MixBytes()

DIA Spectra Security Audit Report

SEPTEMBER 15, 2025

Table of Contents

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

The DIA Oracle protocol is a cross-chain price feed and data delivery system leveraging Hyperlane's interoperability infrastructure. The audit scope consists of three main components: `OracleIntentRegistry` for registering and managing oracle intents with EIP-712 signatures, `OracleTriggerV2` for dispatching oracle data across chains via Hyperlane bridge, and `PushOracleReceiverV2` for receiving and processing oracle updates on destination chains.

The audit was conducted over 3 days by 3 auditors, involving an in-depth manual code review and automated analysis within the scope.

Throughout the audit process, beyond examining standard attack vectors and items from our internal checklist, we conducted a comprehensive analysis of the following areas:

- **Update Blocking and Reordering.** We verified that potential reordering of ISM and intent updates or selective omission by independent actors does not materially distort system behavior.
- **Cross-Chain Message Authorization.** The project uses the Hyperlane bridge to transfer cross-chain messages. We verified that message authorization is sufficiently enforced so that a malicious actor cannot submit a message that would be accepted as valid and either corrupt oracle data or disrupt or halt the system.
- **Cross-Function Intent Registration Order.** We confirmed that the order of calling `registerIntent()` and `registerMultipleIntents()` functions cannot lead to state inconsistency or race conditions because both functions properly validate intents and update the same storage mappings.
- **Signature Malleability Exploitation.** We validated that signature malleability in the `recoverSigner()` function cannot be exploited for replay attacks. We confirmed that intents are properly deduplicated through the `processedIntents` mapping.
- **Domain Separator Synchronization Between Contracts.** We verified that `OracleIntentRegistry` and `PushOracleReceiverV2` contracts should share the same domain separator, which is important for the signature verification process to work correctly across the system, as mismatched domain separators would cause signature validation failures and break the oracle functionality.

- **Intent Hash Deduplication Mechanism.** We confirmed that once an intent hash is populated to the `processedIntents` mapping, it cannot be registered again. The system checks `processedIntents[intentHash]` before processing any new intent, ensuring that each unique intent can only be processed once across the entire system.

The centralized signer model in this protocol provides operational efficiency and streamlined oracle management, which is appropriate for the system's design. However, this approach also creates significant security dependencies, as authorized signers wield substantial power over price data and can submit any price for any asset. In the event of a private key compromise, a malicious actor could submit arbitrary price data for any asset, potentially causing severe economic damage to dependent protocols. While administrators can deauthorize malicious signers, a single malicious or mistaken action can permanently brick the registry contract as described in the M-1 finding.

Detailed findings discovered during the audit are presented in the **Findings Report** section.

1.3 Project Overview

Summary

Title	Description
Client Name	DIA
Project Name	Spectra
Type	Solidity
Platform	EVM
Timeline	01.09.2025 - 10.09.2025

Scope of Audit

File	Link
contracts/contracts/OracleIntentRegistry.sol	OracleIntentRegistry.sol
contracts/contracts/OracleTriggerV2.sol	OracleTriggerV2.sol
contracts/contracts/PushOracleReceiverV2.sol	PushOracleReceiverV2.sol

File	Link
contracts/contracts/libs/OracleIntentUtils.sol	OracleIntentUtils.sol

Versions Log

Date	Commit Hash	Note
01.09.2025	904f540ce2a2503dadcb8eb8c83745682a17edd9	Initial Commit
10.09.2025	0bd0bafd8986ee84c24a091e6de53119fe6fb35c	Commit for the re-audit

Mainnet Deployments

File	Address	Blockchain
OracleIntentRegistry.sol	0xd046fd65B0A59CaB39bB52E94a8555a0c2b9299d	DIA
OracleTriggerV2.sol	0xA5fE9de052CfB6bCA6CaC66b3Ae7Fb9F12607B9F	DIA
ProtocolFeeHook.sol	0x467b3CeCAE5F812863d72C2551Ec28f9E1F7834b	Arbitrum
PushOracleReceiverV2.sol	0xc342DF78FE48C15Af86ddFA99dF65251DF6c177B	Arbitrum

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	0
Medium	1
Low	14

Findings Statuses

ID	Finding	Severity	Status
M-1	Missing Timestamp Validation Allows DoS Attack On Intent Creation	Medium	Fixed
L-1	Inconsistent Expiry Validation Between Intent Registration Functions	Low	Fixed
L-2	`OracleTriggerV2.setDomainSeparator()` Is Never Used	Low	Fixed
L-3	Signature Malleability Vulnerability in `recoverSigner()`	Low	Fixed
L-4	Unused Code Documentation For Non-Existent Function	Low	Fixed
L-5	Missing Expiry Validation In Intent Processing	Low	Acknowledged
L-6	Deauthorized Signers Can Still Provide Oracle Data Through Latest Intent Retrieval	Low	Fixed
L-7	Hardcoded `block.chainId` Is Used in `domainSeparator`	Low	Fixed
L-8	Code Duplication in `registerIntent()` and `registerMultipleIntents()`	Low	Acknowledged
L-9	Zero Address Signer Risk	Low	Fixed

L-10	Weak intent format discrimination can misclassify payloads	Low	Fixed
L-11	Typos	Low	Fixed
L-12	Rejected or Ignored Updates Emit No Events	Low	Fixed
L-13	`retrieveLostTokens()` Functions Lack Amount Parameter For Selective Withdrawal	Low	Fixed
L-14	Protocol Fee Can Be Bypassed	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Missing Timestamp Validation Allows DoS Attack On Intent Creation		
Severity	Medium	**Status**	Fixed in 0bd0bafd

Description

`OracleIntentRegistry.registerIntent()` and `OracleIntentRegistry.registerMultipleIntents()` do not validate the `timestamp` field of incoming oracle intents. The contract accepts any timestamp value without checking if it is reasonable or within acceptable bounds. This creates an issue where an authorized signer can submit an intent with an extremely large future timestamp.

When an intent with a very large timestamp is registered, it becomes the latest intent for that symbol and type combination in the `latestIntentByTypeAndSymbol` mapping. Since the contract only updates this mapping when the new timestamp is greater than the existing one, all subsequent legitimate intents with normal timestamps will be rejected as **stale** data. This effectively creates a denial-of-service condition for that specific oracle symbol, preventing any future price updates until the contract owner intervenes.

The vulnerability is classified as **Medium** severity because it requires only a single malicious or compromised authorized signer to permanently freeze an oracle feed.

Recommendation

We recommend checking that `intent.timestamp` is not greater than `block.timestamp` during intent registration.

Client's Commentary

The changes have been implemented as recommended. PR-11

2.4 Low

L-1	Inconsistent Expiry Validation Between Intent Registration Functions		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

The registerIntent() and registerMultipleIntents() functions in OracleIntentRegistry contract implement inconsistent validation logic for the expiry field. The registerMultipleIntents() function properly validates that intents are not expired and skips expired intents:

```
if (block.timestamp > data.expiry) {
    continue;
}
```

The registerIntent() function completely ignores the expiry field and accepts expired intents without any validation.

Recommendation

We recommend adding expiry validation to the registerIntent() function to match the validation logic in registerMultipleIntents().

Client's Commentary

The changes have been implemented as recommended. PR-11

L-2	OracleTriggerV2.setDomainSeparator() Is Never Used		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

The OracleTriggerV2.setDomainSeparator() function is implemented but never actually used, creating confusion. The function's documentation states that the domain separator must match exactly with PushOracleReceiverV2's domain separator for signature validation to work correctly across the system. However, the implementation uses address(this) as the verifying contract, while PushOracleReceiverV2 expects the verifying contract to be the OracleIntentRegistry contract address.

Recommendation

We recommend removing the unused setDomainSeparator() function from OracleTriggerV2 contract to eliminate confusion and reduce code complexity.

Client's Commentary

The changes have been implemented as recommended. PR-11

L-3	Signature Malleability Vulnerability in recoverSigner()		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

The OracleIntentUtils.recoverSigner() function does not validate that the s component of the ECDSA signature is in the lower half of the secp256k1 curve order. This allows signature malleability where multiple valid signatures can be created for the same message and signer. An attacker can take any valid signature and create a malleable version that recovers to the same address but represents a different byte sequence.
While this vulnerability does not directly enable replay attacks in the current system, it creates potential security risks in the future.

Recommendation

We recommend adding validation to ensure the s component is in the lower half of the secp256k1 curve order.
A secure approach can be found in the ECDSA.sol#L175-L186.

Client's Commentary

The changes have been implemented as recommended. PR-11

L-4	Unused Code Documentation For Non-Existent Function		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

The PushOracleReceiverV2 contract contains documentation for a function that does not exist.

```
/**
 * @notice Performs the actual data update for an intent (unified wrapper)
 * @param intent The OracleIntent containing the data
 * @param intentHash The hash of the intent for events
 * @return updated Whether the data was actually updated
 */
```

Recommendation

We recommend removing this documentation to clean up the codebase and prevent confusion.

Client's Commentary

The changes have been implemented as recommended. PR-11

L-5	Missing Expiry Validation In Intent Processing		
Severity	Low	**Status**	Acknowledged

Description

The `PushOracleReceiverV2._validateIntentStatus()` and `_validateIntentCommonFromMemory()` functions fail to validate the `expiry` field of incoming oracle intents.

This allows expired oracle intents to be processed and accepted by the contract, potentially leading to stale price data being stored and served to downstream consumers.

The vulnerability affects both direct intent processing through `handleIntentUpdate()` and `handleBatchIntentUpdates()`, as well as cross-chain intent processing through `handle()`. Since `_validateIntentCommonFromMemory()` is used in the cross-chain flow, expired intents received via the mailbox can also update oracle prices, bypassing the intended expiry protection mechanism.

The contract includes an `Expired` status in the `ValidationStatus` enum and has error handling for `IntentExpired()` in the `_processIntent()` function, indicating that expiry validation was intended to be implemented but was never actually added to the validation logic.

Recommendation

We recommend adding expiry validation to both `_validateIntentStatus()` and `_validateIntentCommonFromMemory()`.

Client's Commentary

Expiry logic is not required here, so the 'Expired' status has been removed.

L-6	Deauthorized Signers Can Still Provide Oracle Data Through Latest Intent Retrieval		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

The `OracleIntentRegistry.getLatestIntentHashByType()` function returns intent hashes without validating whether the associated signer is still authorized. When a signer is deauthorized, their previously registered intents remain accessible through this function, allowing downstream systems to continue using data from deauthorized signers.

This creates an issue where deauthorized signers can still influence oracle data through their previously registered intents. The `dispatchToChain()` and `dispatch()` functions in `OracleTriggerV2` rely on `getLatestIntentHashByType()` to retrieve the latest intent and forward it to destination chains. If the latest intent belongs to a deauthorized signer, stale or potentially malicious data from that signer will be propagated across the oracle network.

Recommendation

We recommend adding signer authorization validation to the `getLatestIntentHashByType()` function to ensure that only intents from currently authorized signers are returned.

Client's Commentary

The changes have been implemented as recommended.

L-7	Hardcoded block.chainId Is Used in domainSeparator		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

In OracleIntentRegistry contract, the chainId is embedded at deployment time and effectively becomes part of the bytecode. If the network is later forked, this can enable replay attacks between the original and the forked networks.

Recommendation

We recommend using the runtime block.chainId instead of hardcoding it at deploy time. A secure approach can be found in the EIP712.sol#L82-L88.

Client's Commentary

The changes have been implemented as recommended. PR-11

L-8	Code Duplication in registerIntent() and registerMultipleIntents()		
Severity	Low	**Status**	Acknowledged

Description

OracleIntentRegistry.registerIntent() and OracleIntentRegistry.registerMultipleIntents() implement nearly the same functionality, but the logic is duplicated and diverges in places. This leads to inconsistent behavior (e.g., expiry handling) and inconsistent validation error handling (reverts vs. silent ignores).

Recommendation

We recommend consolidating the logic into a single code path for updating one intent and implementing multiple-intent updates as repeated invocations of that single-intent path.

Client's Commentary

The functions operate on different data types; one uses memory while the other uses storage. Deduplication has been performed to the fullest extent possible.

L-9	Zero Address Signer Risk		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

`ecrecover` invoked with an invalid signature may return the zero address. While the contract currently verifies that a signer is included in `authorizedSigners` mapping, the zero address could be added by an administrative mistake.

Recommendation

We recommend validating non-zero addresses in `OracleIntentRegistry.setSignerAuthorization()` and `OracleIntentUtils.validateSignature()` functions and all other code paths relying on recovered addresses. Ensure that `ecrecover` does not yield the zero address before proceeding and treat `address(0)` as invalid even if present in `authorizedSigners` mapping.

Client's Commentary

The changes have been implemented as recommended. PR-11

L-10	Weak intent format discrimination can misclassify payloads		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

OracleIntentUtils.isIntentFormat() returns true if _data.length >= 200. This heuristic can misclassify non-intent payloads as intent, causing abi.decode in _handleIntentMessage() to revert and drop legitimate mailbox messages. For example, a valid ISM message (intended for the _handleISMValidatedMessage path) with a long key string can easily exceed 200 bytes, causing it to be mistakenly treated as an intent message and passed to _handleIntentMessage(), which will then abi.decode it and revert.

Moreover, real intent payloads are significantly larger than 200 bytes:

- ~192 bytes for fixed-size fields
- ~160 bytes for dynamic field offsets
- At least ~160 bytes more for actual dynamic data

A realistic lower bound is approximately 736 bytes, and even minimum size (with empty strings) is 512 bytes.

Recommendation

We recommend using a robust discriminator:

- Prefix intent messages with fixed "magic bytes" (e.g., bytes4(keccak256("DIAI"))) or a versioned envelope struct that starts with a constant.
- Check that prefix before deciding the decoding path.
- Optionally, retain a conservative size check (e.g., >= 512 bytes) as an additional sanity guard, not as the primary discriminator.

Client's Commentary

The payload length was increased, and the intent prefix was omitted to prevent complications for explorers

L-11	Typos		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

Several typos were identified.

In audit scope:

- contracts/contracts/OracleTriggerV2.sol: L33: "OracleRequestReceipent" → "OracleRequestRecipient".

Out of scope files:

- contracts/contracts/OracleTrigger.sol: L35: "OracleRequestReceipent" → "OracleRequestRecipient".
- contracts/contracts/RequestOracle.sol: L263: "reover" → "recover" in "Withdraw ETH to reover stuck funds".
- contracts/contracts/OracleRequestRecipient.sol: L198: "reover" → "recover" in "Withdraw ETH to reover stuck funds".
- contracts/contracts/example/examplerequestbasedoracle.sol: L39, L48: "reciever" → "receiver".

Recommendation

We recommend correcting the typos.

Client's Commentary

Client: The changes have been implemented as recommended. PR-11

MixBytes: All typos have been fixed, except in `OracleTrigger.sol` at line 35.

L-12	Rejected or Ignored Updates Emit No Events		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

Several code paths drop updates silently:

- `OracleIntentRegistry.registerMultipleIntents()` continues on invalid items without emitting a reasoned event.
- `PushOracleReceiverV2._handleISMValidatedMessage()` returns on stale data without emitting.
- `PushOracleReceiverV2.handleBatchIntentUpdates()` skips invalid intents with no event.
- `PushOracleReceiverV2._processIntent()` with `revertOnFailure == false` returns `false` on failed validation with no event.

This reduces transparency and complicates monitoring.

Recommendation

We recommend emitting reasoned events for all ignore paths.

Client's Commentary

The changes have been implemented as recommended.

L-13	retrieveLostTokens() Functions Lack Amount Parameter For Selective Withdrawal		
Severity	Low	Status	Fixed in 0bd0bafd

Description

Both OracleTriggerV2.retrieveLostTokens() and PushOracleReceiverV2.retrieveLostTokens() withdraw the entire ETH balance to receiver. This makes partial recoveries impossible and increases blast radius of operator error.

Recommendation

We recommend adding an amount parameter and transfer exactly that amount, with a balance check.

Client's Commentary

The changes have been implemented as recommended. PR-11

L-14	Protocol Fee Can Be Bypassed		
Severity	Low	**Status**	Fixed in 0bd0bafd

Description

In `PushOracleReceiverV2._transferProtocolFee()`, fee is `gasUsed * tx.gasprice`, then capped to contract balance. If `tx.gasprice == 0` or balance is 0, the fee is effectively bypassed. If the economic model expects mandatory fee payment, this is a gap.

Recommendation

We recommend enforcing a minimum fee floor independent of `tx.gasprice`, or requiring pre-funding:

- Set `uint256 minFee = ProtocolFeeHook(paymentHook).minFeeWei();`
- Compute `fee = max(gasUsed * tx.gasprice, minFee)` and revert if `fee > balance`.
- Alternatively, require caller to prepay fee via `msg.value` and forward that amount.

Client's Commentary

A mandatory fee has been added on top of the gas fee.

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

https://mixbytes.io/

https://github.com/mixbytes/audits_public

hello@mixbytes.io

https://x.com/mixbytes