119	Resolved

Description

```
116     function updateFeed(bytes32 _feedHash, address _feed) public onlyOwner {
117         IFeed feed = IFeed(_feed);
118         if (address(feeds[_feedHash]) == address(0)) {
119             feedAddresses.push(feed);
120         }
121
122         feeds[_feedHash] = feed;
123         emit FeedAddressUpdated(_feedHash, _feed);
124     }
```

`feedAddresses` array contains all the feed addresses available in `feeds` mapping.

If `feeds[_feedHash]` is non-zero, the `updateFeed()` function updates it. However, the `feedAddresses` still contains the old value for the specified `_feedHash`.

Scenario

1. The owner calls `updateFeed(BTC-USD, 0xBEEF)`. `feeds[BTC-USD]` is assigned `0xBEEF`, `feedAddresses` gets one element `0xBEEF`.
2. The owner calls `updateFeed(BTC-USD, 0xDEAD)`. `feeds[BTC-USD]` is assigned `0xDEAD`, `feedAddresses` **is not updated** and contains `0xBEEF`.
3. `getFeeds()` returns and array with `0xBEEF` element.

Recommendation

We recommend updating the `feedAddresses` in `updateFeed()`.

FRB-03 | `FeedRouter.twap()` DOESN'T CHECK IF `validFeed`

Category	Severity	Location	Status
Inconsistency	Minor	src/FeedRouter.sol (base): 96, 107	Resolved

Description

`twap()` and `twapFromProposedFeed()` don't check if `validFeed`. As a result, the functions will not revert with the expected `FeedNotSetInRouter()` error.

Recommendation

We recommend adding `validFeed` modifier to revert with the expected error.

SPB-07 | INCONSISTENT `setMaxSubmission()` BOUNDS

Category	Severity	Location	Status
Inconsistency	Minor	src/SubmissionProxy.sol (base): 75	Resolved

Description

```
75          if (_maxSubmission == MIN_SUBMISSION || _maxSubmission > MAX_SUBMISSION)
 {
76              revert InvalidMaxSubmission();
```

The `SubmissionProxy.setMaxSubmission()` reverts if `_maxSubmission` is strictly bigger than `MAX_SUBMISSION` or is equal to `MIN_SUBMISSION`. To make the function behavior consistent with `setExpirationPeriod()` and `setDefaultProofThreshold()` it is reasonable to revert if `_maxSubmission < MIN_SUBMISSION` and setting `MIN_SUBMISSION = 1`.

Recommendation

We recommend using the strict comparison in all 3 functions.

SPB-08 | `removeOracle()` DOESN'T CHECK IF ORACLE EXISTS

Category	Severity	Location	Status
Volatile Code	Minor	src/SubmissionProxy.sol (base): 200	Resolved

Description

The `owner` can call `removeOracle()` with unexisting `_oracle` argument. As a result, `whitelist[_oracle].expirationTime` will be set to current time.

Recommendation

We recommend explicitly checking if oracle exists.

SPB-09 | LACK OF SANITY CHECKS IN `validateProof()`

Category	Severity	Location	Status
Volatile Code	Minor	src/SubmissionProxy.sol (base): 411	Resolved

Description

`validateProof()` contains a volatile code that can't be exploited by different reasons:

- `validateProof()` is not protected from signature malleability.
- `quorum()` returns 0 if `oracles` array is empty, so 0 valid signatures are required for the transaction to be verified.
- `ecrecover()` returns 0 if the signature is invalid, however, `validateProof()` doesn't check that - `whitelist[signer_].index` gives a valid zero index.

Recommendation

We recommend adding sanity checks to straighten the code.

GIT-01 | INACCURATE COMMENTS

Category	Severity	Location	Status
Coding Issue	Informational	src/SubmissionProxy.sol (base): 293~296; src/SubmissionProxy.sol (update1): 486~487	Resolved

Description

```
293       * @dev The function intentionally does not test whether the
294       * @param _data The bytes to be split
295       * @return proofs_ The split bytes
296       * @return success_ `true` if the split was successful, `false`
```

Some comments are inaccurate or outdated.

Recommendation

We recommend updating the comments.

OPTIMIZATIONS | ORAKL NETWORK

ID	Title	Category	Severity	Status
FRB-01	`updateProxy()` Enumerates All `feedNames`	Gas Optimization	Optimization	Resolved
SPB-01	Arguments Should Be `calldata`	Gas Optimization	Optimization	Resolved
SPU-01	Redundant Code	Code Optimization	Optimization	Resolved

FRB-01 | `updateProxy()` ENUMERATES ALL `feedNames`

Category	Severity	Location	Status
Gas Optimization	Optimization	src/FeedRouter.sol (base): 200	Resolved

Description

`FeeRouter.updateProxy()` enumerates all the `feedNames` array to check if `_feedName` is new or not. However, the check `feedToProxies[_feedName] == address(0)` can be used instead.

`feedToProxies[_feedName]` is non-zero if and only if `feedNames` array contains `_feedName`.

Recommendation

We recommend simplifying the code to save gas.

SPB-01 | ARGUMENTS SHOULD BE `calldata`

Category	Severity	Location	Status
Gas Optimization	Optimization	src/SubmissionProxy.sol (base): 251	Resolved

Description

Non changed arguments of external functions are declared as `memory` .

Recommendation

We recommend declaring the non changed arguments of external functions as `calldata` to save gas.

SPU-01 | REDUNDANT CODE

Category	Severity	Location	Status
Code Optimization	Optimization	src/SubmissionProxy.sol (update1): 81~88, 117~118	Resolved

Description

```
 81      function getFeeds() external view returns (address[] memory) {
 82          address[] memory feedAddressesArray = new address[](feedAddresses.
length);
 83          for (uint256 i = 0; i < feedAddresses.length; i++) {
 84              feedAddressesArray[i] = address(feedAddresses[i]);
 85          }
 86          return feedAddressesArray;
 87      }
```

`feedAddresses` can be returned directly, like:

```
 81      function getFeeds() external view returns (IFeed[] memory) {
 82          return feedAddresses;
 83      }
```

```
116      function updateFeed(bytes32 _feedHash, address _feed) public onlyOwner {
117          IFeed feed = IFeed(_feed);
```

`feed` variable is redundant, `_feed` argument can be `IFeed`, like:

```
116      function updateFeed(bytes32 _feedHash, IFeed feed) public onlyOwner {
```

Recommendation

We recommend removing of redundant code.

FORMAL VERIFICATION | ORAKL NETWORK

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of Standard Ownable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,
- function `transferOwnership` that transfers the ownership to a new owner.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
ownable-renounceownership-correct	Ownership is Removed.
ownable-transferownership-correct	Ownership is Transferred.
ownable-renounce-ownership-is-permanent	Once Renounced, Ownership Cannot be Regained
ownable-owner-succeed-normal	`owner` Always Succeeds

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract FeedProxy (contracts/v0.2/src/FeedProxy.sol) In Commit 6b33b7294a5348b17bd030d468952af4c34aa665

Verification of Standard Ownable Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results For Contract FeedRouter (contracts/v0.2/src/FeedRouter.sol) In Commit 6b33b7294a5348b17bd030d468952af4c34aa665

Verification of Standard Ownable Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounce-ownership-is-permanent	True	
ownable-renounceownership-correct	True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results For Contract Feed (contracts/v0.2/src/Feed.sol) In Commit 6b33b7294a5348b17bd030d468952af4c34aa665

Verification of Standard Ownable Properties

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract SubmissionProxy (contracts/v0.2/src/SubmissionProxy.sol) In Commit 6b33b7294a5348b17bd030d468952af4c34aa665

Verification of Standard Ownable Properties

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	Inconclusive	

APPENDIX | ORAKL NETWORK

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings 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.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well

as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed Ownable Properties

Properties related to function `renounceOwnership`

ownable-renounce-ownership-is-permanent

The contract must prohibit regaining of ownership once it has been renounced.

Specification:

```
constraint \old(owner()) == address(0) ==> owner() == address(0);
```

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to address(0).

Specification:

```
ensures this.owner() == address(0);
```

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

Properties related to function `owner`

ownable-owner-succeed-normal

Function `owner` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.