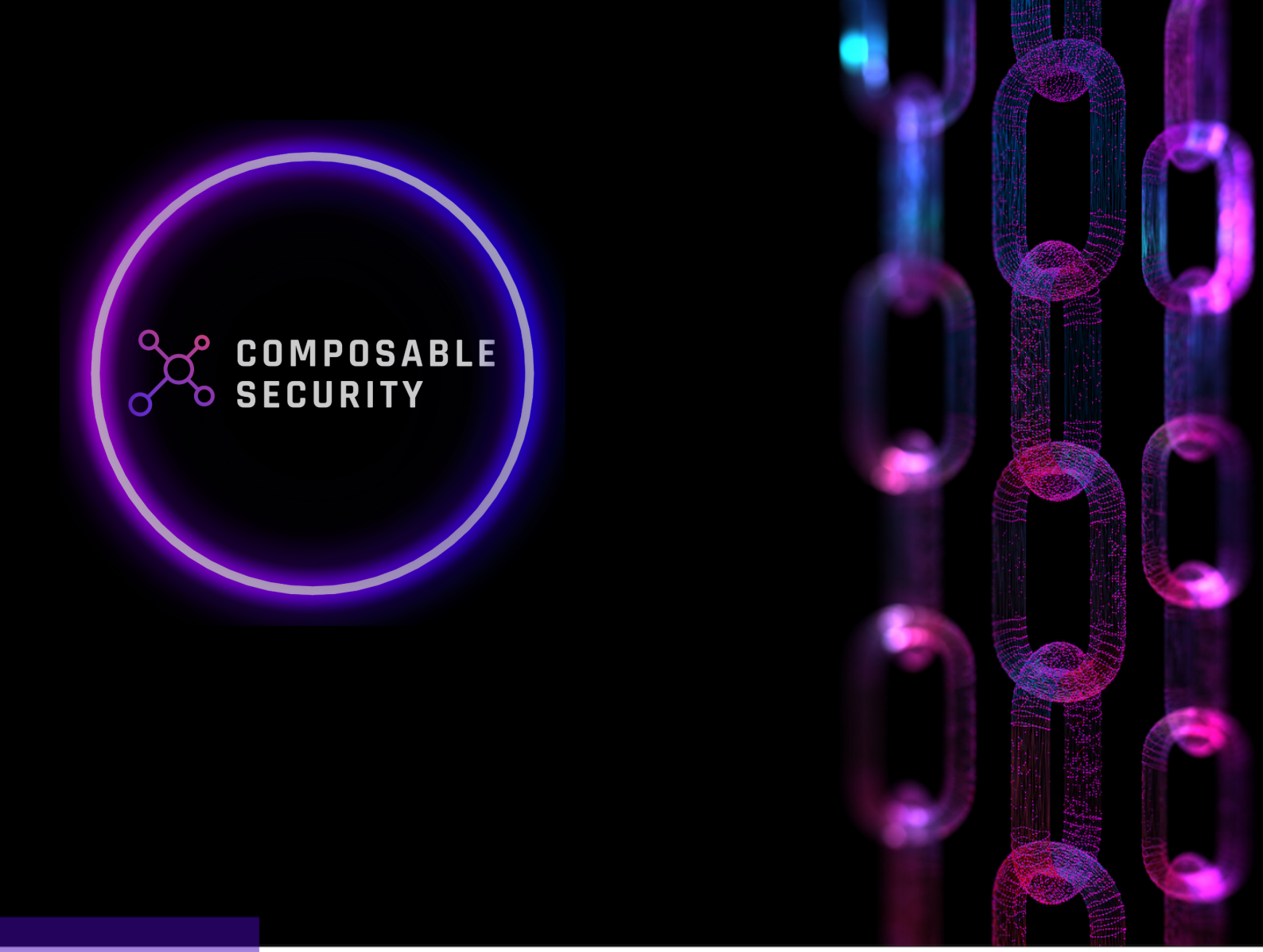


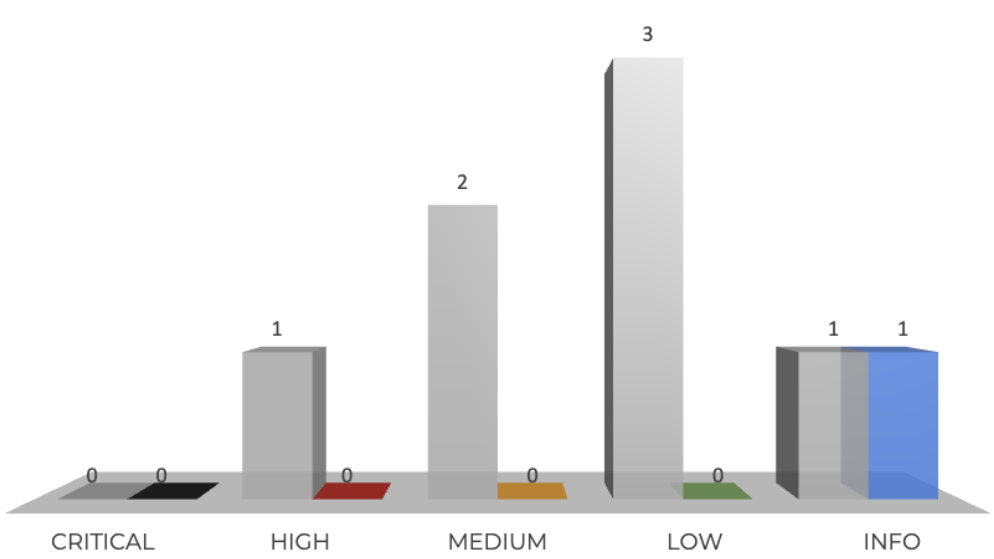
REPORT

Smart contract security review for Othentic

Prepared by: Composable Security
Report ID: OTNC-9a405548
Test time period: 2025-02-10 - 2025-02-14
Retest time period: 2025-02-27 - 2025-03-11
Report date: 2025-03-11
Version: 1.0
Visit: composable-security.com

Contents

1. Retest summary (2025-03-11)



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 participated in a one-time iteration to verify if the vulnerabilities detected during the tests (between 2025-02-10and 2025-02-14) were correctly removed and no longer appear in the code.

The current status of detected issues is as follows:

- The only one **high** vulnerability has been fixed.
- All **2** vulnerabilities with a **medium** impact on risk were fixed.
- All **3** vulnerabilities with a **low** impact on risk were fixed.
- The security **recommendation** has not been implemented.

1.2. Scope

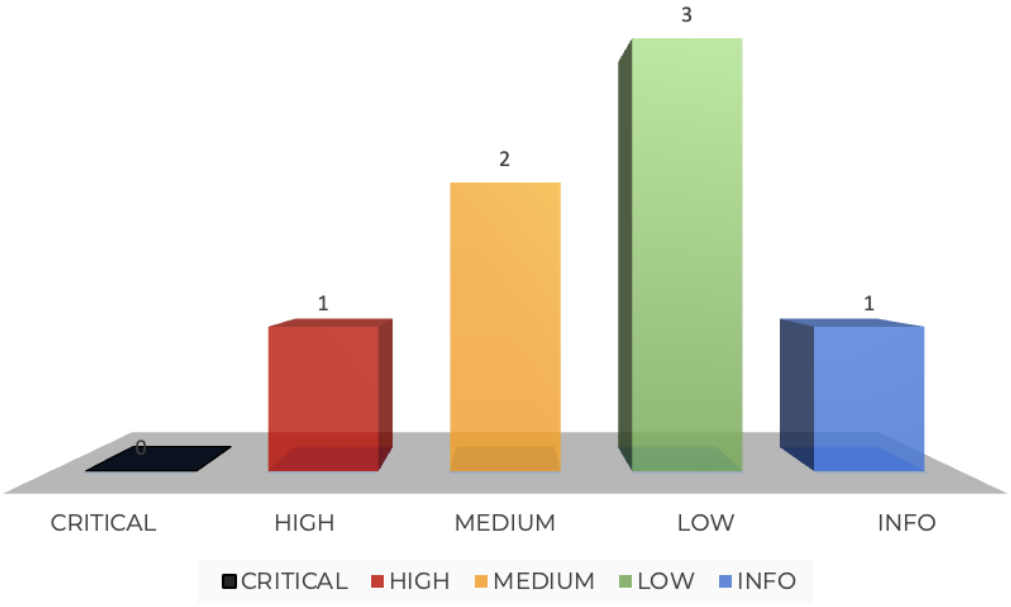
The retest scope included the same contracts, on a different commit in the same repository.

GitHub repository: https://github.com/Othentic-Labs/contracts
CommitID: 9fc45f6e6291ea2f5beb2a25c1ec9deb2615abd7

2. Current findings status

ID	Severity	Vulnerability	Status
OTNC-9a405548-H01	HIGH	Uncleared claims	FIXED
OTNC-9a405548-M01	MEDIUM	Permanent revert on rewards submission	FIXED
OTNC-9a405548-M02	MEDIUM	Invalid EigenLayer reward submission	FIXED
OTNC-9a405548-L01	LOW	Unsafe cast	FIXED
OTNC-9a405548-L02	LOW	Duplicated operators in rewards submission	FIXED
OTNC-9a405548-L03	LOW	Double add of the default LayerZero endpoint	FIXED
ID	**Severity**	**Recommendation**	**Status**
OTNC-9a405548-R01	INFO	Include L2 identifier in emitted payment request events	NOT IMPLEMENTED

3. Security review summary (2025-02-14)

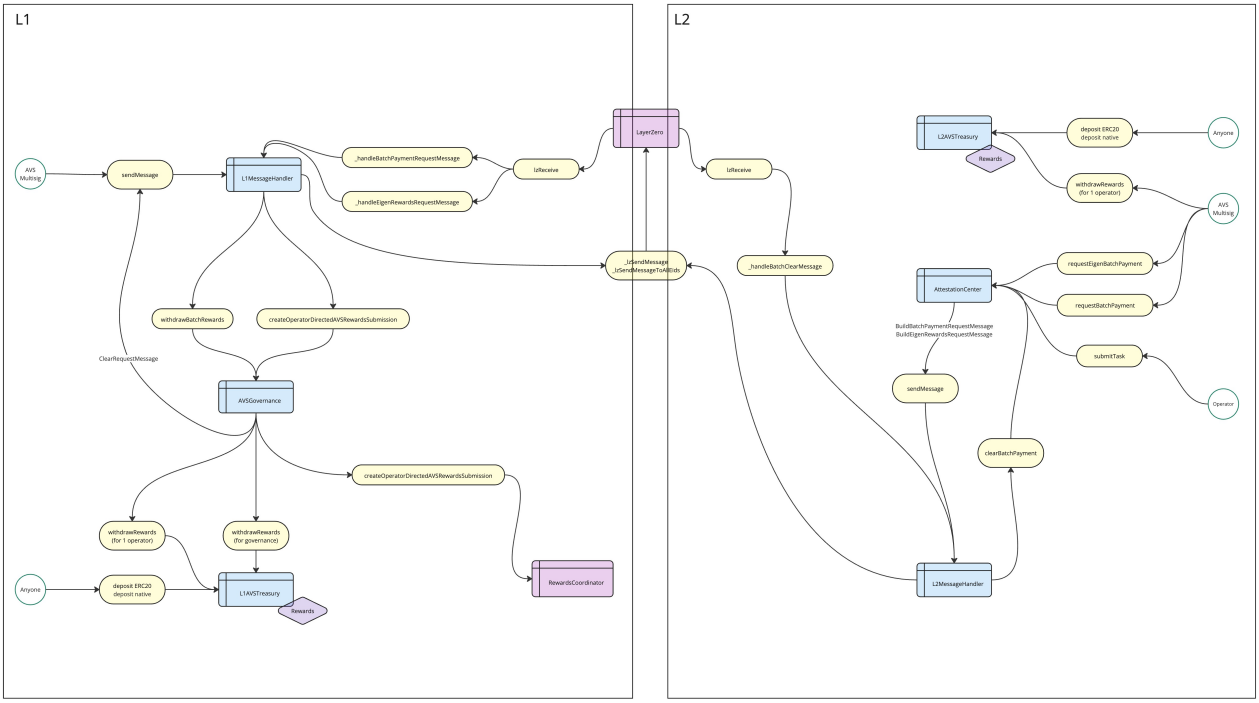


3.1. Client project

The **Othentic** project is a framework to build distributed systems, highly configurable, and versatile in all aspects of Actively Validated Services (AVS) development.

The **Rewards V2** update adds support for EigenLayer V2 Rewards distribution through the `RewardsCoordinator` contract's `createOperatorDirectedAVSRewardsSubmission` function.

This implementation adds a new flow in the `AttestationCenter` contract which triggers Eigen rewards distribution when provided with the necessary parameters (start timestamp, duration, and description).



3.2. Results

The **Othentic** engaged Composable Security to review security of **Rewards V2**. Composable Security conducted this assessment over 4 person-days with 2 engineers.

The summary of findings is as follows:

- no **critical** risk impact vulnerabilities were identified.
- 1 vulnerability with a **high** impact on risk was identified. Its potential consequence is losing rewards on the default L2 by operators and also jeopardize subsequent rewards on the non-default L2.
- 2 vulnerabilities with a **medium** impact on risk were identified.
- 3 vulnerabilities with a **low** impact on risk were identified.
- 1 **recommendation** has been proposed that can improve overall security and help implement best practice.
- The most important issues detected concern *Othentic* integration with LayerZero and cross-chain communication.
- The team was engaged and the communication was very good.

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

- Address all reported issues.
- Extend unit tests with scenarios that cover detected vulnerabilities where possible.
- 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. Scope

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

GitHub repository: https://github.com/Othentic-Labs/contracts
CommitID: 9a405548df489a489d6344242577f7ac50ff4bc3

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 **Othentic** and their users.
- The secondary goal is to improve code clarity and optimize code where possible.

4.2. Agreed scope of tests

The subject of the test was the following pull request from the **Othentic** repository: #280

GitHub repository:
https://github.com/Othentic-Labs/contracts
CommitID: 9a405548df489a489d6344242577f7ac50ff4bc3

Documentation: https://docs.othentic.xyz/main

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:

- Reward tokens.

Potential attackers goals:

- Theft of rewards.
- Lock rewards in the contract.
- Block the contract, so that others cannot use it.

Potential scenarios to achieve the indicated attacker's goals:

- Invalid parameters sent to EigenLayer.
- Invalid operator sorting.
- No update of fee to claim for claimed reward do multiple chains (clearance sent to one chain).
- Subtracting fee to claim for non-claimed reward.
- Submitting invalid strategies to EigenLayer.
- Spamming the operators list.

- Excluding some operators in rewards distribution.
- Permanent revert on `lzReceive`.
- Invalid rewards calculation.
- Influence or bypass the business logic of the system.
- Take advantage of arithmetic errors.
- Privilege escalation through incorrect access control to functions or badly written modifiers.
- Existence of known vulnerabilities (e.g., front-running, re-entrancy).
- Design issues.

4.4. Testing methodology

Smart contract security review was performed using the following methods:

- Q&A sessions with the **Othentic** 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

Smart contract 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 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

[OTNC-9a405548-H01] Uncleared claims

HIGH FIXED

Retest (2025-02-27)

The vulnerability has been fixed as recommended. The remote ID is taken from the cross-chain message and used to send the confirmation message.
CommitID: *05a7282b030146ca2caab2fb0e77bc88b5613109*

Affected files

- AvsGovernance.sol#L238
- AvsGovernance.sol#L182

Description

The `AVSGovernance` contract on Layer 1 (L1) can interface with multiple `AttestationCenter` contracts located on various Layer 2 (L2) networks. For any cross-chain communication initiated from these L2s that requires confirmation, it is essential to send the confirmation back to the specific `AttestationCenter` that originated the communication.

In the current rewards distribution implementation, when an `AttestationCenter` on any L2 requests rewards distribution, it awaits confirmation from L1 to complete the process (which includes updating payment status and the fee to claim value).

However, the `AVSGovernance` contract on L1 incorrectly routes the confirmation to the `AttestationCenter` contract associated with a default L2, identified by the `lzEid` variable, regardless of the original request's source.

Vulnerable scenario

The scenario unfolds as follows:

① The AVS is deployed on Ethereum, with the `AVSGovernance` contract supporting Arbitrum and Optimism via their respective `AttestationCenter` contracts, with Arbitrum designated as the default.
② Operators receive rewards on both L2s (`feeToClaim` updated on both L2s).
③ During a rewards distribution request coming from Optimism through EigenLayer, the `AttestationCenter` contract on Optimism sends a rewards distribution request to L1.

④ Upon processing, the `AVSGovernance` contract on L1 redistributes rewards but mistakenly sends confirmation to the `AttestationCenter` contract **on Arbitrum**.
⑤ Corresponding fees to claim on Arbitrum are deducted (if they exist), resulting in a loss of rewards for the operator on Arbitrum.
⑥ The payment status on Optimism remains marked as COMMITTED, preventing further rewards distribution from Optimism.

Result: Operators face the risk of losing rewards on the default L2 and also jeopardize subsequent rewards on the non-default L2.

Recommendation

The `AVSGovernance` contract must be updated to accurately read the `lzEid` of the chain that submits the rewards distribution request. This information should then be used to correctly send back the `BATCH_CLEAR_SIG` message to the corresponding `AttestationCenter`.

References

1. SCSVS G4: Business Logic

[OTNC-9a405548-M01] Permanent revert on rewards submission

MEDIUM FIXED

Retest (2025-03-11)

The vulnerability has been fixed as recommended. Contract uses `SafeERC20` library and `safeIncreaseAllowance` function.
CommitID: *1123f9e5c8a6642ce5efe5aa0733c9c3e49318e4*

Affected files

- AvsGovernance.sol#L213

Description

In the `AVSGovernance` contract, when processing a payment request from the `AttestationCenter` contract, the function `createOperatorDirectedAVSRewardsSubmission` constructs the submission and prepares to interact with the `RewardsCoordinator` contract. This involves withdrawing rewards from the treasury and granting token approval.

While the withdrawal of rewards and the call to the `RewardsCoordinator` contract are conducted without the possibility of reverting, the approval process is not handled in the same way. Some tokens do not permit approval from a non-zero value to another non-zero value, leading to a potential revert.

If this occurs, the execution of `createOperatorDirectedAVSRewardsSubmission` will fail, resulting in successful cross-chain execution on the source chain (in `AttestationCenter`), but rewards will remain undelivered, and operators' payment statuses will not be updated.

Vulnerable scenario

The situation unfolds as follows:

1. The AVS utilizes a token that restricts approvals from a non-zero to a non-zero value.
2. The AVS manager initiates the distribution of rewards.
3. The `AVSGovernance` contract invokes the function `createOperatorDirectedAVSRewardsSubmission`. The token is approved successfully, but the subsequent call to `createOperatorDirectedAVSRewardsSubmission` in the `RewardsCoordinator` contract fails (for instance, due to the flow being paused).
4. The `AVSGovernance` contract modifies the operator's list to indicate that no rewards have been distributed.
5. The `AttestationCenter` contract on L2 updates the payment statuses of operators, leaving the claimable fees unaltered.
6. The AVS manager attempts to distribute the rewards again.
7. Upon retrying `createOperatorDirectedAVSRewardsSubmission`, the approval call now reverts.

Result: Permanent blockage in the rewards distribution process.

Recommendation

Implement the `SafeERC20` library to manage token approvals robustly, either by increasing or decreasing approval amounts.

References

1. SCSVS I2: Token
2. SCSVS G4: Business Logic
3. SCSVS G8: Denial of Service

[OTNC-9a405548-M02] Invalid EigenLayer reward submission

MEDIUM FIXED

Retest (2025-02-27)

The vulnerability has been fixed as recommended. The operators' list length is updated to reflect the number of non-empty items.
CommitID: *1eb0277b7c5c5013ffa92522c1387b30fb764cf6*

Affected files

- AttestationCenter.sol#L503-L529
- AttestationCenter.sol#L279

Description

The `OperatorDirectedRewardsSubmission` struct submitted to EigenLayer undergoes validation, and the transaction will revert if any validation rules are not satisfied. One critical rule is that the operator addresses must be in ascending order.

In the `AttestationCenter` contract, the `requestEigenBatchPayment` function sorts the operators list obtained from `_collectEligibleOperators`. However, this list may include trailing items with a zero address, representing empty operators.

For this to happen, some operators in the queried range must meet certain conditions:

```
_details.operator == address(0) ||
_details.paymentStatus != PaymentStatus.REDEEMED ||
_details.lastPaidTaskNumber <= _taskNumber ||
_details.feeToClaim == 0
```

Subsequently, this list is forwarded to the `AVSGovernance` contract, which passes it to EigenLayer's `RewardsCoordinator`. The transaction reverts because the zero addresses appear at the end of the list and are considered smaller than the actual operator addresses.

Vulnerable scenario

The following sequence of events leads to this issue:

① The AVS is deployed on Ethereum, and the `AVSGovernance` contract supports Arbitrum through the corresponding `AttestationCenter` contract.

② There are 100 operators registered with the AVS, out of which 95 have received rewards (`feetoClaim` > 0).
③ The AVS manager attempts to distribute rewards to all 100 operators, resulting in a list with 95 valid operators followed by 5 empty entries.
④ The `AVSGovernance` contract receives the payment request and forwards the operators' list to EigenLayer.
⑤ The `RewardsCoordinator` checks that each operator address is greater than the previous one. Upon reaching index 95 (the first empty entry), it identifies that a zero address is not greater than any of the actual operator addresses, causing a revert.

Result: The standard rewards distribution process (as recommended by the CLI) fails, resulting in gas loss for the AVS manager.

Recommendation

Truncate the list of rewarded operators to eliminate empty items.

References

1. SCSVS G4: Business Logic
2. SCSVS G8: Denial of Service

[OTNC-9a405548-L01] Unsafe cast

LOW FIXED

Retest (2025-02-27)

The vulnerability has been fixed with the second approach. The `_votingPowerMultiplier` parameter is validated whether it is lower or equal to `type(uint96).max`.
CommitID: *f5a9c4ce182743fd36fce24fe099cecc4198e3e3*

Affected files

- AvsGovernance.sol#L444

Description

The `_buildRewardsSubmission` function needs to modify the type of the multiplier from `uint256`, as defined by Othentic, to `uint96`, which is a requirement of EigenLayer. This change is implemented using an unsafe cast that truncates bits beyond the 96th position.

If the multiplier exceeds `2**96`, the highest 160 bits will be cleared. It results in a signifi-

cant reduction of its value, which could potentially set it to zero. This situation arises when multipliers have 29 or more decimal places.

Note: The severity of this issue has been classified as low due to the unlikely occurrence of such event.

Vulnerable scenario

The following sequence of actions can lead to the outlined issue:

① The AVS manager sets the strategy multiplier to `2**96` using `setStakingContractMultiplier` function, executed by a party that holds the `MULTIPLIER_SYNCER` role.
② The AVS manager starts the rewards distribution process from some L2 via `AttestationCenter` contract.
③ The `AVSGovernance` contract receives the request and builds a submission to EigenLayer. The multiplier of the strategy is casted to `uint96` which modifies the value from `2**96` to `0`.
④ EigenLayer receives the submission with 0 multiplier assigned to a strategy.

Result: This could result in a submission of rewards using an invalid multiplier, possibly reverting it to zero.

Recommendation

Several approaches can be taken to address this issue:

1. Update the type of `multipliers` to `mapping(address => uint96)` and adjust the parameters of the functions that set these multipliers. It is important to review any existing AVS instances to determine if they have multipliers set above `2**96`, and to update these values as necessary.
2. Implement a check to determine if `votingPowerMultiplier` exceeds `2**96` and abort the submission process. This cancellation should be managed by the `AVSGovernance` contract, which must call back the clearance function with zeroed claimed fee amounts.

References

1. SCSVS G4: Business Logic

[OTNC-9a405548-L02] Duplicated operators in rewards submission

LOW FIXED

Retest (2025-03-11)

The vulnerability has been fixed. When the operator is unregistered, the connection between their address and payment details (`operatorsIdsByAddress`) is not cleared but it's set as inactive.
Later, when they register again, the connection is used to override new fee to be claimed and the payment status. After it's set, the connection is updated to the new one.
CommitID: *7ca4b766b4dd51b1a6a6b6da1b142764eed4eb8b*

Affected files

- AttestationCenter.sol#L203-L217

Description

The `OperatorDirectedRewardsSubmission` struct submitted to EigenLayer undergoes validation, and the transaction will revert if any validation rules are not satisfied. One of the rules is that the operator addresses must be unique.

An operator can make their address appear twice in the list. The operator has to first register and collect some fee to claim.

Then, the operator:

1. unregisters,
2. registers again,
3. and finally claim some fee in the same payment window (a week by default).

When the AVS manager request another reward payout, the operator address will be duplicated in the submission to EigenLayer, leading to a revert in the call to `RewardsCoordinator` contract.

Result: The standard rewards distribution process (as recommended by the CLI) fails, resulting in gas loss for the AVS manager.

Recommendation

Aggregate the fee to claim value for the operator when preparing the list of rewarded operators, but only for those entries that meet other requirements (e.g. the payment status is `REDEEMED`).

References

1. SCSVS G4: Business Logic

[OTNC-9a405548-L03] Double add of the default LayerZero endpoint

LOW FIXED

Retest (2025-03-11)

The vulnerability has been fixed. The `setLzEids` function has been removed.
CommitID: *043762597b1904bd31754900fd4eba5f9a5e7d2a*

Affected files

- L1MessageHandler.sol#L97-L101

Description

The `setLzEids` function is designed to add an existing `lzEid` (LayerZero destination endpoint) as the first entry in a new `lzEids` list, streamlining the transition to a multi-endpoint setup.

However, unlike the `setNewSupportedL2` function, the `setLzEids` function does not verify whether the `lzEid` is already present in the `lzEids` list. If the function is inadvertently called multiple times, it may add the same default endpoint ID repeatedly. This could result in sending duplicate cross-chain messages to the default chain, such as registering the same operator multiple times in the `AttestationCenter`.

Result: Duplication of LayerZero endpoint ID and sending identical cross-chain messages multiple times to the same `AttestationCenter` contract.

Recommendation

Implement a validation check within the function to ensure that the `lzEid` does not already exist in the `lzEids` list, and revert the transaction if a duplicate is detected.

References

1. SCSVS G4: Business Logic

6. Recommendations

[OTNC-9a405548-R01] Include L2 identifier in emitted payment request events

INFO NOT IMPLEMENTED

Retest (2025-05-11)

The recommendation has not been implemented.

Description

The `AVSGovernance` contract processes payment requests from `AttestationCenter` contracts operating across several Layer 2 (L2) networks. Each request includes information such as operators and the last task number. However, the current implementation lacks an indication of which L2 network the request originated from, leaving listeners unaware of the source L2 that is distributing rewards.

Recommendation

- Include the L2 identifier in the emitted events (`NativeCoinNotSupportedForEigenRewards`, `RewardsCoordinatorReverted`) to provide clarity on the source of the events.
- Introduce an event that verifies the distribution of rewards through EigenLayer, as well as directly through Othentic.

References

1. SCSVS G1: Architecture, design and threat modeling

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.

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com