factcheck.fun - Security Review

28.05.2025

Conducted by:

Kann, Lead Security Researcher

Ivan Fitro, Lead Security Researcher

Pelz, Associate Security Researcher

Table of Contents

1 About Kann

Kann a Security Reseacher and Founder of Kann Audits.

2 About Ivan Fitro

Ivan Fitro a Security Reseacher and Founding SR in Kann Audits.

3 About Pelz

Pelz ASR in Kann Audits.

4 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

5 Risk classification

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

5.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

5.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

6 Executive summary

Overview

Project Name	factcheck.fun
Repository	N/A
Commit hash	N/A
Resolution	N/A
Documentation	N/A
Methods	Manual review

Scope

/src/FactCheckExchange.sol

Issues Found

Critical risk	0
High risk	0
Medium risk	2
Low risk	0
Informational	3

7 Findings

7.1 Medium risk

7.1.1 Signature Verification Broken Under EIP-7702 Due to Reliance on isContract Logic

Severity: *Medium risk*

Description: In the FactCheckExchange.sol contract, the settleMatchedOrders() function attempts to validate orders using signatures by distinguishing EOAs from contracts. It uses the following logic:

-If the address has code (isContract returns true), it calls isValidSignature() via ERC-1271.

-If the address has no code, it assumes it's an EOA and uses ECDSA.recover() to verify the signature.

This approach breaks under the upcoming Ethereum Pectra upgrade, which includes EIP-7702. EIP-7702 allows EOAs to temporarily attach code during a transaction, meaning any externally owned account can now appear as a contract at runtime.

As a result:

A 7702-enabled EOA with temporary code will cause the logic to treat it as a contract.

If the code does not implement ERC-1271, the isValidSignature() call will fail.

The logic does not fall back to ECDSA.recover(), causing legitimate EOA signatures to be rejected.

Resolution: Acknowledged

7.1.2 Lack of Minimum feeRate Check May Allow Orders to be fullfilled without paying any fees

Severity: *Medium risk*

Description: The settleMatchedOrders function allows the backend to settle matched orders that were created and matched off-chain. While the contract enforces an upper bound for feeRate, it does not enforce a minimum fee rate.

This omission allows orders with extremely low feeRate values (e.g., 1 wei) to be matched and settled. As a result, users can effectively bypass protocol fees altogether, undermining protocol revenue and fairness.

Since order matching is handled off-chain and executed on-chain by an address with the BACKEND_ROLE, malicious or careless matching could lead to widespread abuse of low-fee orders.

Recommendation: Introduce a minFeeRate variable, configurable by protocol administrators, and enforce it during order validation to ensure all orders pay at least a minimum protocol fee:

```
require(order.feeRate >= minFeeRate, "Fee rate below minimum allowed");
```

Resolution: Fixed

7.2 Informational

7.2.1 Missing Market ID Consistency Check in settleMatchedOrders

Severity: *Informational risk*

Description: The settleMatchedOrders function doesn't enforce that all makerOrders share the same marketId as the provided takerOrder. While it correctly verifies signature validity and array length consistency, it does not validate that each makerOrder.marketId equals takerOrder.marketId.

Recommendation:

```
for (uint i = 0; i < makerOrders.length; i++) {
    require(
        makerOrders[i].marketId == takerOrder.marketId,
        "FactCheckExchange: Market ID mismatch"
    );
}
```

Resolution: Acknowledged

7.2.2 Ensure Taker and Maker Orders Are on Opposite Sides

Severity: *Informational risk*

Description: In the settleMatchedOrders function, there is currently no check to enforce that the takerOrder and each makerOrder are on opposite sides of the trade (i.e., one is a buyer, the other is a seller). While it may be assumed that the backend prevents invalid matches, relying solely on off-chain guarantees can introduce hidden risks.

Recommendation:

```
require(takerOrder.side != makerOrder.side, "Orders must be on opposite sides");
```

Resolution: Acknowledged

7.2.3 Old Treasury Doesn't Get Revoked When Updated To New One

Severity: *Informational risk*

Description: In the updateTreasuryWallet() function, the contract sets a new treasuryWallet address and grants it the TREASURY_ROLE

However, the role from the previous treasury address is not revoked. As a result, any previously assigned treasury address retains the TREASURY_ROLE

Recommendation: Revoke the old treasuryWallet.

Resolution: Fixed