Kann Audits

Encifher Security Review

12/02/2026

Contents

1 About Kann Audits

Kann Audits is a top-tier Web3 security audit company, trusted by leading projects and providing comprehensive audits and expert guidance to secure the most critical blockchain protocols.

Check out our previous work or reach out on Twitter @KannAudits.

2 Disclaimer

A security audit can never guarantee the complete absence of vulnerabilities. Audits are a time, resource, and expertise-bound effort in which we aim to identify as many issues as possible.

Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users
- **Medium** - leads to a moderate material loss of assets in the protocol or moderately harms a group of users
- **Low** - leads to a minor material loss of assets in the protocol or harms a small group of users

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost

- **Medium** - only a conditionally incentivized attack vector, but still relatively likely
- **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive

3 About Encifher

Encifher is a Web3 privacy protocol focused on bringing confidential and compliant transactions to decentralized finance (DeFi) on the Solana blockchain. Instead of exposing sensitive details like transaction amounts or trading strategies publicly on-chain, Encifher uses advanced cryptography and off-chain secure computation to encrypt value while keeping key parts of DeFi composability intact.

4 Executive Summary

Protocol Summary

Project Name	Encifher
Protocol Type	Privacy bridging
Timeline	04/02/2026 - 11/02/2026
Review commit hash:	
`Private`	
Fixes review commit hash:	
`Private`	

Scope

`packages/consumer/*`	`packages/listener/*`
`packages/mixer/*`	`packages/pool-manager/*`
`packages/refund/*`	`packages/server/*`
`packages/solver/*`	

5 Findings

Findings count

Severity	**Amount**
Critical	0
High	0
Medium	2
Low	0
Informational	2
Total findings	**4**

Summary of findings

ID	**Title**	**Severity**
[M-01]	Slippage Validation Failures	Medium
[M-02]	Kafka Consumer Retry Mechanism	Medium
[I-01]	Kafka missing SSL/TLS encryption and no auth	Info
[I-02]	Incomplete Balance Check in SolanaSolver	Info

5.1 Medium Severity

5.1.1 [M-01] Slippage Validation Failures

Description: Slippage values are calculated and displayed to users but never enforced during swap execution, leading to potential silent losses and misleading quotes.

Code Example:

```
// Quote calculation
slippageLegOne: jupQuote.slippageBps,
slippgageLegTwo: 50,

// Swap execution - missing slippageBps
orderUrl.searchParams.set("inputMint", inputMint);
orderUrl.searchParams.set("outputMint", outputMint);
orderUrl.searchParams.set("amount", amount);
```

Impact: - Users may receive significantly less than quoted amounts - Silent failures with no warning to users - Misleading quotes could harm user trust

Recommendation: - Pass ‘slippageBps‘ to Jupiter API:

```
orderUrl.searchParams.set("slippageBps", slippageLegOne.toString());
```

- Implement post-execution validation:

```
const actualOutput = await getTransactionOutput(signature);
if (BigInt(actualOutput) < BigInt(orderData.otherAmountThreshold)) {
    throw new Error(`Slippage exceeded: expected , got `);
}
```

Resolution: Fixed

5.1.2 [M-02] Kafka Consumer Retry Mechanism

Description: Single shared Kafka consumer instance (‘kafkaConsumer‘) across three consumers prevents proper error recovery, causing retries to fail and manual restarts to be required.

Code Example:

```
// Shared consumer
export const kafkaConsumer = kafka.consumer({
  groupId: "consumer-group",
  sessionTimeout: 120 * 1000,
  heartbeatInterval: 10000,
});
```

```
// All three consumers call kafkaConsumer.run() independently
```

Impact: - Failed consumers cannot recover automatically - Retry mechanism fails repeatedly every 500ms - Reduced system availability and reliability

Recommendation: - Option 1 (recommended): Create separate consumer instances:

```
export const preDepositConsumer = kafka.consumer({ groupId: "predeposit-consumer-
    group" });
export const postDepositConsumer = kafka.consumer({ groupId: "postdeposit-consumer-
    group" });
export const solverConsumer = kafka.consumer({ groupId: "solver-consumer-group" });
```

- Option 2: Use ‘Promise.allSettled()‘ instead of ‘Promise.all()‘ to handle failures without retrying entire group.

Resolution: Fixed

5.2 Informational findings

5.2.1 [I-01] Kafka missing SSL/TLS encryption and no auth

Description: All Kafka connections operate without SSL/TLS encryption and lack authentication mechanisms.

Code Example:

```
// Kafka server and consumer configuration
// No SSL/TLS or SASL auth enabled
```

Impact: - All messages transmitted in plaintext - Wallet addresses, transaction IDs, and order details exposed to network eavesdropping

Recommendation: - Enable SSL/TLS encryption - Implement SASL authentication for all Kafka connections

Resolution: Fixed

5.2.2 [I-02] Incomplete Balance Check in SolanaSolver

Description: ‘isSolvable()‘ function performs an incomplete balance check and does not account for optional gas deposit transfers.

Code Example:

```
async isSolvable(params: OrderParams): Promise<boolean> {
    const balance = await this.getConnection().getBalance(new PublicKey(currentPool)
        );
    const requiredAmount = BigInt(params.amount); // Missing gas
    return BigInt(balance) >= requiredAmount;
}
```

Impact: - Orders marked as solvable may fail due to insufficient funds - User transactions fail after acceptance - Wasted RPC calls and fees

Recommendation: Include gas deposit in balance validation:

```
let requiredAmount = BigInt(params.amount);
if (params.destinationGasDepositAddress) {
    requiredAmount += BigInt(DESTINATION_GAS_AMOUNT_LAMPORTS);
}
return availableBalance >= requiredAmount;
```

Resolution: Fixed