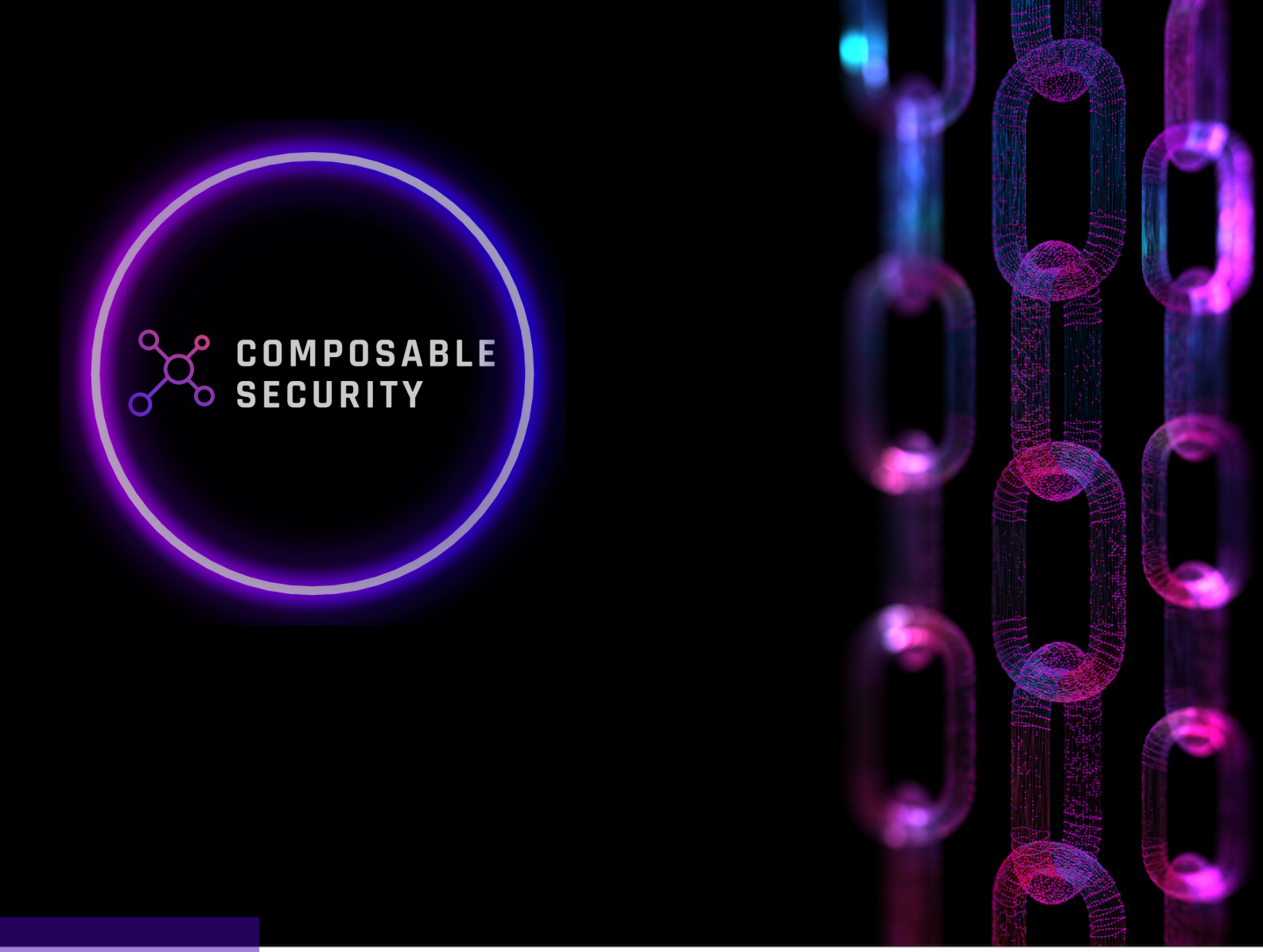


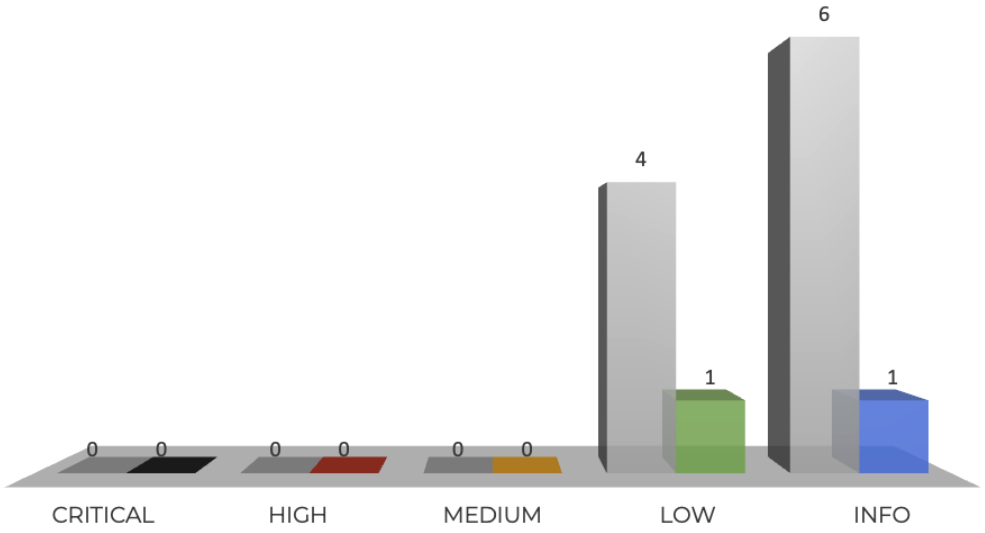
REPORT

AVS security review for RedStone

Prepared by: Composable Security
Report ID: RS-747786a5
Test time period: 2024-10-03 - 2024-10-11
Retest time period: 2024-12-05 - 2024-12-05
Report date: 2025-02-04
Version: 1.1
Visit: composable-security.com

Contents

1. Retest summary (2024-12-05)



The description of the current status for each retested vulnerability and recommendation has been added in its section.

1.1. Results

The **Composable Security** team was involved in a one-time iteration of verification whether the vulnerabilities detected during the tests (between 2024-10-03 and 2024-10-11) were removed correctly and no longer appear in the code.

The current status of detected issues is as follows:

- 4 vulnerabilities with a **low** impact on risk were handled as follows:
  - 1 has been acknowledged,
  - 3 have been fixed.
- 6 security **recommendations** were handled as follows:
  - 5 have been implemented,
  - 1 have been acknowledged.

The team was very involved in analyzing the proposed recommendations and improving the entire system. Each reported issue was handled with care and discussed to ensure the highest level of security. Those recommendations that were not implemented were reasonably justified by the team.

1.2. Scope

The retest scope included the same contracts, on a different commit in the same repository.

GitHub repository: https://github.com/redstone-finance/redstone-avs

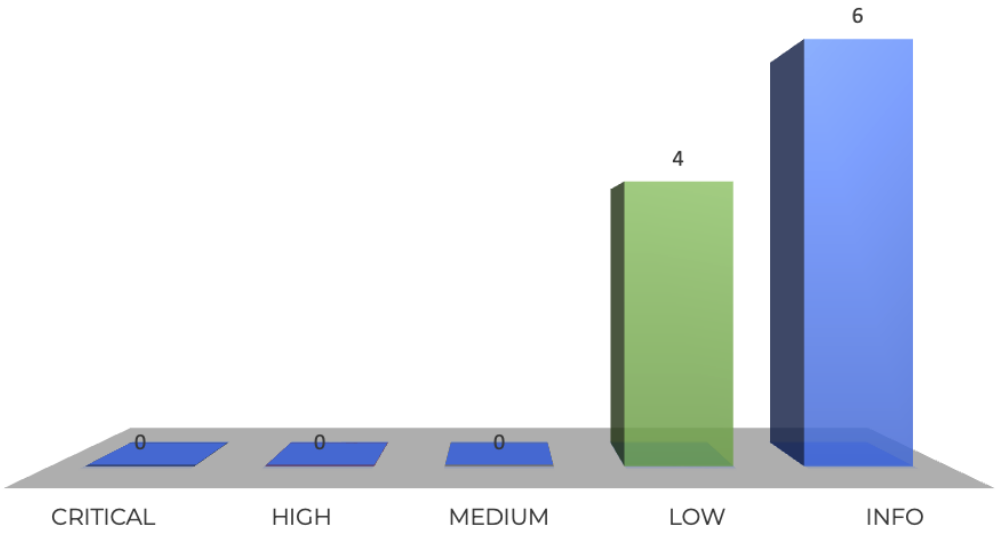
CommitID: 9b6771d9514b82a51de9ee5982ae432b381f0d26

2. Current findings status

ID	Severity	Vulnerability	Status
RS-747786a5-L01	LOW	Resubmitting the same data to generate rewards	FIXED
RS-747786a5-L02	LOW	Denial of service on desynchronization	ACKNOWLEDGED
RS-747786a5-L03	LOW	Stealing rewards by aggregator	FIXED
RS-747786a5-L04	LOW	Insecure architecture with single points of failure	FIXED

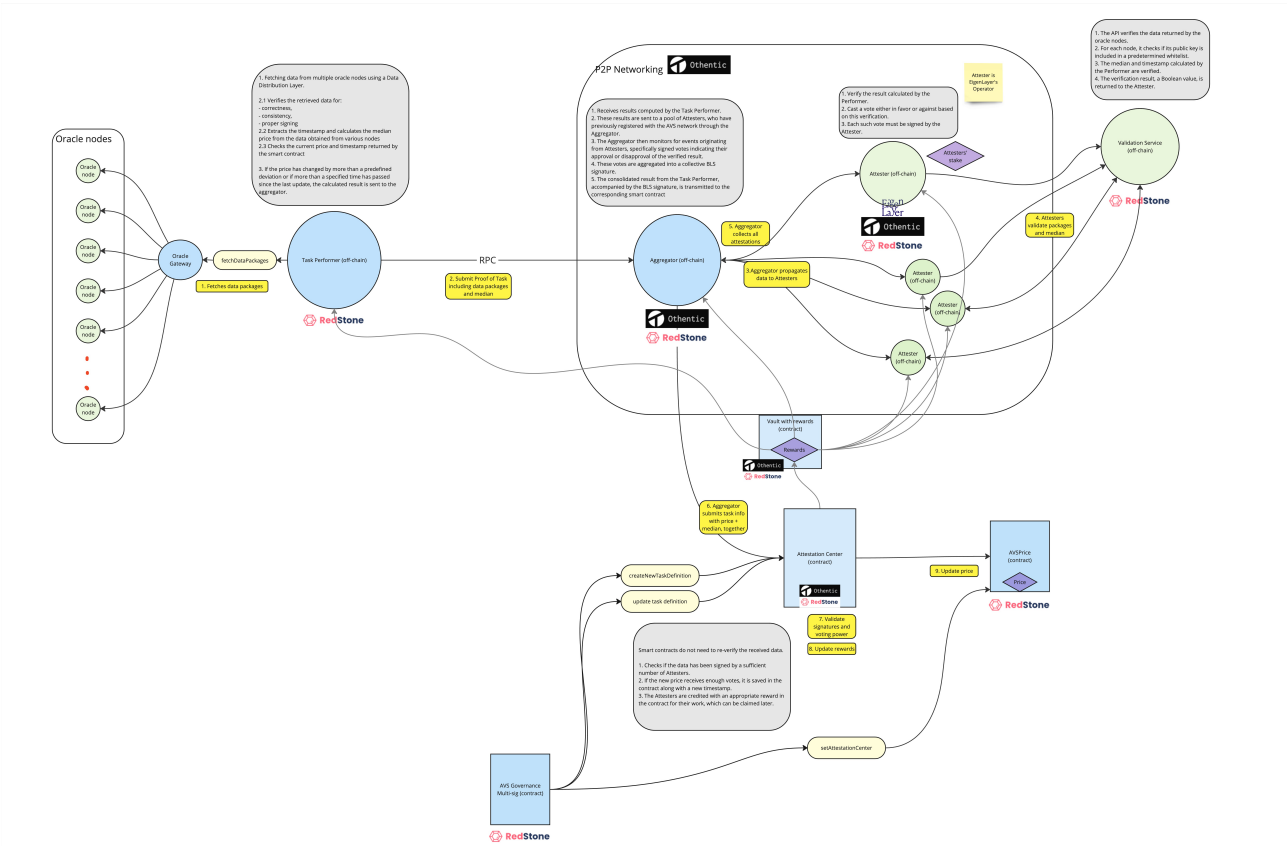
ID	Severity	Recommendation	Status
RS-747786a5-R01	INFO	Protect key material	IMPLEMENTED
RS-747786a5-R02	INFO	Harden docker containers	IMPLEMENTED
RS-747786a5-R03	INFO	Consider using specific solidity version	ACKNOWLEDGED
RS-747786a5-R04	INFO	Add indexed fields to events	IMPLEMENTED
RS-747786a5-R05	INFO	Consider adding NatSpec comments	IMPLEMENTED
RS-747786a5-R06	INFO	Emit events for important state changes	IMPLEMENTED

3. Security review summary (2024-10-11)



3.1. Client project

The **AVS** built by the RedStone team, is an Oracle. Its main task is to provide prices for selected assets in a secure and reliable way. AVS moves the validation of data off-chain, significantly reducing on-chain gas consumption of Push Model.



The AVS operates through several key components:

- **Oracle Gateway** - collects data from Oracle nodes.
- **Task Performer** - fetches data from Oracle Gateway. It verifies the retrieved data for correctness, consistency, and proper signing, then extracts timestamps and calculates the median price. It also checks the current price and timestamp from the smart contract. If the new price differs by more than a predefined deviation or if a certain time has passed since the last update, the calculated result is sent to the aggregator. This ensures the aggregator receives accurate and timely price updates.
- **Aggregator** - receives results computed by the Task Performer. These results are then forwarded to a pool of Attesters who have registered with the AVS network. The Aggregator actively monitors messages from these Attesters, specifically their signed votes indicating approval or disapproval of the verified result. The individual votes are aggregated into a collective BLS signature. Finally, the consolidated result from the Task Performer, along with this collective BLS signature and attesters' IDs, is submitted to Attestation Center.
- **Attester** - verifies the results calculated by the Performer. Based on this verification, it casts a vote either in favor of or against the result. Each vote is signed by the Attester, ensuring the authenticity and integrity of the decision.
- **Validation API** - responsible for verifying the data returned by the oracle nodes. It checks whether each node's public key is included in a predetermined whitelist, ensuring that only authorized nodes contribute data. It also verifies the median price and timestamp calculated by the Performer. Based on this verification, it returns a Boolean result to the Attester, indicating approval or disapproval of the data.
- **Vault with rewards** - hold rewards.
- **Attestation Center** - ensures that smart contracts do not need to re-verify the received data. It checks whether the data has been signed by a sufficient number of Attesters. If the new price obtains enough votes, it is sent to the RedStone AVS contract along with the new timestamp. Attesters and Task Performer are then credited with an appropriate reward for their work, which they can claim later.
- **AVSPrice** - Manages price updates via attestation verification.

3.2. Results

The **RedStone** engaged Composable Security to review security of their **AVS**. Composable Security conducted this assessment over 1 person-week with 2 engineers.

The summary of findings is as follows:

- No vulnerabilities with **critical** or **high** impact on risk were identified.
- 1 vulnerability with a **medium** impact on risk was identified.
- 4 vulnerabilities with a **low** impact on risk were identified.

- 5 **recommendations** have been proposed that can improve overall security and help implement best practice.
- The code is of a good quality, most of the detected issues concern the proper configuration and the possibility of hardening the architecture.
- During the security review, in addition to personalized threats, the following best practices were also verified:
  - AVS Developer Security Best Practices
  - Key Security Considerations for Developers
- There was no access to a fully functioning test environment during the security review.

Composable Security recommends that **RedStone** complete the following:

- Address all reported issues.
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Review dependencies and upgrade to the latest versions.

3.3. Centralization Risk

The current implementation of the system is highly centralized. Operations can be performed without any additional security mechanisms and have a critical impact on security of projects that are integrated with `AVSPrice` contract.

The owner role can immediately change `attestationCenter` responsible for delivering the price through `setAttestationCenter` function.

```
1  function setAttestationCenter(address attestationCenter_) external {
2    require(msg.sender == owner, "Caller must be the owner");
3    attestationCenter = attestationCenter_;
4  }
```

Although `AttestationCenter` was not part of this audit, it was also noted that this contract could also pose a risk without proper configuration because the roles there allow for powerful operations such as:

MESSAGE_HANDLER

- Register to network
- Unregister operator from network
- Clear payment
- Clear batch payment

AVS_GOVERNANCE_MULTISIG

- Request batch payment
- Set AVS logic

- Set before payment logic
- Set fee calculator
- Transfer AVS governance Multisig
- Create new task definition
- Set task definition minimum voting power
- Set task definition restricted operators
- Set OBLS shares syncer

OPERATIONS_MULTISIG

- Change the message handler address

VOTING_POWER_SYNCER

- Modify Operator voting power
- Modify Operator active status

Even though the code does not indicate any malicious intent, the users should be aware of their dependency.

Recommendation: To mitigate the risks associated with potential compromises, appropriate security requirements and procedures should be established to limit the impact in the event of loss of access or key compromise.

3.4. Low voting power pools risk

In the case of AVS, one of the key security assumptions is a sufficiently high voting power so that an attacker, as in the case of PoS, cannot easily manipulate it. For a task that updates the price of an asset to be considered valid, the data must be attested by participants whose combined voting power exceeds 2/3 of the total. This voting power is determined by the amount of tokens staked on Layer 1 or can be explicitly set per task.

When an AVS or task has a low total voting power threshold, it becomes feasible for an attacker to stake enough tokens to exceed the 2/3 threshold, allowing them to manipulate the task data submitted to the `AVSPrice` token and, consequently, affect the reported price.

To mitigate this risk, the team decided to implement attester whitelisting during the initial stage by utilizing the `setTaskDefinitionRestrictedOperators` function. This approach ensures that only trusted operators can participate, effectively mitigating the identified risk.

Long-term recommendation Consider including support for enforcing a required minimum total voting power. This will help minimize the risk of price manipulation once the system is fully open. E.g. Add dynamic checks for minimum voting power within the `beforeTaskSubmission` function to enhance security. While this approach could increase the likelihood of a denial of service if the threshold is unmet, it is preferable to the immediate exploitation that can occur through price manipulation.

3.5. Scope

Smart contracts security review

The scope of the tests included selected contracts from the following repository.

GitHub repository: https://github.com/redstone-finance/redstone-avs
CommitID: 747786a558a5250417c7f6332980327737cdb511

Off-chain components and integration security review

Time reserved for examining the Task Performer, Validation API, and threats coming from potentially malicious Attesters and Aggregators.

GitHub repository: https://github.com/redstone-finance/redstone-avs

CommitID: 747786a558a5250417c7f6332980327737cdb511

The detailed scope of tests can be found in Agreed scope of tests.

4. Project details

4.1. Projects goal

The Composable Security team focused during this audit on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **RedStone** and their users.
- The secondary goal is to improve code clarity and optimize code where possible.

4.2. Agreed scope of tests

Smart contracts security review

The subjects of the test were selected contracts from the **RedStone** repository.

GitHub repository:
https://github.com/redstone-finance/redstone-avs
CommitID: 747786a558a5250417c7f6332980327737cdb511

Files in scope:

```
avs-contracts/contracts
└── AvsPrice.sol
```

Off-chain components and integration security review

Time reserved for examining the Task Performer, Validation API, and threats coming from potentially malicious Attesters and Aggregators.

GitHub repository: https://github.com/redstone-finance/redstone-avs

CommitID: 747786a558a5250417c7f6332980327737cdb511

The analysis was preceded by threat modeling, which guided the subsequent verification of threat scenarios.

Out of scope: The Othentic external dependencies are not subject to security review.

Documentation: https://docs.redstone.finance/docs/category/-avs/

4.3. Threat analysis

This section summarizes the potential threats that were identified during initial threat modeling performed before the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Key assets that require protection:

- Price
- Price timestamp
- Private keys
- Task configuration

Potential attackers goals:

- Price manipulation.
- Theft of rewards.
- Lock of rewards.
- Bypassing voting power requirements.
- Data verification with unauthorized oracle.
- Account takeover.
- Lock users' funds in the contract.
- Denial of Service

Potential scenarios to achieve the indicated attacker's goals:

- Invalid verification of data correctness.
- Modifying fetched data.
- Improper signature validation.
- Using malicious data sources.
- Spoofing Task Performer.
- Submitting many tasks with different order of packages.
- Attesting a low voting power tasks.
- Improper calculations of median and timestamp.
- Adding Attesters who not contributed to steal part of reward.
- Unauthorized change of Attestation Center.
- Influence or bypass the business logic of the system.
- Privilege escalation through incorrect access control.
- Design issues.
- Excessive power, too much in relation to the declared one.
- Unintentional loss of the ability to govern the system.
- Poor security against taking over the managing account.
- Private key compromise, rug-pull.

4.4. Testing methodology

Smart contract security review was performed using the following methods:

- Q&A sessions with the **RedStone** development team to thoroughly understand intentions and assumptions of the project.

- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests using slither.
- Custom scripts (e.g. unit tests) to verify scenarios from initial threat modeling.
- **Manual review of the code.**

4.5. Disclaimer

AVS security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing off-chain components and smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.

5. Vulnerabilities

[RS-747786a5-L01] Resubmitting the same data to generate rewards

LOW FIXED

Retest (2024-12-05)

The vulnerability has been fixed as recommended. Now the new price timestamp must be greater than the current one.

Affected files

- AvsPrice.sol#L48

Description

The `AttestationCenter` contract detects when there is the same task submitted multiple times using the hash value of task info and reverts such submission.

However, there are several ways in which the same task data can be resubmitted with modified task information to bypass this detection:

1. **Altering proofOfTask:** Since proofOfTask is not verified within the contract, changing its value results in a different hash, allowing the same task data to appear unique.
2. **Reordering Oracle Data Packages:** Modifying the order of the oracle data packages changes the hash value, even though the underlying data remains the same.
3. **Changing the Task Performer:** Submitting the task under a different task performer alters the task information hash, bypassing the duplicate submission check.

These methods enable an attacker to submit identical oracle data multiple times and unjustly receive additional rewards. The same task data (price and timestamp) are also accepted by the `AVSPrice` contract as it reverts only if the new timestamp is lower than the current one:

```
48      if (newPriceTime < priceTime || newPriceTime > block.timestamp) {
49        revert InvalidPriceTime(newPriceTime);
50      }
```

Result of the attack: Extorting multiple rewards for the same task.

Recommendation

Do not accept the price update with the timestamp equal to the current one:

```
- if (newPriceTime < priceTime || newPriceTime > block.timestamp) {
+ if (newPriceTime <= priceTime || newPriceTime > block.timestamp) {
```

References

1. SCSVS G4: Business logic

[RS-747786a5-L02] Denial of service on desynchronization

LOW ACKNOWLEDGED

Retest (2024-12-05)

The team decided to continue to check if the gateway has definitely returned the same timestamps and throw an exception otherwise, due to the following reasons given by the team:

- all other RedStone contracts reject an update that has different timestamps, so it would be inconsistent behavior of AVS,
- it is unlikely that the gateway will return different timestamps, because we have it well tested and monitored.

Affected files

- TaskPerformer.ts#L89
- TaskValidator.ts#L108

Description

The Task Performer aggregates data packages from multiple oracle nodes, each containing a `timestampMilliseconds` field that records the exact time the value was retrieved from the source. However, if there is even a minimal discrepancy (as small as 1 millisecond) between the timestamps of different data packages, the Task Performer cancels the submission and aborts the price update.

Note: During the discussion with the RedStone team it has been raised that the `timestampMilliseconds` gets its milliseconds cleared so the oracle nodes need to be synchronised to 1 sec. Even though there might be a case when different oracle nodes query the data on the edge of a second and desynchronise, such situations are monitored and oracles get synchronised again.

This lowers the severity of the issue, but the issue is still reported because it can come back when more oracles are started and especially, when multiple oracle providers are introduced.

Vulnerable scenario

The following steps lead to the described result:

① One of five oracles submit different timestamp than the others.
② The Task Performer tries to get the timestamp, but the set of timestamps has two values, so the Task Performer throws an error and cancels the current task generation.

Result: DoS by reporting different (yet correct) timestamps as Oracles.

Recommendation

Consider either allowing a range of values for the timestamp and calculate the median or select only those data packages that have the same timestamp and are the majority. If the first recommendation is selected, remember to update the code of `TaskValidator` to filter data packages that fit the range.

References

1. CWE-703: Improper Check or Handling of Exceptional Conditions

[RS-747786a5-L03] Stealing rewards by aggregator

LOW FIXED

Retest (2024-12-05)

The team decided to follow the recommendation and use the functions `setTaskDefinitionMinVotingPower` and `setTaskDefinitionRestrictedOperators`.

Affected files

- AvsPrice.sol

Description

The role of the Aggregator is to collect all attestations, verify they meet the required voting power, and then submit the task to the `AttestationCenter` along with Attesters signatures.

In the process of forwarding tasks to the `AttestationCenter`, the Aggregator has the ability to add additional signatures from attesters it controls. These attesters, despite not having validated the data, are still eligible for the base reward.

Attack scenario

The following sequence outlines a potential attack:

① The Aggregator registers multiple attesters on the network with minimal stakes.
② Once a task has been processed and attested by the required number of legitimate attesters, the Aggregator adds signatures from their controlled attesters.
③ These added signatures use the same `isApproved` field value.
④ As a result, the `AttestationCenter` distributes the base fee to all attesters, including those controlled by the Aggregator that did not participate in data validation.

Result of the attack: This vulnerability allows for undeserved base rewards to be distributed to Attesters that did not contribute to task validation, leading to potential financial exploitation.

Recommendation

To mitigate this risk, consider implementing the following measures:

- Define a minimum voting power threshold for attesters using the `setTaskDefinitionMinVotingPower` function.
- Implement attester whitelisting at the initial stage, using the `setTaskDefinitionRestrictedOperators` function.
- Add dynamic checks for minimal voting power and verify attester IDs in the `beforeTaskSubmission` function.

Additionally, as slashing and penalty mechanisms are introduced, ensure that inactive attesters are penalized accordingly.

References

1. SCSVS G5: Access control

[RS-747786a5-L04] Insecure architecture with single points of failure

LOW FIXED

Retest (2024-12-05)

During the discussions it was agreed that only the test environment contained individual components in a basic, minimally working form. According to the team's response, the production environment is configured securely:
The production environment that was not subject to the audit, provides or will provide redundancy, load balancing, rate limiting and other recommendations from the report.

Description

The architecture of the designed system is vulnerable to both external and internal risks. Without redundancy, an overload, takeover, or shutdown of any component (i.e. `Task Performer` or `Aggregator`) would render the entire system inoperative, halting all price reporting.

Moreover, the DDoS attack directed at a specific `Validation API` service might block its attester from submitting the prices.

Note: The severity has been downgraded to Low because the vulnerability could not be fully verified in a practical setting due to the absence of a complete replicated production environment.

Vulnerable scenario

The attackers might take the following steps in turn to cause the Denial of Service of a particular attester:

① The attacker identifies that the `Validation API` service belonging to a particular attester.
② They launch a Denial-of-Service (DoS) attack by sending a high volume of requests to the `Validation API` service.
③ The overwhelming traffic causes the `Validation API` service to crash or become unresponsive.
④ With the `Validation API` service down, the attester cannot validate and aggregate price data, halting price reporting.

The following steps would cause the Denial of Service:

① The internal error or network failure causes the `Task Performer` or `Aggregator` service to crash or become unresponsive.
② With the `Task Performer` or `Aggregator` service down, the entire system cannot submit and handle tasks, halting all price reporting.

Result: Complete halt in price updates or blocking the operation of a specific attester due to unresponsive or shut down components.

Recommendation

- **Isolate Critical Components:** Deploy the `Task Performer` and `Validation API` service in separate environments with distinct management access to prevent simultaneous compromise.
- **Implement Redundancy:** Consider using the load balancer and duplicating the `Validation API` service to handle high traffic volumes and ensure continuous operation even if one instance fails.
- **Apply Rate Limiting:** Introduce rate limiting on the `Aggregator` to restrict the

number of data packages accepted from each Attester and on the `Validation API` service to restrict the number of validation requests, preventing resource exhaustion from repeated submissions.

- **Expose Only Necessary Ports:** Configure components to allow traffic only on essential ports required for your services to function. Close all non-essential ports to minimize the attack surface and prevent unauthorized access to your network resources.

References

1. SCSVS G1: Architecture, design and threat modeling

6. Recommendations

[RS-747786a5-R01] Protect key material

INFO IMPLEMENTED

Retest (2024-12-05)

The `.env` file is used only in the local environment. According to the team's response, the key material is securely managed in the production environment: *The production environment has sensitive configuration data appropriately encrypted and delivered only at application startup.*

Description

The key material belonging to Task Performer is used to sign the performed task. Keys should be encrypted, either using a password or a passphrase. Currently, the `PRIVATE_KEY` of Task Performer is kept in the `.env` file in plain text.

Recommendation

Pass the private key using Docker `environment` field as it will be possible to configure the cloud service that starts the containers to read them from encrypted storage and pass to container's memory.

References

1. Key Security Considerations for Developers

[RS-747786a5-R02] Harden docker containers

INFO IMPLEMENTED

Retest (2024-12-05)

The container user has been set to an unprivileged `node` user as recommended. According to the team's response, the resources are limited in the production environment: *The production environment has set limits on resource consumption by the container.*

Description

By default, Docker executes commands inside the container as the root user. This violates the Principle of Least Privilege when superuser permissions are unnecessary. Additionally, any Docker Container can consume unlimited hardware resources such as CPU and RAM, potentially leading to resource exhaustion or system performance issues.

Recommendation

Implement the following recommendations:

- Run the container as an unprivileged user. The node images provide the `node` user for such purpose. Use `USER node` in `Dockerfile` or `-u "node"` flag.
- If you are running multiple containers on the same host, you should limit how much memory they can consume. Use `-m "300M"--memory-swap "1G"` flags.
- Use "Docker Bench for Security" tool.

References

1. Docker and Node.js Best Practices
2. Docker Bench for Security

[RS-747786a5-R03] Consider using specific solidity version

INFO ACKNOWLEDGED

Retest (2024-12-05)

The team decided to keep the pragma as it is due to the fact that the Solidity version in AVS is consistent with the version in the other RedStone packages. The team's response: *We want to maintain this consistency. We regularly update the dependency versions in our projects, and the Solidity version will also be updated according to this procedure.*

Description

In accordance with the best security practices, it is recommended to use the latest stable versions of major Solidity releases.

Very often, older versions contain bugs that have been discovered and fixed in newer versions. Moreover, it is worth remembering that the version should be clearly specified so that all tests and compilations are performed with the same version.

The current implementation uses floating pragma:

```
1 pragma solidity >=0.8.20;
```

Recommendation

Use the latest stable version of major Solidity release:

```
1 pragma solidity 0.8.27;
```

Note: If it is planned to deploy on multiple chains, stay aware that some of them don't support `PUSH0` opcode. If solc `>=0.8.20` is used, the `PUSH0` opcode will be present in the bytecode. In this situation, it is recommended to choose `0.8.19`.

References

1. SCSVS G1: Architecture, design and threat modeling
2. Floating pragma

[RS-747786a5-R04] Add indexed fields to events

INFO IMPLEMENTED

Retest (2024-12-05)

The recommendation has been implemented as recommended.

Description

Indexing event fields makes them more quickly accessible to off-chain tools that parse events, which is especially useful for filtering based on addresses. However, it is worth to notice that each indexed field incurs extra gas costs during emission. Therefore, it's not always optimal to index the maximum allowed number of fields per event (three fields), as it may lead to unnecessary expenses.

Recommendation

Consider marking the `price` and `newPrice` fields as `indexed` in the following event:

```
53 event PriceUpdated(uint256 price, uint256 priceTime);
```

References

1. SCSVS G1: Architecture, design and threat modeling
2. Principles and Best Practices to Design Solidity Events in Ethereum and EVM

[RS-747786a5-R05] Consider adding NatSpec comments

INFO IMPLEMENTED

Retest (2024-12-05)

The recommendation has been implemented as recommended.

Description

Comments increase readability and clearly inform about the assumptions of a given contract. Thanks to NatSpec it will be much easier not only to understand the code faster, but also to create accurate documentation based on it.

Below is an example of what this might look like:

```
1  // SPDX-License-Identifier: BUSL-1.1
2  pragma solidity >=0.8.20;
3
4  import "./IAvsLogic.sol";
5  import "./IAttestationCenter.sol";
6
7  /**
8      @title AvsPrice Contract
9      @author RedStone Finance
10     @notice Manages price updates via attestation verification.
11  */
12 contract AvsPrice is IAvsLogic {
13     /**
14     * @notice Emitted when the price is updated.
15     * @param price The new price value.
16     * @param priceTime The timestamp when the new price was recorded.
17      */
18     event PriceUpdated(uint256 price, uint256 priceTime);
19
20     /**
21     * @notice Thrown when an invalid price time is provided.
22     * @param invalidPriceTime The invalid price time that was provided.
23      */
24     error InvalidPriceTime(uint256 invalidPriceTime);
25
26     /// @notice The owner of the contract.
27     address public owner = msg.sender;
```

```
28
29      /// @notice The address of the attestation center contract.
30      address public attestationCenter;
31
32      /// @notice The current price value.
33      uint256 public price;
34
35      /// @notice The timestamp when the current price was recorded.
36      uint256 public priceTime;
37
38      /**
39      * @notice Function called before a task is submitted. Currently does nothing.
40      * @param _taskInfo The task information (unused).
41      * @param _isApproved Indicates if the task is approved (unused).
42      * @param _tpSignature The signature of the trusted party (unused).
43      * @param _taSignature The signature of the attestation (unused).
44      * @param _operatorIds The operator IDs involved (unused).
45       */
46      function beforeTaskSubmission(
47          IAttestationCenter.TaskInfo calldata /*_taskInfo*/,
48          bool /*_isApproved*/,
49          bytes calldata /*_tpSignature*/,
50          uint256[2] calldata /*_taSignature*/,
51          uint256[] calldata /*_operatorIds*/
52      ) external {}
53
54      /**
55      * @notice Processes the task after submission if it is approved.
56      * @dev Updates the price and priceTime if the task is approved and called by
            the attestation center.
57      *      Extracts new price and time from the task data.
58      * @param taskInfo The task information containing the data.
59      * @param isApproved Indicates if the task has been approved.
60      * @param _tpSignature The signature of the trusted party (unused).
61      * @param _taSignature The signature of the attestation (unused).
62      * @param _operatorIds The operator IDs involved (unused).
63       */
64      function afterTaskSubmission(
65          IAttestationCenter.TaskInfo calldata taskInfo,
66          bool isApproved,
67          bytes calldata /*_tpSignature*/,
68          uint256[2] calldata /*_taSignature*/,
69          uint256[] calldata /*_operatorIds*/
```

```
70      ) external {
71          require(
72              msg.sender == attestationCenter,
73              "Caller must be the attestation center"
74          );
75          if (isApproved) {
76              bytes calldata data = taskInfo.data;
77              uint128 newPrice;
78              uint128 newPriceTime;
79              assembly {
80                  // 60 = REDSTONE_MARKER_BS + UNSIGNED_METADATA_BYTE_SIZE_BS +
                        MEDIAN_AND_TIMESTAMP_BS + 32
81                  newPrice := calldataload(add(data.offset, sub(data.length, 60)))
82                  // 44 = REDSTONE_MARKER_BS + UNSIGNED_METADATA_BYTE_SIZE_BS + 32
83                  newPriceTime := calldataload(add(data.offset, sub(data.length, 44)
                        ))
84              }
85              if (newPriceTime < priceTime || newPriceTime > block.timestamp) {
86                  revert InvalidPriceTime(newPriceTime);
87              }
88              price = newPrice;
89              priceTime = newPriceTime;
90              emit PriceUpdated(price, priceTime);
91          }
92      }
93
94      /**
95      * @notice Sets the attestation center address.}
96      * @dev Only callable by the owner.}
97      * @param attestationCenter_ The address of the new attestation center.}
98       */
99      function setAttestationCenter(address attestationCenter_) external {
100         require(msg.sender == owner, "Caller must be the owner");
101         attestationCenter = attestationCenter_;
102     }
103 }
```

Recommendation

Add NatSpec comments.

References

1. SCSVS G11: Code clarity

[RS-747786a5-R06] Emit events for important state changes

INFO IMPLEMENTED

Retest (2024-12-05)

The recommendation has been implemented as recommended.

Description

The `setAttestationCenter` function does not emit an event when the attestation center address is changed.

```
57  function setAttestationCenter(address attestationCenter_) external {
58    require(msg.sender == owner, "Caller must be the owner");
59    attestationCenter = attestationCenter_;
60  }
```

Recommendation

Emit an event when the attestation center address is changed.

References

1. SCSVS G1: Architecture, design and threat modeling
2. Principles and Best Practices to Design Solidity Events in Ethereum and EVM

7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

OVERALL RISK SEVERITY				
Impact on risk	HIGH	CRITICAL	HIGH	MEDIUM
	MEDIUM	MEDIUM	MEDIUM	LOW
	LOW	LOW	LOW	INFO
		HIGH	MEDIUM	LOW
		Likelihood		

[1] OWASP Risk Rating methodology

8. Long-term best practices

8.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools (e.g. slither) into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

8.2. Perform threat modeling

Before implementing or introducing changes to smart contracts, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

8.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

8.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

8.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.

8.6. Go through the provided checklists

The security of a project is influenced not only by its code, but also by the level of security awareness of the team. Read and apply the checklists prepared by Security Alliance:

- X/Twitter security self-audit
- Telegram Security Self-audit
- Google Security Self-Audit

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com