Aligned Layer

Security Assessment

August 2024

FUZZING LABS

Contents

About FuzzingLabs

Founded in 2021 and headquartered in Paris, FuzzingLabs is a cybersecurity startup specializing in vulnerability research, fuzzing, and blockchain security. We combine cutting-edge research with hands-on expertise to secure some of the most critical components in the blockchain ecosystem.

At FuzzingLabs, we aim to uncover and mitigate vulnerabilities before they can be exploited. Over the past year, our tools and methodologies have identified hundreds of vulnerabilities in essential blockchain components, such as RPC libraries, cryptographic systems, compilers, and smart contracts. We collaborate with leading protocols and foundations to deliver open-source security tools, continuous audits, and comprehensive fuzzing services that help secure the future of blockchain technology.

If you're interested, we have a blog available at fuzzinglabs.com and an X account @FuzzingLabs. You can also contact us at contact@fuzzinglabs.com.

Executive Summary

Goals

The primary goal of this pre-audit is to gain an initial understanding of the Aligned Layer project and assess the overall security posture of its codebase. The objectives are structured into two phases:

1. **Initial Assessment (Quick Dive):**
    - Perform a rapid analysis of the code to detect straightforward patterns and vulnerabilities. This phase will focus on identifying low-hanging fruits, such as common security flaws, poor coding practices, or weak implementation patterns.
    - The intent is to familiarize ourselves with the architecture and identify areas that may need further attention in the next phase.
2. **In-Depth Analysis (Deep Dive):**
    - Conduct a more thorough examination of the codebase to identify more complex patterns, logical bugs, or security vulnerabilities that may not be immediately apparent. This includes looking for subtle issues related to control flow, data validation, and other complex logic errors.
    - Writing fuzzing harnesses will be a key part of this stage to automate the detection of edge cases, and unexpected behaviors, and to expose vulnerabilities that may not be easily found through manual analysis.
3. **Focus on Race Conditions:**
    - As part of the audit, special attention will be given to the potential for race conditions within the system. This includes reviewing concurrency and thread management within the code to ensure that no vulnerabilities exist that could lead to unexpected behavior, security flaws, or system crashes due to improper synchronization.

This pre-audit will help form the basis for a comprehensive security assessment of Aligned Layer, preparing for future audit stages with a clear understanding of both potential and identified risks.

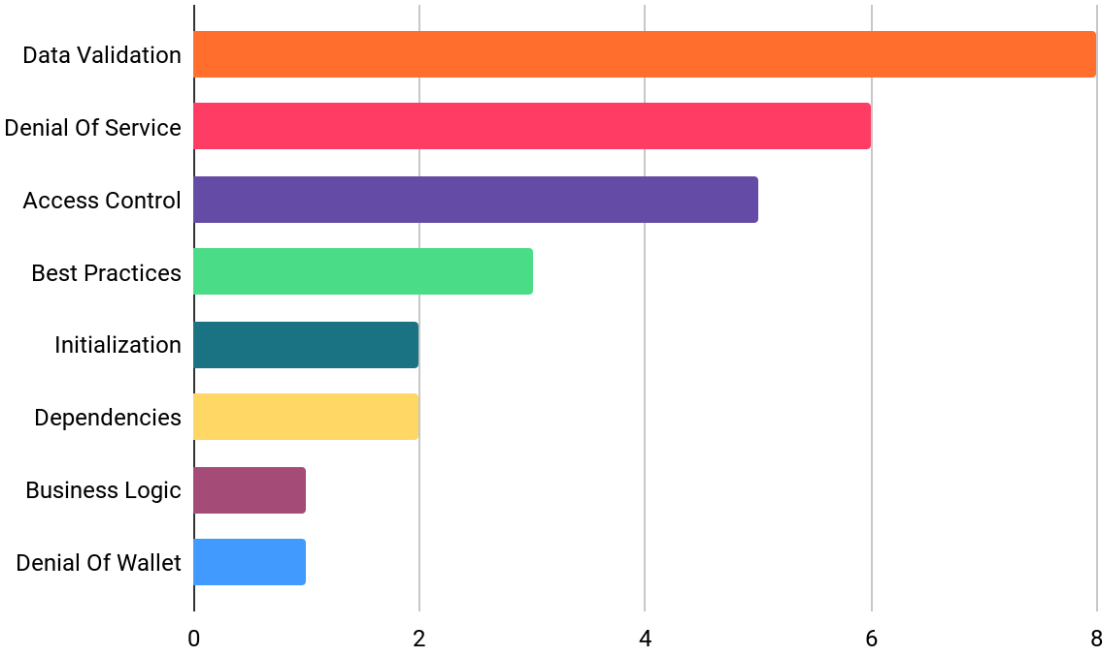
Limitations

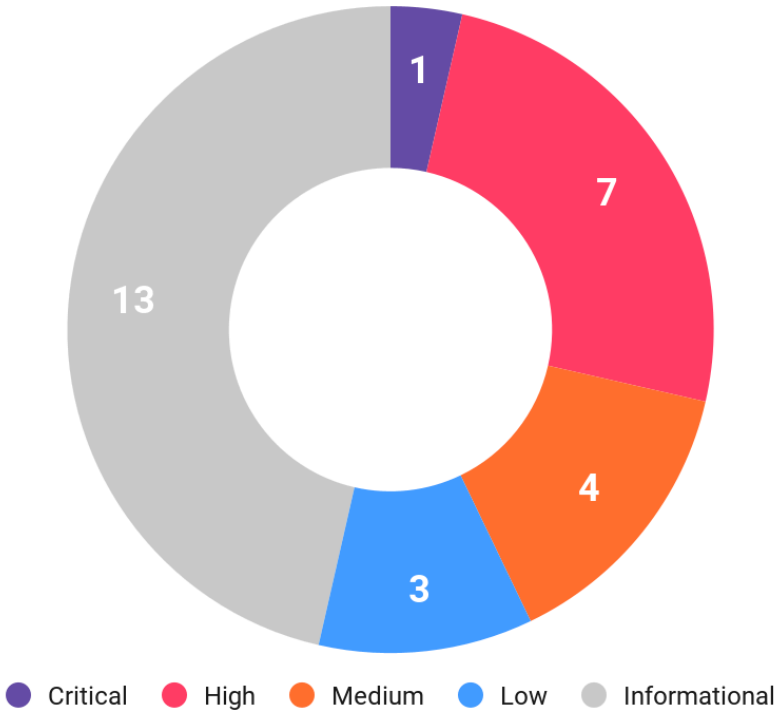
The audit of Aligned Layer was conducted with full white card access, providing our team unrestricted visibility into the system's architecture, codebase, and operational environment. However, certain limitations were agreed upon with the client, as outlined below:

- **Exclusion of Cryptographic Analysis**: Although the audit scope included comprehensive assessments, the client explicitly requested the exclusion of cryptographic analysis. As such, we did not conduct any evaluations or validations of cryptographic primitives, protocols, or implementations in the system.
- **Explorer Analysis Not Prioritized**: Another area that was deemed non-essential by the client for this particular audit was the functionality and security of the explorer. As a result, our focus was not directed toward the analysis of the explorer's robustness, user interface, or potential vulnerabilities in this component.

These restrictions were client-driven and aligned with their specific risk tolerance and project needs. While they represent areas not covered in this audit, they can be revisited in future assessments if required.

Summary of Findings





ID	Title	Target	Rating
1	Multiple OOBs in the verify function of the operator	Operator	Critical
2	The operator is vulnerable to OOM when fetching a batch	Operator	High
3	Bypass operator length check when fetching a batch	Operator	High
4	Unsafe pointer casting without length verification	Operator	High
5	BatcherPaymentService is vulnerable to DoS and grief via frontrunning	Contracts	High
6	Aggregator fee can be MEV	Contracts	High
7	Multiple OOB in batcher halo2_ipa and halo2_kzg verification	Batcher	High
8	A batch can be submitted without paying the batcher fee	Contracts	High
9	OOM explorer when fetching batch	Explorer	Medium
10	Amplified denial of wallet	S3	Medium
11	verify_merkle_tree_batch_FFI can panic	Operator, Batcher	Medium

12	An Aligned user can be frontrun when creating a task leading to a user DoS	**Contracts**	**Medium**
13	BatcherPaymentService initialize can be frontrun	**Contracts**	**Low**
14	AlignedLayerServiceManager fund can't be withdrawn	**Contracts**	**Low**
15	Metrics package is vulnerable to Slowloris attack	**Aggregator, Operator**	**Low**
16	BatcherPaymentService ecrecover missing checks	**Contracts**	**Info**
17	BatcherPaymentService switch to Ownable2StepUpgradeable	**Contracts**	**Info**
18	BatcherPaymentService incomplete initialization	**Contracts**	**Info**
19	BatcherPaymentService missing address(0) check in initializer	**Contracts**	**Info**
20	AlignedLayerServiceManager initialization can be frontrun	**Contracts**	**Info**
21	AlignedLayerServiceManager incomplete initialization	**Contracts**	**Info**
22	AlignedLayerServiceManager missing address(0) check in constructor and initializer	**Contracts**	**Info**
23	Dockerfiles need to be hardened	**Dockerfiles**	**Info**

24	Inconsistent Serialization in the Marshal & Unmarshal Functions	**Operator**	**Info**
25	Verify groth16 proof allows malleability	**Batcher, Operator**	**Info**
26	Missing content security policy in router.ex	**Explorer**	**Info**
27	Aggregator has a data race during BLS aggregation	**Aggregator**	**Info**
28	Unmaintained and yanked crates	**Batcher, Operator**	**Info**

Project Summary

About Aligned Layer

Aligned Layer is a decentralized network designed to provide fast, efficient, and low-cost verification of zero-knowledge (ZK) and validity proofs on the Ethereum blockchain. It seeks to overcome the limitations of current blockchain verification systems, which are often slow and expensive due to the need for nodes to re-execute each transaction. By offloading the computation off-chain, Aligned Layer enables faster proof verification and significantly reduces costs.

The platform operates in two modes: **fast mode** and **aggregation mode**. In **fast mode**, a subset of Ethereum's validators (operators) verify proofs in parallel, posting the results to Ethereum only after reaching a two-thirds consensus. This approach allows Aligned Layer to achieve verification speeds of over 1,000 proofs per second, far outpacing Ethereum's native capabilities. In **aggregation mode**, the system further compresses proofs for even greater efficiency at scale.

Aligned Layer is particularly useful for applications requiring large-scale computations with the assurance that they are verified in a trustless manner. It supports various proof systems, making it versatile for developers working on ZK technology, including zk-rollups, identity protocols, and decentralized applications that demand high throughput and low latency.

The goal of Aligned Layer is to accelerate Ethereum's roadmap, supporting the adoption of verifiable computation and making ZK verification accessible and affordable to a wider range of developers.

Scope

During this assessment, we conducted our work on two separate branches. This approach was necessary due to the project being in both pre-audit and audit stages.
The branches we focused on were identified by the following commit hashes:

- 81e1ae4ff6cdddca28808ac071d4b1ae72793346 pre-audit
- 325aef8c3f54ec596b4733956a8ac487d5535fc3 audit

This dual-branch strategy allowed us to address the distinct requirements of each stage effectively.

The audit for Aligned Layer focused on the following critical components, identified as essential for system stability and security:

- **Foreign Function Interface (FFI)**: As the FFI manages interactions between external code and the internal system, it was prioritized to identify risks such as improper memory handling or input validation issues.
- **Batcher**: This component groups proofs into batches for verification. Its role is critical for efficiency and accuracy, so we focused on ensuring proper batch formation and error handling.
- **Operator**: The Operator oversees task execution within the system, coordinating various processes. Its interactions with the Batcher were evaluated to ensure synchronization and reliability in managing proofs.
- **Conflict Between Batcher and Operator**: We examined potential conflicts between these two components, especially regarding timing, transaction order, and any possible deadlocks that could hinder the system's operations.
- **Privacy**: Since the system does not implement privacy measures, we excluded data leak checks from our audit.
- **Single Aggregator Assumption**: The system currently operates with only one aggregator, and the audit was performed with this assumption, simplifying the analysis of data flow and aggregation risks.
- **ServiceManager**: This component handles service coordination and process management. Its audit focused on ensuring robust service lifecycle management, including safe startup, shutdown, and failure recovery processes.

Audit Timeline

The security audit for Aligned Layer was conducted over a total of 40 man-days by two engineers, Nabih Benazzouz and Mohammed Benhelli, each working for 20 days. The timeline was structured as follows:

1. **Initial Discovery, Deployment, and Testing:**
    - The audit began with familiarizing ourselves with Aligned Layer's architecture, deploying the platform in a controlled environment, and conducting initial tests.
    - This phase also included a preliminary analysis of the attack surface, identifying critical components and areas for deeper exploration.
2. **Static Code Analysis and Fuzzing:**
    - A comprehensive static audit of the codebase was performed, focusing on potential vulnerabilities, security flaws, and code patterns that may expose the system to threats.
    - In parallel, we developed fuzzing harnesses to automate the process of input validation, edge case detection, and vulnerability identification.
    - At the end of this phase, reports were compiled detailing the findings from the static analysis and fuzzing exercises
3. **Smart Contract Audit:**
    - A thorough audit of all smart contracts in the Aligned Layer ecosystem was conducted to ensure that they adhered to best security practices.
    - This involved checking for vulnerabilities specific to blockchain environments such as reentrancy, overflow/underflow issues, and improper access control mechanisms.
4. **Dependency Audit:**
    - A quick audit of all external libraries and dependencies was performed to check for any known vulnerabilities, outdated versions, or weak points that could affect the overall security of the system.
5. **Proof-of-Concept (PoC) for Batcher Fuzzer:**

- A specialized fuzzing tool was developed as a proof-of-concept for the batcher component, allowing for automated detection of issues related to batch processing and potential vulnerabilities specific to the component.

This structured approach allowed us to cover both the high-level architecture and the critical components of Aligned Layer, ensuring a comprehensive security review.

Threat Model

Based on the attack surface presented in the Aligned Layer diagram, we were able to construct an enhanced attack surface and corresponding threat model. This model identifies critical risks associated with components like the **Batcher**, **Contracts**, **Operator**, **Aggregator**, and **FFI**. By analyzing these interactions, we highlighted potential conflicts, vulnerabilities in input validation, synchronization issues, and possible points of external manipulation. The threat model provides a structured view of these risks, facilitating focused security assessments and suggesting mitigations tailored to each component's role within the system.
While this audit focused on specific aspects, it's important to note that potential vulnerabilities originating from the EigenLayer security model **were not within the scope of this assessment**.
These EigenLayer-related risks, though not examined in this audit, may warrant consideration in future security evaluations.

The following considerations were taken into consideration when creating the threat model.

DevSecOps

- How secrets are handled in production
- ACL for production infra
- CI/CD are safe
- How can an operator join the AVS?
- Ensure that the test private keys are useless in other contexts
- Keys/Multisig usage and ACLs for upgrade/withdrawal?

CryptoEconomic

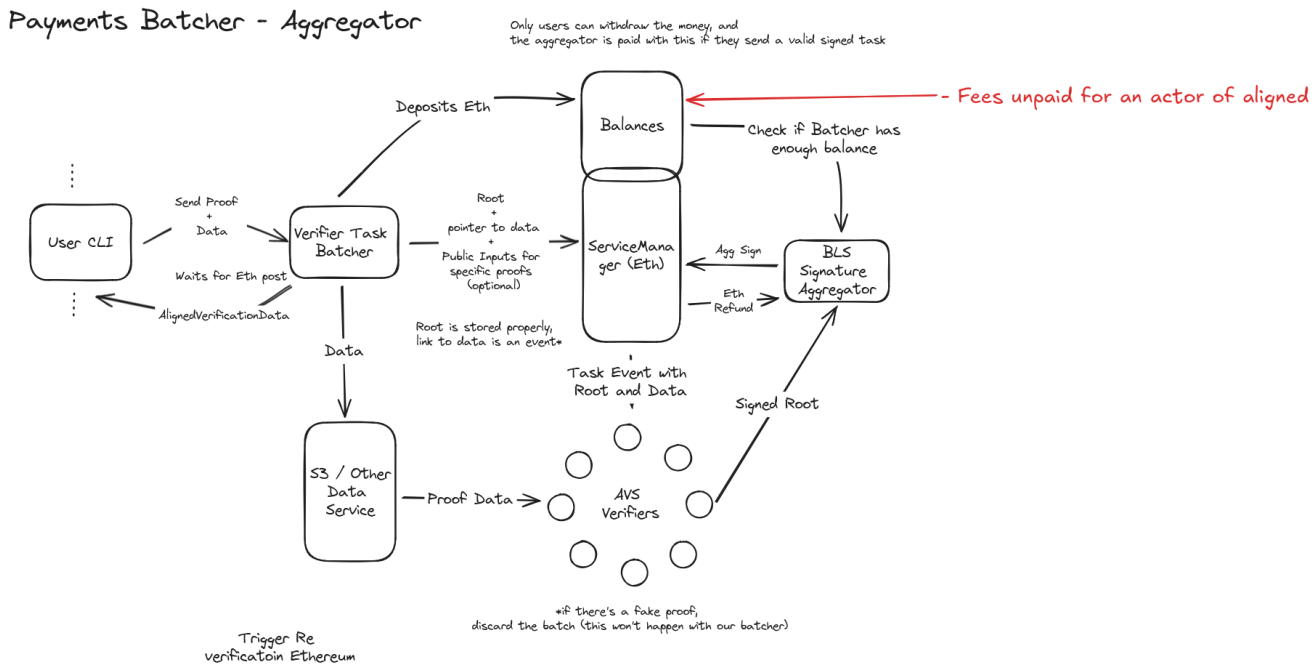
- Can user funds be lost? (locked...)
- Exposure to unintended slashing?
- Fees unpaid for an actor of aligned (batcher, aggregator...)
- Onchain DoS

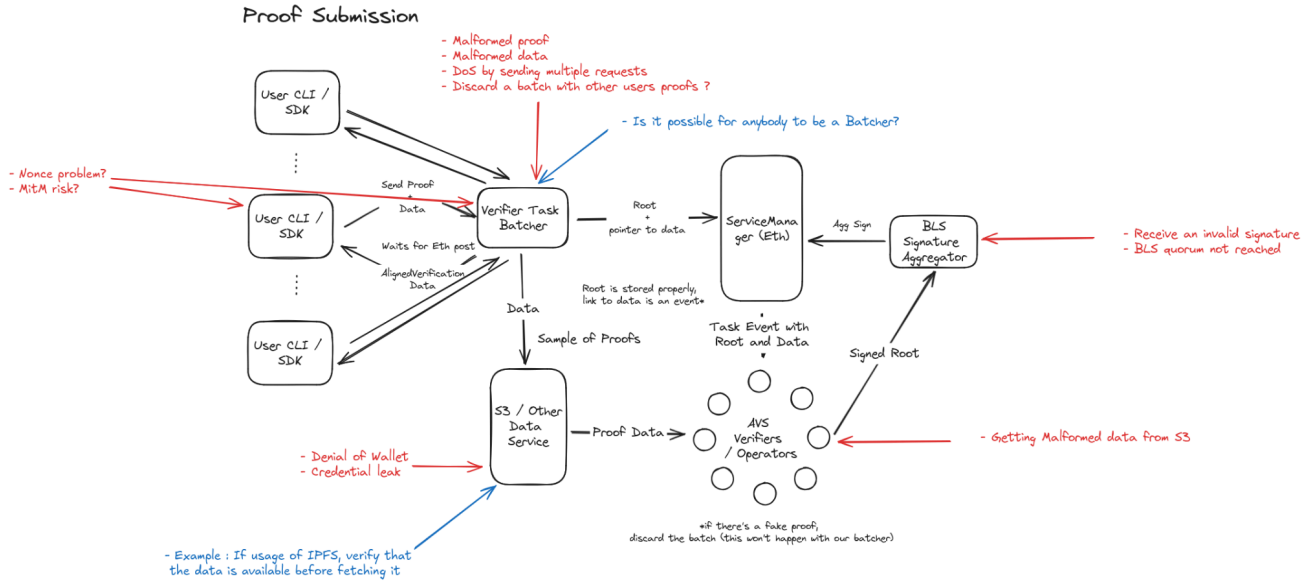
Explorer

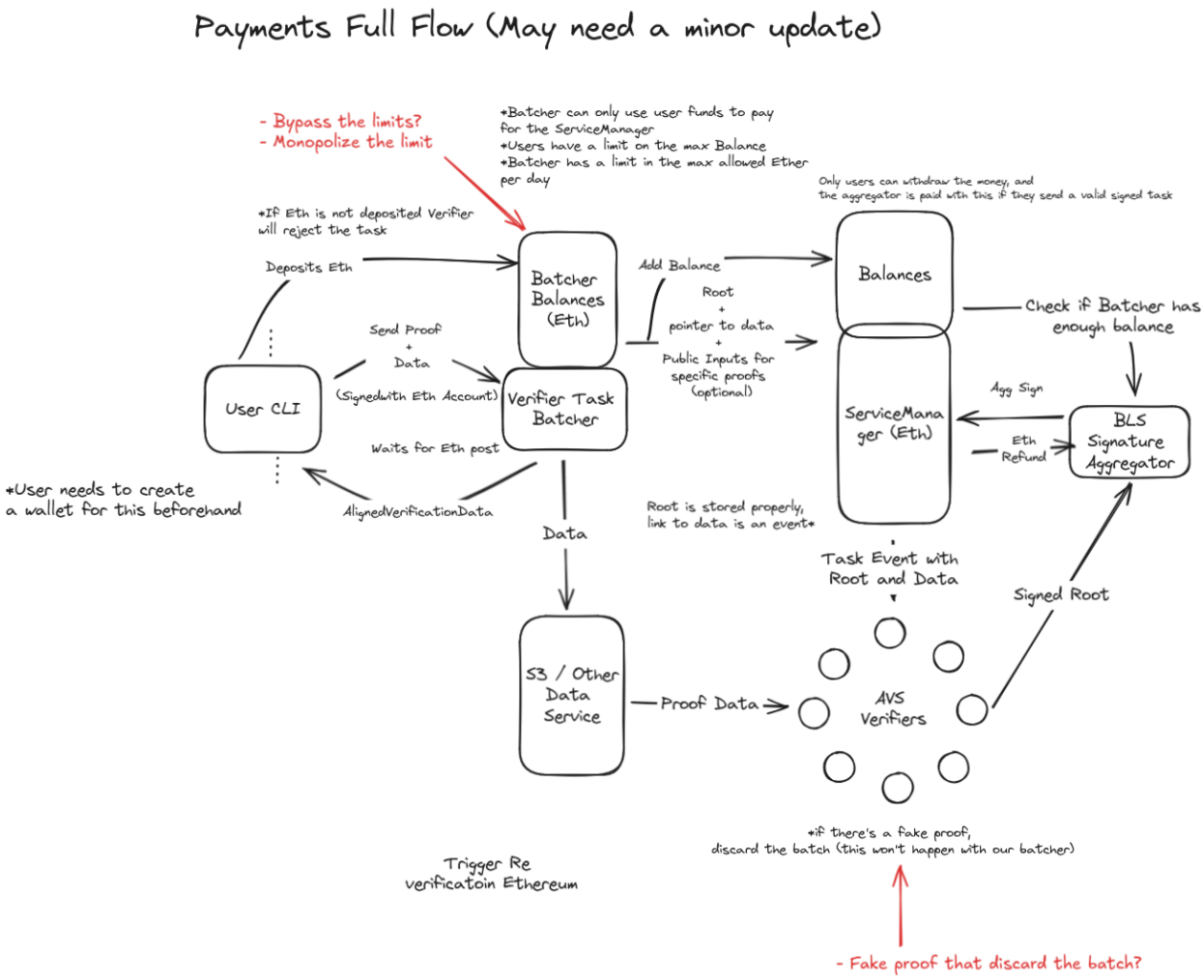
- XSS and other client side vectors
- Fake data can be displayed in the explorer (SQLi...)

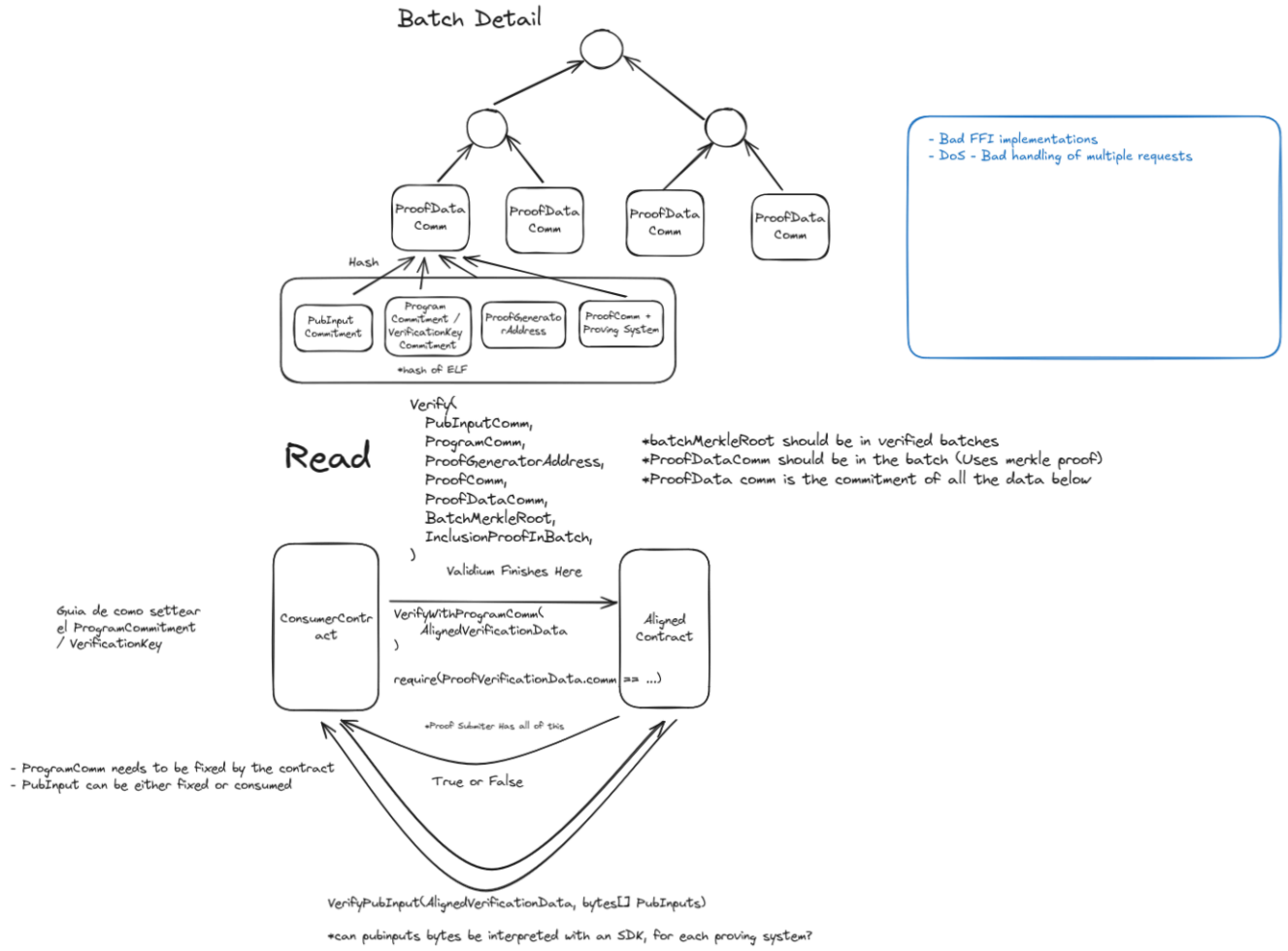
Metrics

- Bad configuration of metrics stack?
- Ex: Prometheus open on public IP






Payments Full Flow (May need a minor update)
- Bypass the limits?
- Monopolize the limit
*Batcher can only use user funds to pay for the ServiceManager
*Users have a limit on the max Balance
*Batcher has a limit in the max allowed Ether per day
Only users can withdraw the money, and the aggregator is paid with this if they send a valid signed task
*If Eth is not deposited Verifier will reject the task
Deposits Eth
Batcher Balances (Eth)
Add Balance
Balances
Root + pointer to data + Public Inputs for specific proofs (optional)
Check if Batcher has enough balance
Send Proof + Data
(Signedwith Eth Account)
User CLI
Verifier Task Batcher
Agg Sign
ServiceManager (Eth)
Eth Refund
BLS Signature Aggregator
Waits for Eth post
*User needs to create a wallet for this beforehand
AlignedVerificationData
Root is stored properly, link to data is an event*
Data
Task Event with Root and Data
Signed Root
S3 / Other Data Service
Proof Data
AVS Verifiers
*if there's a fake proof, discard the batch (this won't happen with our batcher)
Trigger Re verificatoin Ethereum
- Fake proof that discard the batch?


Batch Detail
ProofData Comm
ProofData Comm
ProofData Comm
ProofData Comm
Hash
PubInput Commitment
Program Commitment / VerificationKey Commitment
ProofGeneratorAddress
ProofComm + Proving System
*hash of ELF
- Bad FFI implementations
- DoS - Bad handling of multiple requests
Read
Verify(
PubInputComm,
ProgramComm,
ProofGeneratorAddress,
ProofComm,
ProofDataComm,
BatchMerkleRoot,
InclusionProofInBatch,
)
*batchMerkleRoot should be in verified batches
*ProofDataComm should be in the batch (Uses merkle proof)
*ProofData comm is the commitment of all the data below
Validium Finishes Here
Guia de como settear el ProgramCommitment / VerificationKey
ConsumerContract
VerifyWithProgramComm(
AlignedVerificationData
)
require(ProofVerificationData.comm == ...)
Aligned Contract
*Proof Submiter Has all of this
True or False
- ProgramComm needs to be fixed by the contract
- PubInput can be either fixed or consumed
VerifyPubInput(AlignedVerificationData, bytes[] PubInputs)
*can pubinputs bytes be interpreted with an SDK, for each proving system?

Contracts

There are two contracts deployed on Ethereum:

- **AlignedLayerServiceManager** that handles users' payments to fund the verification of their proofs.
- **BatcherPaymentService** that handles the reception of new batches to Aligned, keeps their status on-chain and receives their response.

AlignedLayerServiceManager

ID	Threat	Impact
1	Increase batcher balance without a deposit	HIGH
2	Decrease batcher balance without validating a batch	HIGH
3	Invalid BLS signature can be submitted	HIGH
4	DoS when creating a task	MEDIUM
5	Grief of balance	MEDIUM

BatcherPaymentService

ID	Threat	Impact
1	Increase user balance without a deposit	HIGH
2	Decrease user without creating a batch	HIGH
3	Invalid user signature can be submitted	HIGH
4	Invalid Merkle tree can be submitted	HIGH
5	DoS when creating a task	MEDIUM
6	Grief of balance	MEDIUM

Batcher

One batcher with a fallback uses Cloudflare R2 to upload batches and has its own Eth RPC.
The batcher receives proofs from Users, bundles them, creates a Merkle Root, and submits them to the AlignedLayerServiceManager.
It provides Merkle Proofs to Users for verification, performs preliminary checks, and handles payment through its BatcherPaymentService.
The batch is stored for a week, with the link sent to the AlignedLayerServiceManager.

ID	Threat	Impact
1	Invalid batch can be submitted	HIGH
2	Wrong state sync with the contracts	HIGH
3	Invalid user signature can be submitted	HIGH
4	A batch bigger than the operator limit can be submitted	HIGH
5	DoS that shutdown the batcher	HIGH
6	Wrong calculation on gas per proof	MEDIUM
7	Wrong calculation on aggregator fees	MEDIUM

Batcher-cli

CLI tool helps an Aligned user to submit proof to the batcher.

ID	Threat	Impact
1	Error in the user's deposit amount	LOW
2	Wrong nonce sync when signing	LOW

Operator

Multiple operators (20-100) are deployed having their own Eth RPC and metrics endpoint. Operators, as Eigenlayer restakers, verify ZK Proofs and enhance system security.
They process AlignedLayerServiceManager's batch events, download and verify proofs against the Merkle Root, and execute verification programs. After batch verification, operators sign their response with a BLS signature and send it to the aggregator.

ID	Threat	Impact
1	Invalid batch can be validated	HIGH
2	Valid batch is discarded	HIGH
3	DoS that shutdown the operator	HIGH
4	A batch bigger than the operator limit can be validated	HIGH
5	Dos that shutdown the metrics	LOW

Aggregator

One Aggregator with a fallback has its own Eth RPC. The aggregator collects the operator's BLS signatures.
When the quorum of responses is reached, it will submit a task response with the aggregated signatures back to the AlignedLayerServiceManager.

ID	Threat	Impact
1	Invalid batch can be validated	HIGH
2	Valid batch is lost	HIGH
3	DoS that shutdown the aggregator	HIGH

Explorer

One Explorer is deployed, it keeps track of AlignedLayerServiceManager. It has an internal state of previous batches and actively listens for new batches and their responses.
The explorer then displays this information for users to visualize the submitted batches, their states, and more useful information in real-time.

ID	Threat	Impact
1	XSS and other vectors that target a user	MEDIUM
2	DoS that shutdown the explorer	MEDIUM
3	Wrong data is indexed	MEDIUM

Findings

I. Multiple OOBs in the verify function of the operator

Rating	**Critical**
ID	**FL-AL-1**
Category	**Data Validation**
Target	**Operator**
Commit	**81e1ae4**

Description

We identified multiple out-of-bounds (OOB) vulnerabilities within the verify function. This function manipulates arrays without performing adequate size checks. For example, the code snippet copy(csLenBuffer, paramsBytes[:4]) is used without verifying the size of the paramsBytes array, leading to potential OOB errors.

The vulnerability in the verify function arises due to the lack of length checks for various arrays or slices before they are manipulated. This oversight can lead to out-of-bounds errors.

operator/pkg/operator.go#L213

```
...
            csLenBuffer := make([]byte, 4)
            copy(csLenBuffer, paramsBytes[:4])
            csLen := (uint32)(binary.LittleEndian.Uint32(csLenBuffer))

            // Deserialize vkLen
            vkLenBuffer := make([]byte, 4)
            copy(vkLenBuffer, paramsBytes[4:8])
            vkLen := (uint32)(binary.LittleEndian.Uint32(vkLenBuffer))

            // Deserialize ipaParamLen
            IpaParamsLenBuffer := make([]byte, 4)
            copy(IpaParamsLenBuffer, paramsBytes[8:12])
            IpaParamsLen := (uint32)(binary.LittleEndian.Uint32(IpaParamsLenBuffer))
...
and more
```

Since the verificationData utilized in this function is retrieved from an S3 bucket, there is a possibility that a user could send malformed data to the S3, triggering these vulnerabilities.

operator/pkg/operator.go#L175

```
...
func (o *Operator) ProcessNewBatchLog(newBatchLog
*servicemanager.ContractAlignedLayerServiceManagerNewBatch) error {
    ...
      verificationDataBatch, err := o.getBatchFromS3(newBatchLog.BatchDataPointer,
newBatchLog.BatchMerkleRoot)
    ...
      for _, verificationData := range verificationDataBatch {
            go func(data VerificationData) {
                  defer wg.Done()
                  o.verify(data, results)
...
```

Additionally, a user could exploit this vulnerability by providing their URL in the smart contract via AlignedServiceManager.createNewTask.

contracts/src/core/AlignedLayerServiceManager.sol#L56

```
...
contract AlignedLayerServiceManager is
    ...
{
    ...
    // ! Everyone can call this function
    function createNewTask(
        bytes32 batchMerkleRoot,
        string calldata batchDataPointer
    ) external payable {
        ...
    }
}
```

Recommendations

- Implement strict input validation to ensure all input fields are the expected size before processing.
- Add a modifier to the *createNewTask* function to restrict access to only authorized users.

II. The operator is vulnerable to OOM when fetching a batch

Rating	**High**
ID	**FL-AL-2**
Category	**Denial Of Service**
Target	**Operator**
Commit	**81e1ae4**

Description

The getBatchFromS3 method of Operator is vulnerable to an OOM attack via a crafted gzip file, for example.

operator/pkg/s3.go#L40

```
package operator
...
func (o *Operator) getBatchFromS3(batchURL string, expectedMerkleRoot [32]byte)
([]VerificationData, error) {
   ...
   // ! Read until EOF
   batchBytes, err := io.ReadAll(resp.Body)
   ...
}
```

io.ReadAll reads from the resp.Body until an EOF is encountered. If the resp.Body is a gzip file, it will be decompressed in memory, which can lead to an OOM attack.

Recommendations

- Using io.LimitReader to limit the amount of data read from the response body.
- Using http.MaxBytesReader could also solve this issue.

III. Bypass operator length check when fetching a batch

Rating	**High**
ID	**FL-AL-3**
Category	**Data Validation**
Target	**Operator**
Commit	**81e1ae4**

Description

The getBatchS3 method of Operator length check can be bypassed via a crafted HTTP response for example.

operator/pkg/s3.go#L14

```
package operator
...
func (o *Operator) getBatchFromS3(batchURL string, expectedMerkleRoot [32]byte)
([]VerificationData, error) {
	...
	resp, err := http.Head(batchURL)
	if err != nil {
		return nil, err
	}

	// Check if the response is OK
	if resp.StatusCode != http.StatusOK {
		return nil, fmt.Errorf("error getting Proof Head from S3: %s", resp.Status)
	}

	if resp.ContentLength > o.Config.Operator.MaxBatchSize {
		return nil, fmt.Errorf("proof size %d exceeds max batch size %d",
			resp.ContentLength, o.Config.Operator.MaxBatchSize)
	}
    ...
}
```

Making a HEAD request to the batch URL to get the content length can be bypassed if the server does not return the correct content length.

Recommendations

- One possible way could be adding access control when creating a task. This way, the operator would trust the batch url.
- Another option could be allowing a whitelist of domains that are allowed to host a batch file.

IV. Unsafe pointer casting without length verification

Rating	**High**
ID	**FL-AL-4**
Category	**Data Validation**
Target	**Operator**
Commit	325aef8

Description

During an audit and fuzzing of several operator functions, we identified multiple instances where buffers are cast to unsafe.Pointer without verifying their lengths.
This pattern can lead to potential out-of-bounds (OOB) vulnerabilities. The following code snippets demonstrate this unsafe practice.

```
39  proofPtr := (*C.uchar)(unsafe.Pointer(&proofBuffer[0]))
40  csPtr := (*C.uchar)(unsafe.Pointer(&csBuffer[0]))
41  vkPtr := (*C.uchar)(unsafe.Pointer(&vkBuffer[0]))
42  ipaParamPtr := (*C.uchar)(unsafe.Pointer(&ipaParamBuffer[0]))
43  publicInputPtr := (*C.uchar)(unsafe.Pointer(&publicInputBuffer[0]))
```

Recommendations

- Input Validation: Implement strict input validation to ensure all buffers are of the expected size before casting them to unsafe.Pointer.
- Length Checks: Add length checks before accessing the first element of any buffer to prevent out-of-bounds errors.

V. BatcherPaymentService is vulnerable to DoS and grief via frontrunning

Rating	**High**
ID	**FL-AL-5**
Category	**Denial Of Service**
Target	**BatcherPaymentService**
Commit	325aef8

Description

We identified a vulnerability in the BatcherPaymentService contract. The checkMerkleRootAndVerifySignatures has public visibility and decreases the balance of the users.
This leads to different DoS attacks:
- A malicious user can decrease the balance of the user.
- A malicious user can increase the nonce of the user.
- A batch can be discarded by a malicious user.

An attacker can also craft a specific batch, leading to a grief attack on some targeted users.

contracts/src/core/BatcherPaymentService.sol#L173

```
...
contract BatcherPaymentService is
...
{
    ...
    function checkMerkleRootAndVerifySignatures(
        ...
    ) public {
      ...
         verifySignatureAndDecreaseBalance(
           ...
         );
      }
      ...
   }
}
```

Recommendations

- Make checkMerkleRootAndVerifySignatures private and add a view function to check the signature.

VI. Aggregator fee can be MEV

Rating	High
ID	FL-AL-6
Category	Access Control
Target	AlignedLayerServiceManager
Commit	325aef8

Description

This MEV vulnerability is because the AlignedLayerServiceManager contract does not have a mechanism to prevent frontrunning attacks when responding to a task and the aggregator fees are calculated based tx.gasprice making it possible to extract all batchersBalances[batchesState[batchMerkleRoot].batcherAddress] since the tx.gasprice have no upper limit.

contracts/src/core/AlignedLayerServiceManager.sol#L82

```
...
contract AlignedLayerServiceManager is
    ...
{
    ...
    function respondToTask(
        bytes32 batchMerkleRoot,
        NonSignerStakesAndSignature memory nonSignerStakesAndSignature
    ) external {
      ...
      uint256 txCost = (initialGasLeft - finalGasLeft + 70000) * tx.gasprice;
      ...
      batchersBalances[
        batchesState[batchMerkleRoot].batcherAddress
      ] -= txCost;
      payable(msg.sender).transfer(txCost);
    }
}
```

A potential optimized exploit would do the following:

- Listen to the pending transactions

- Wait for the respondToTask transactions
- Read the batcher balance using balanceOf function
- Calculate the txCost and send the transaction with the optimal tx.gasprice that will extract all the balance

Recommendations

- We suggest implementing a mechanism to prevent frontrunning attacks when responding to a task. This can be achieved by adding a modifier that restricts the respondToTask function to a whitelist of addresses that correspond to aggregators for example.

VII. Multiple OOB in batcher halo2_ipa and halo2_kzg verification

Rating	**High**
ID	**FL-AL-7**
Category	**Data Validation**
Target	**Batcher**
Commit	**81e1ae4**

Description

The verify_halo2_kzg and verify_halo2_ipa functions manipulate arrays without performing size verification. For instance, it uses the code snippet let cs_len_buf: [u8; 4] = verification_key[..4] without checking the size of the array.

The vulnerable functions are:

- batcher/aligned-batcher/src/halo2/ipa/mod.rs#L23
- batcher/aligned-batcher/src/halo2/kzg/mod.rs#L26

Recommendations

- Implement strict input validation to ensure all input fields are of the expected size before processing.

VIII. A batch can be submitted without paying the batcher fee

Rating	**High**
ID	**FL-AL-8**
Category	**Access Control**
Target	**AlignedLayerServiceManager**
Commit	**81e1ae4**

Description

An Aligned Layer user can submit a batch without paying the required fees to the batcher. This vulnerability can cause various issues:
- The economic model of the platform can be affected.
- Arbitrary URLs can be submitted.

The root cause of the vulnerability is that the AlignedLayerServiceManager createNewTask function can be called by anyone instead of allowing only the BatcherPaymentService.

contracts/src/core/AlignedLayerServiceManager.sol#L56

```
...
contract AlignedLayerServiceManager is
    ...
{
    ...
    function createNewTask(
        bytes32 batchMerkleRoot,
        string calldata batchDataPointer
    ) external payable {
        ...
        if (msg.value > 0) {
            batchersBalances[msg.sender] += msg.value;
        }
        ...
    }
}
```

Sending a transaction to the createNewTask function with only the aggregator fees will allow the user to validate a batch without paying all the required fees.

Recommendations

- Add a modifier to the createNewTask function to restrict its access to the BatcherPaymentService, assuming only one BatcherPaymentService will be deployed.

IX. OOM explorer when fetching batch

Rating	**Medium**
ID	**FL-AL-9**
Category	**Denial Of Service**
Target	**Explorer**
Commit	325aef8

Description

The fetch_batch_data_pointer function in the explorer is vulnerable to an OOM attack. The function reads the entire response body without any limitation.

explorer/lib/explorer_web/live/utils.ex#L141

```
...
  def fetch_batch_data_pointer(batch_data_pointer) do
    case Finch.build(:get, batch_data_pointer) |> Finch.request(Explorer.Finch) do
      {:ok, %Finch.Response{status: 200, body: body}} ->
        case Jason.decode(body) do
          {:ok, json} -> {:ok, json}
          {:error, reason} -> {:error, {:json_decode, reason}}
        end

      {:ok, %Finch.Response{status: status_code}} ->
        {:error, {:http_error, status_code}}

      {:error, reason} ->
        {:error, {:http_error, reason}}
    end
  end
...
```

The Finch.build(:get, batch_data_pointer) |> Finch.request(Explorer.Finch) function reads the entire response body without any limitation.

Recommendations

- Set a timeout and a read limit for the request to prevent an OOM attack.

X. Amplified denial of wallet

Rating	**Medium**
ID	**FL-AL-10**
Category	**Denial Of Wallet**
Target	**S3**
Commit	**81e1ae4**

Description

The s3 bucket can be attacked with an amplified denial of wallet.
The root cause of the issue is that the s3 bucket is public and a batch can have a size up to 256MB. This can be used to amplify the cost of the attack.
An attacker can request a large number of files from the s3 bucket with range requests. The attacker can request to retrieve a part of a file, but not the entire file.
By quickly canceling such requests, an attacker can request parts of a file without downloading all the data.
Due to the way AWS calculates egress costs the transfer of the entire byte range is billed, even though the requests are canceled before the data is sent.
For example, an attacker can request a range from a 256MB file, cancel the request, and repeat this process at scale on a VPS with unlimited bandwidth like Hetzner.
This would lead to high egress costs for the owner of the s3 bucket.

Recommendations

- Use a service for file storage that does not charge for egress costs.

XI. verify_merkle_tree_batch_FFI can panic

Rating	**Medium**
ID	**FL-AL-11**
Category	**Data Validation**
Target	**Operator, Batcher**
Commit	**81e1ae4**

Description

During the process of auditing the code and developing fuzzing harnesses, we found that a malformed JSON input can cause the verify_merkle_tree_batch_ffi to panic due to a subtraction with overflow. This can be triggered by an attacker to cause a denial of service.

crypto/src/merkle_tree/utils.rs#L38

```
...
// ! CAUTION !
// Make sure n=nodes.len()+1 is a power of two, and the last n/2 elements (leaves) are
populated with hashes.
// This function takes no precautions for other cases.
pub fn build<B: IsMerkleTreeBackend>(nodes: &mut [B::Node], leaves_len: usize)
where
    B::Node: Clone,
{
    let mut level_begin_index = leaves_len - 1;
    ...
        level_end_index = level_begin_index - 1;
    ...
}
```

If the leaves_len is 1, the level_begin_index will be set to 0 which will cause a panic when trying to subtract 1 from it since it's an unsigned integer.

Recommendations

- Input Validation: Ensure that the proof field in the JSON input is properly validated before processing it.

XII. An Aligned user can be frontrun when creating a task leading to a user DoS

Rating	**Medium**
ID	**FL-AL-12**
Category	**Denial Of Service**
Target	**AlignedLayerServiceManager**
Commit	**81e1ae4**

Description

We discovered a DoS vulnerability in the createNewTask function of AlignedLayerServiceManager. This vulnerability enables an attacker to frontrun a user when creating a task, resulting in a user DoS.
This vulnerability is possible because the createNewTask function checks if the batchesState[batchMerkleRoot] is not initialized.
A malicious user can frontrun a user by creating a task with the same batchMerkleRoot as the user and setting the batchesState[batchMerkleRoot] to true. This will cause the targeted user's createNewTask to fail.

contracts/src/core/AlignedLayerServiceManager.sol#L56

```
...
contract AlignedLayerServiceManager is
    ...
{
    ...
    function createNewTask(
        ...
    ) external payable {
        // ! Check if the batch was already submitted
        require(
            batchesState[batchMerkleRoot].taskCreatedBlock == 0,
            "Batch was already submitted"
        );
        ...
    }
}
```

Recommendations

- One possible way would be to store the batchesState[batchMerkleRoot] in a mapping with the user address as the key.
- Or using a hash of batchDataPointer and batchMerkleRoot as a key in the mapping to store the state of the batch.

XIII. BatcherPaymentService initialize can be frontrun

Rating	Low
ID	FL-AL-13
Category	Access Control
Target	BatcherPaymentService
Commit	325aef8

Description

This is a classic frontrun vulnerability. The BatcherPaymentService contract does not implement a way to prevent frontrunning during the initialization step.

contracts/src/core/BatcherPaymentService.sol#L49

```
...
contract BatcherPaymentService is
    ...
{
    ...
    function initialize(
        address _AlignedLayerServiceManager,
        address _BatcherPaymentServiceOwner,
        address _BatcherWallet
    ) public initializer {
        __Ownable_init(); // default is msg.sender
        __UUPSUpgradeable_init();
        _transferOwnership(_BatcherPaymentServiceOwner);

        AlignedLayerServiceManager = _AlignedLayerServiceManager;
        BatcherWallet = _BatcherWallet;
    }
    ...
}
```

Recommendations

- We recommend implementing access control checks in the BatcherPaymentService contract to prevent frontrunning during the initialization step.

XIV. AlignedLayerServiceManager fund can't be withdrawn

Rating	Low
ID	**FL-AL-14**
Category	**Business Logic**
Target	**AlignedLayerServiceManager**
Commit	325aef8

Description

When a user sends ether to the contract, the balance is updated in the batchersBalances mapping. However, there is no way to withdraw the balance from the contract.

contracts/src/core/AlignedLayerServiceManager.sol#L190

```
...
contract AlignedLayerServiceManager is
    ...
{
    ...

    function balanceOf(address account) public view returns (uint256) {
        return batchersBalances[account];
    }

    receive() external payable {
        batchersBalances[msg.sender] += msg.value;
    }
    ...
}
```

Recommendations

- We recommend adding a withdrawal function to the AlignedLayerServiceManager contract to allow users to withdraw their balances.

XV. Metrics package is vulnerable to Slowloris attack

Rating	Low
ID	**FL-AL-15**
Category	**Denial Of Service**
Target	**Aggregator, Operator**
Commit	325aef8

Description

We found that the metrics package is vulnerable to a Slowloris attack.
This denial-of-service attack affects web servers by keeping many connections to the target web server open and holding them open as long as possible.
This consumes both the maximum number of connections the server can accept.

Recommendations

- Initialize the metrics server with http.Server struct and set *Timeouts and other fields like MaxHeaderBytes to prevent DoS attacks.

XVI. BatcherPaymentService ecrecover missing checks

Rating	Informational
ID	FL-AL-16
Category	Data Validation
Target	BatcherPaymentService
Commit	325aef8

Description

The BatcherPaymentService uses ecrecover without any checks on the result.
This isn't exploitable in the current codebase but future changes could introduce a vulnerability if the returned address is not checked.

contracts/src/core/BatcherPaymentService.sol#L239

```
...
contract BatcherPaymentService is
    ...
{
    ...
    function verifySignatureAndDecreaseBalance(
        bytes32 hash,
        SignatureData calldata signatureData,
        uint256 feePerProof
    ) private {
        ...
        address signer = ecrecover(
            noncedHash,
            signatureData.v,
            signatureData.r,
            signatureData.s
        );
        ...
    }
    ...
}
```

Recommendations

- Validate that the returned address of calling ecrecover isn't address zero

XVII. BatcherPaymentService switch to Ownable2StepUpgradeable

Rating	Informational
ID	FL-AL-17
Category	Access Control
Target	BatcherPaymentService
Commit	325aef8

Description

The BatcherPaymentService uses OwnableUpgradeable which can be replaced with Ownable2StepUpgradeable.
OwnableUpgradeable has a shortcoming that allows the owner to transfer ownership to a non-existent or mistyped address. This can lead to a loss of control over the contract.
Ownable2StepUpgradeable is a more secure alternative.

contracts/src/core/BatcherPaymentService.sol#L10

```
...
contract BatcherPaymentService is
    ...
    OwnableUpgradeable,
    ...
{
    ...
}
```

Recommendations

We recommend using Ownable2StepUpgradeable instead of OwnableUpgradeable to prevent the owner from transferring ownership to a non-existent or mistyped address.

XVIII. BatcherPaymentService incomplete initialization

Rating	Informational
ID	FL-AL-18
Category	Initialization
Target	BatcherPaymentService
Commit	325aef8

Description

BatcherPaymentService does not call __Pausable_init() during the initialization of the contract.
Even though the __Pausable_init() hook is currently empty, it is recommended to call it nonetheless. This would help reduce the error-proneness of the initialization of future implementations that implement some logic in this hook.

contracts/src/core/BatcherPaymentService.sol#L49

```
...
contract BatcherPaymentService is
    ...
{
    ...
    function initialize(
        address _AlignedLayerServiceManager,
        address _BatcherPaymentServiceOwner,
        address _BatcherWallet
    ) public initializer {
        __Ownable_init(); // default is msg.sender
        __UUPSUpgradeable_init();
        _transferOwnership(_BatcherPaymentServiceOwner);

        AlignedLayerServiceManager = _AlignedLayerServiceManager;
        BatcherWallet = _BatcherWallet;
    }
    ...
}
```

Recommendations

- We recommend calling __Pausable_init() during the initialization of the BatcherPaymentService contract to ensure that the contract is correctly initialized.

XIX. BatcherPaymentService missing address(0) check in initializer

Rating	Informational
ID	**FL-AL-19**
Category	**Data Validation**
Target	**BatcherPaymentService**
Commit	325aef8

Description

The BatcherPaymentService contract does not check that an address is not equal to address(0) during the initialization process.

contracts/src/core/BatcherPaymentService.sol#L49

```
...
contract BatcherPaymentService is
    ...
{
    ...
    function initialize(
        address _AlignedLayerServiceManager,
        address _BatcherPaymentServiceOwner,
        address _BatcherWallet
    ) public initializer {
        __Ownable_init(); // default is msg.sender
        __UUPSUpgradeable_init();
        _transferOwnership(_BatcherPaymentServiceOwner);

        AlignedLayerServiceManager = _AlignedLayerServiceManager;
        BatcherWallet = _BatcherWallet;
    }
    ...
}
```

Recommendations

- We recommend adding a check to ensure that the addresses are not equal to address(0) during the initialization process.

XX. AlignedLayerServiceManager initialization can be frontrun

Rating	Informational
ID	FL-AL-20
Category	Access Control
Target	AlignedLayerServiceManager
Commit	325aef8

Description

The AlignedLayerServiceManager contract does not implement a way to prevent frontrunning during the initialization step.

However since the deployer script uses a ProxyAdmin with an upgradeAndCall to initialize the contract, the frontrunning is impossible in this case.

contracts/src/core/AlignedLayerServiceManager.sol#L52

```
...
contract AlignedLayerServiceManager is
    ...
{
    ...
    function initialize(address _initialOwner) public initializer {
        _transferOwnership(_initialOwner);
    }
    ...
}
```

Recommendations

- We recommend implementing access control checks in the AlignedLayerServiceManager contract to prevent frontrunning during the initialization step in case the contract is behind another type of proxy.

XXI. AlignedLayerServiceManager incomplete initialization

Rating	Informational
ID	FL-AL-21
Category	Initialization
Target	AlignedLayerServiceManager
Commit	325aef8

Description

The AlignedLayerServiceManager contract does not call the __ServiceManagerBase_init() hook in the initialize function.

Even though the __ServiceManagerBase_init() hook also initializes the contract owner.
The AlignedLayerServiceManager contract does not call initialize the rewardInitiator like in the ServiceManagerBase contract.

contracts/src/core/AlignedLayerServiceManager.sol#L52

```
...
contract AlignedLayerServiceManager is
    ServiceManagerBase,
    ...
{
    ...
    function initialize(address _initialOwner) public initializer {
        _transferOwnership(_initialOwner);
    }
    ...
}
...
```

eigenlayer-middleware/blob/dev/src/ServiceManagerBase.sol#L62

```
contract ServiceManagerBase is
    ...
{
    ...
    function __ServiceManagerBase_init(
        address initialOwner,
        address _rewardsInitiator
    ) internal virtual onlyInitializing {
        _transferOwnership(initialOwner);
        _setRewardsInitiator(_rewardsInitiator);
    }
    ...
}
```

Recommendations

- We recommend calling __ServiceManagerBase_init() during the initialization of the BatcherPaymentService contract to ensure that the contract is correctly initialized.

XXII. AlignedLayerServiceManager missing address(0) check in constructor and initializer

Rating	Informational
ID	FL-AL-22
Category	Data Validation
Target	AlignedLayerServiceManager
Commit	325aef8

Description

The AlignedLayerServiceManager contract does not check that an address is not equal to address(0) in the constructor or during the initialization process.

contracts/src/core/AlignedLayerServiceManager.sol#L35

```
...
contract AlignedLayerServiceManager is
    ...
{
    ...
    constructor(
        IAVSDirectory __avsDirectory,
        IRewardsCoordinator __rewardsCoordinator,
        IRegistryCoordinator __registryCoordinator,
        IStakeRegistry __stakeRegistry
    )
        BLSSignatureChecker(__registryCoordinator)
        ServiceManagerBase(
            __avsDirectory,
            __rewardsCoordinator,
            __registryCoordinator,
            __stakeRegistry
        )
    {
        _disableInitializers();
    }
    ...
}
```

contracts/src/core/AlignedLayerServiceManager.sol#L52

```
...
contract AlignedLayerServiceManager is
    ...
{
    ...
    function initialize(address _initialOwner) public initializer {
        _transferOwnership(_initialOwner);
    }
    ...
}
```

Recommendations

- We recommend adding a check to ensure that the addresses are not equal to address(0) during the deployment and the initialization process.

XXIII. Dockerfiles need to be hardened

Rating	Informational
ID	**FL-AL-23**
Category	**Best Practices**
Target	**Dockerfiles**
Commit	325aef8

Description

Multiple best practices can be added to the Dockerfiles:

- The FROM directive is used without specifying a hash. The base image is pulled from the Docker Hub registry. If the base image is compromised, the entire image is compromised.
- The project images update the package manager cache without installing any packages in the same layer and delete the apt-get lists afterward.
- The project images don't have a HEALTHCHECK directive.
- The Dockerfiles don't use the USER directive to specify the user that the image should run as. By default, Docker runs the image as the root user. This could help to make a privilege escalation.
- Use multi-stage builds for images.
- Limit syscalls for a container.
- Limit CPU usage.
- Limit memory usage.
- Limit swap memory usage.
- Limit disk I/O.
- Add the --security-opt no-new-privileges directive when launching the container.
- Prevent PID namespace sharing.
- Use user namespaces to isolate container users from host users.
- Disable inter-container communication.

Recommendations

- Using a specific version of the base image and verifying the integrity of the base image before building the image.
  explorer/Dockerfile

```
# https://hub.docker.com/_/postgres
FROM
postgres:16.3@sha256:d0f363f8366fbc3f52d172c6e76bc27151c3d643b870e1062b4e8bfe65baf609
...
```

- Updating the packages and installing the required packages in the same layer to reduce the image size.
  operator/docker/operator.Dockerfile

```
FROM golang:1.22.4

# Get Ubuntu packages
RUN apt-get update && apt-get install --no-install-recommends -y \
    build-essential \
    curl \
    openssl \
    libssl-dev && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

...
```

- Adding the HEALTHCHECK directive in the Dockerfiles to ensure that the containers are healthy and ready to serve traffic.
  explorer/Dockerfile

```
# https://hub.docker.com/_/postgres
FROM postgres:16.3

# Environment variables
ENV POSTGRES_USER=explorer_user
ENV POSTGRES_PASSWORD=explorer_pass
ENV POSTGRES_DB=explorer_db

# Expose the default PostgreSQL port
EXPOSE 5432

HEALTHCHECK --interval=5s --timeout=3s \
  CMD pg_isready -U $POSTGRES_USER -d $POSTGRES_DB || exit 1
```

- Using the USER directive to specify the user that the image should run as.
  explorer/Dockerfile

```
# https://hub.docker.com/_/postgres
FROM postgres:16.3

# Environment variables
ENV POSTGRES_USER=explorer_user
ENV POSTGRES_PASSWORD=explorer_pass
ENV POSTGRES_DB=explorer_db

# Expose the default PostgreSQL port
EXPOSE 5432

# ? The postgres image runs as the postgres user by default
https://hub.docker.com/layers/library/postgres/16.3/images/sha256-10a028bdde46f9a
c6786a8609b16672a7d0d141d6cbe776f7fbad5e82f4fca3a?context=explore
USER postgres
```

XXIV. Inconsistent Serialization in the Marshal & Unmarshal Functions

Rating	Informational
ID	**FL-AL-24**
Category	**Data Validation**
Target	**Operator**
Commit	81e1ae4

Description

During an audit and fuzzing of the Marshal and Unmarshal functions, we identified an inconsistency in the serialization and deserialization process. The function does not correctly handle certain inputs, leading to discrepancies between the original and marshaled data.

Recommendations

- Implement strict input validation to ensure that all input fields are as expected before processing.
- Improve error handling to provide more informative error messages when the input data does not match the expected structure.

XXV. Verify groth16 proof allows malleability

Rating	Informational
ID	**FL-AL-25**
Category	**Best Practices**
Target	**Operator**
Commit	81e1ae4

Description

The groth16 verification function allows malleability. This is an expected behavior of the groth16 proof system.
However, an Aligned Layer user could be exposed to potential security risks if the malleability is a risk for his usage

Recommendations

- Inform the user that the groth16 proof system allows malleability in case it could affect their usage.

XXVI. Missing content security policy in router.ex

Rating	Informational
ID	**FL-AL-26**
Category	**Best Practices**
Target	**Explorer**
Commit	81e1ae4

Description

There is a missing Content-Security-Policy header in the router.ex file.

Recommendations

You can specify CSP for put_secure_browser_headers in the browser pipeline of ExplorerWeb.Router

XXVII. Aggregator has a data race during BLS aggregation

Rating	Informational
ID	FL-AL-27
Category	Dependencies
Target	Aggregator
Commit	81e1ae4

Description

The bls aggregation library from eigen-sdk contains a data race.
It was already found that the eigen-sdk bls aggregation service contains a data race and the issue is described here.

Recommendations

Update the eigen-sdk library to v0.1.8 or later to fix the data race issue.

XXVIII. Unmaintained and yanked crates

Rating	Informational
ID	FL-AL-28
Category	Dependencies
Target	Batcher, Operator
Commit	325aef8

Description

We used the cargo deny to identify the outdated crates with the following command:

```
cargo deny check advisories
```

Recommendations

- For ansi_term crate, we recommend contacting the maintainers of the sp1 crates to discuss the possibility of migrating to one of the suggested alternatives.
- For the dotenv crate, we recommend considering the dotenvy crate as an alternative.
- For the proc-macro-error crate, we recommend contacting the maintainers of the alloy-* crates to discuss the possibility of migrating to one of the suggested alternatives.
- Update openssl since it has a fixed vulnerability
- For the yanked bytemuck and bytes crates, we recommend updating the dependencies to a non-yanked version.

Conclusion

Our audit of Aligned Layer highlighted this innovative protocol's potential and challenges. While we uncovered several vulnerabilities, the swift and attentive response from the Aligned Layer team underscores their commitment to security and transparency. Their ability to address issues rapidly, combined with the protocol's impressive throughput and integration with Ethereum's proof-of-stake system, positions Aligned Layer as a leading solution in the zk-proof verification space.

Disclaimer

This report is provided under the terms of the agreement between FuzzingLabs and the client. It is intended solely for the client's use and may not be shared or referenced without FuzzingLabs' prior written consent.
The findings in this report are based on a point-in-time assessment and do not guarantee the absence of vulnerabilities or security flaws. FuzzingLabs cannot ensure future security as systems and threats evolve. We recommend continuous monitoring, independent assessments, and a bug bounty program.
This report is not financial, legal, or investment advice, and should not be used for decision-making regarding project involvement or investment. FuzzingLabs aims to help reduce risks, but we do not provide any guarantees regarding the complete security of the technology assessed.