BLOCKSEC

Security Audit Report for Agg-Oracle

Date: August 18, 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	August 18, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-08-18 09:14:44+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.

The reviewed update extends the protocol's oracle infrastructure by integrating Atlas and Outlayer as dedicated price-data sources. The changes introduce source-specific payload decoding and validation, signature verification, authorized-signer configuration, price normalization, price-reporting interfaces, and token-to-feed mappings. The contract's storage layout and migration procedures have also been updated to accommodate the new signer-related state while maintaining compatibility with previously stored oracle data.

Note this audit focuses on the codes located 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	a99b93c6c8ff954b4473fd020f74051d3b64bdf0
	Version 2	ba866eb500af7c4c5c7599a83c503a479432ca50

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. 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.

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

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 have **one** recommendation and **two** notes.

- Recommendation: 1
- Note: 2

ID	Severity	Description	Category	Status
1	-	Add duplicate `feed_id` checks in function `parse_and_recover_atlas_message()`	Recommendation	Fixed
2	-	Operational risk from off-chain TEE life-cycle	Note	-
3	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Recommendation

2.1.1 Add duplicate `feed_id` checks in function `parse_and_recover_atlas_message()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `parse_and_recover_atlas_message()` parses the signed Atlas payload into fixed size packages and extracts a `feed_id`, `price`, and `timestamp` from each package. The parsed results are stored in a HashMap keyed by `feed_id`. The insertion currently uses `or_insert`, so if a payload with repeated `feed_id` entries were ever provided, the first package for that `feed_id` would be retained and later entries would be ignored. Although Atlas is expected to prevent duplicate feed packages upstream, adding a local duplicate check would make this invariant explicit in the contract and avoid relying solely on the external payload producer.

```
for package in payload[..packages_end].chunks_exact(PACKAGE_LEN) {
    let feed_id = u32::from_be_bytes(package[..FEED_ID_LEN].try_into().map_err(|_| AtlasError::
        InvalidPayloadLength)?).to_string();
    let price = read_be_u128(&package[FEED_ID_LEN..FEED_ID_LEN + VALUE_LEN]);
    let timestamp = read_be_u64(&package[FEED_ID_LEN + VALUE_LEN..]);
    prices.entry(feed_id).or_insert(AtlasPriceUpdate { price, timestamp });
}
```

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

Suggestion Add duplicate `feed_id` checks in function `parse_and_recover_atlas_message()`.

2.2 Note

2.2.1 Operational risk from off-chain TEE lifecycle

Introduced by `Version 1`

Description This protocol relies on off-chain TEE workers to generate attestations and submit oracle prices. While the on-chain contracts verify TEE attestations and enforce the configured trust model, they cannot validate the integrity of the entire off-chain TEE lifecycle, including image build, release, deployment, runtime configuration, and operational management.

Consequently, the overall security of the oracle network depends not only on the correctness of the on-chain verification logic, but also on secure operational procedures for the TEE infrastructure. Weaknesses in the off-chain deployment or operational process may compromise the intended trust assumptions and ultimately affect the integrity of reported prices.

2.2.2 Potential centralization risks

Introduced by `Version 1`

Description In this project, the DAO is responsible for sensitive oracle management operations. For example, it can register or remove price sources, configure token price mappings, approve or revoke TEE `codehashes`, manage the worker whitelist, configure trusted Pyth Pro, Outlayer, and Atlas signers, and insert or deactivate Chainlink DON configurations. These permissions directly determine which workers, signers, and oracle sources are trusted and how token prices are resolved. Therefore, the security of the protocol depends on a secure governance process. Governance proposals affecting these privileged operations should follow appropriate review, authorization, and execution procedures to reduce the risk of unintended or malicious configuration changes.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS