BLOCKSEC

Security Audit

Report for agg-oracle

Date: July 10, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Rhea Finance
Target	agg-oracle

Version History

Version	Date	Description
1.0	July 10, 2026	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-07-10 11:36:10+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Rust
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of agg-oracle of Rhea Finance.

Agg-oracle is a NEAR-based aggregated oracle contract that normalizes multiple price reporting paths into a unified latest-price storage model. It supports project-managed worker reports, Pyth Pro/Lazer signed payloads, and Chainlink Data Streams reports, with DAO-controlled configuration for price sources, token mappings, trusted signers, DON configs, TEE workers, and approved codehashes. The contract stores per token, per source prices and exposes Pyth compatible query methods that aggregate valid source prices through a median based price window, enabling downstream protocols to consume multi-source oracle prices through a familiar interface.

Note this audit only focuses on the smart contracts in the following directories/files:

- contracts/agg_oracle/src

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
agg-oracle	`Version 1`	`61712c4616ed01c751f4d7c714d329216f2e424c`
	`Version 2`	`bb30a7986182429ec304ce0ba79028279e6b5fb5`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

[1] https://github.com/rhea-finance/agg-oracle

Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism

* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **four** potential security issues. Besides, we have **two** notes.

- Medium Risk: 3
- Low Risk: 1
- Note: 2

ID	Severity	Description	Category	Status
1	Medium	Lack of binding between TEE codehash and quote	Security Issue	Fixed
2	Medium	Unapproved images can be accepted due to partial `codehash` allowlist matching	Security Issue	Fixed
3	Medium	Lack of codehash whitelist status check in price reporting services	Security Issue	Fixed
4	Low	Incorrect `conf` calculation may understate price risk	Security Issue	Fixed
5	-	Potential centralization risks	Note	-
6	-	Operational risk from off-chain TEE deployment	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of binding between TEE codehash and quote

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `tee.rs` contract, the function `verify_codehash()` is used to verify that the `RTMR3` measurement result in the TEE quote is consistent with the compose information in `tcb_info`. It extracts the image digest from the `docker_compose_file` for `codehash` whitelist `validation`. However, the function retrieves the first `compose-hash` event without ensuring that this event belongs to `imr == 3`, nor does it ensure that the event was actually included in the measurement replay performed by `replay_rtmr(event_log, 3)`.

As a result, an attacker can inject a forged `compose-hash` event at the beginning of the `tcb_info.event_log` that is not included in the `RTMR3` replay process, while still associating it with an `app_compose` containing an approved codehash. At the same time, the genuine `imr == 3` event is preserved, allowing `replayed_rtmr3 == quote.rt_mr3` to remain valid.

```
24    pub fn verify_codehash(raw_tcb_info: String, rtmr3: String) -> Vec<String> {
25        let tcb_info: Value = near_sdk::serde_json::from_str(&raw_tcb_info).expect("TCB Info should
              be valid JSON");
26        let event_log = tcb_info["event_log"].as_array().unwrap();
27
28        // Get compose hash from events
```

```
29      let expected_compose_hash = event_log.iter().find(|e| e["event"].as_str().unwrap() == "
            compose-hash").unwrap()["digest"].as_str().unwrap();
30
31      // Replay the rtmr3 and compose hash
32      let replayed_rtmr3 = replay_rtmr(event_log.to_owned(), 3);
33      let app_compose = tcb_info["app_compose"].as_str().unwrap();
34      let replayed_compose_hash: String = replay_app_compose(app_compose);
35
36      require!(replayed_compose_hash == expected_compose_hash, "Invalid compose hash");
37      require!(replayed_rtmr3 == rtmr3, "Invalid rtmr3");
38
39      let app_compose_json: Value = near_sdk::serde_json::from_str(app_compose).expect("
            app_compose should be valid JSON");
40      let docker_compose_file = app_compose_json["docker_compose_file"].as_str().expect("
            docker_compose_file should be a string");
41
42      extract_codehashes(docker_compose_file)
43  }
```

Listing 2.1: contracts/agg_oracle/src/collateral.rs

Impact An attacker may cause the contract to accept an `app_compose` that is not actually bound to the quote measurement result, thereby bypassing the codehash whitelist.

Suggestion Revise the logic to ensure that when selecting the `compose-hash` event, the event must bt bound `imr == 3`.

2.1.2 Unapproved images can be accepted due to partial `codehash` allowlist matching

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `require_approved_codehash()` is used during worker registration to validate `codehashes` extracted from the TEE `app_compose`. It receives the extracted `codehash` list and compares it against the protocol allowlist. Instead of requiring every relevant `codehash` to be approved, it only checks whether at least one extracted codehash appears in the allowlist.

As a result, an `app_compose` with multiple images can pass validation when one image is approved, even if other images are not, which is incorrect.

```
117  fn require_approved_codehash(&self, codehashes: &[String]) -> String {
118      self.approved_codehashes.iter().find(|codehash| codehashes.contains(*codehash)).cloned().
             expect("Invalid code hash")
119  }
```

Listing 2.2: contracts/agg_oracle/src/api/tee.rs

Impact A worker may be registered with unapproved images in its TEE composition.

Suggestion Require all relevant `codehashes` extracted from `app_compose` to be approved before registering the worker.

2.1.3 Lack of codehash whitelist status check in price reporting services

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `tee.rs` contract, the `revoke_codehash()` function is used to remove an approved image digest from `approved_codehashes`, in order to prevent newly registered workers from using the revoked codehash. However, the price reporting entrypoints, including the functions `report_prices()`, `report_pyth_pro_prices()`, and `report_chainlink_prices()`, rely solely on the `require_approved_worker()` function for authorization. This function does not revalidate whether the worker's original registration codehash remains present in `approved_codehashes`.

```
2   "must": {
3       "source_type": "git",
```

Listing 2.3: .DeepAudit/info.jsonc

```
24  pub fn revoke_codehash(&mut self, codehash: String) {
25      assert_one_yocto();
26      require!(self.approved_codehashes.remove(&codehash), "Not exist");
27      Events::CodehashRevoked { codehash }.emit();
28  }
```

Listing 2.4: contracts/agg_oracle/src/api/tee.rs

Impact An attacker may introduce additional unapproved components into the measured environment.

Suggestion Revise the logic to ensure that all image digests in the compose file are included in the whitelist.

2.1.4 Incorrect `conf` calculation may understate price risk

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The Pyth-compatible price response uses `conf` to represent price uncertainty, where a larger value indicates higher risk for external consumers. However, the function `price_window_conf()` calculates `lower_distance` and `upper_distance` from the selected price to the valid price window boundaries, then returns the smaller of the two distances. This selects the smaller side of the deviation range instead of reflecting the wider uncertainty of the price window. As a result, external integrations that rely on `conf` may receive an understated risk value.

```
130 fn price_window_conf(price: i64, prices: &[Price]) -> Option<u64> {
131     let min_price = prices.first()?.price.0;
132     let max_price = prices.last()?.price.0;
133     let lower_distance = price - min_price;
134     let upper_distance = max_price - price;
```

```
135         u64::try_from(lower_distance.min(upper_distance)).ok()
136     }
```

Listing 2.5: contracts/agg_oracle/src/api/pyth_compat.rs

Impact External consumers may underestimate the uncertainty of the returned price.

Suggestion Calculate `conf` using the larger relevant deviation.

2.2 Note

2.2.1 Potential centralization risks

Introduced by `Version 1`

Description In this project, privileged roles such as the `DAO` and upgrade related roles can perform sensitive oracle management operations. For example, they can register or remove price sources, configure token price mappings, approve or revoke TEE `codehashes`, manage the worker whitelist, configure trusted Pyth Pro signers, and insert or deactivate Chainlink `DON` configurations. These permissions directly affect which workers can report prices, which oracle sources are trusted, and how token prices are resolved. If the private keys of privileged accounts are lost, compromised, or maliciously used, oracle configuration may be altered in a way that affects the integrity of reported prices and downstream protocols relying on them.

2.2.2 Operational risk from off-chain TEE deployment

Introduced by `Version 1`

Description This project relies on off-chain TEE workers to generate attestations and report oracle prices. The security guarantees of the on-chain verification flow depend not only on the contract logic, but also on how the TEE image is built, deployed, measured, and operated. TEE workers should be launched strictly according to the official release and deployment process, using approved images, expected runtime configuration, and verified attestation materials. Any deviation in the off-chain build or deployment process may weaken the intended trust assumptions and affect the reliability of the reported prices.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS