SIR-Trading Security review

Version 2.0

Table of Contents

1 About Egis Security **3**

2 About SIR-Trading **3**

3 Disclaimer **3**

4 Risk classification **4**
4.1 Impact 4
4.2 Likelihood 4
4.3 Actions required by severity level 4

5 Executive summary **5**

6 Findings **6**
6.1 Medium risk 6
6.1.1 User may receive less tokens than expected on minting and burning 6
6.2 Low risk 7
6.2.1 Possible stuck funds in case of min/max tick 7
6.2.2 MKR token cannot be used when creating a vault 7
6.3 Informational 8
6.3.1 Safe `PERMIT_TYPEHASH` as constant 8
6.3.2 CARDINALITY_DELTA should be different for different chains 8
6.3.3 CREATE3 isn't supported on ZkSync 8

1 About Egis Security

Egis Security is a team of experienced smart contract researchers, who strive to provide the best smart contract security services possible to DeFi protocols.

The team has a proven track record on public auditing platforms like Code4rena, Sherlock, and Cantina, earning top placements and rewards exceeding $250,000. They have identified over 300 high and medium-severity vulnerabilities in both public contests and private audits.

2 About SIR-Trading

SIR is a DeFi protocol designed to address the key challenges of leveraged trading, such as volatility decay and liquidation risks. It is designed for investors looking for small, long-term compounding leverage, in a maximally trustless setting. By only charging fees on token minting and burning, and addressing the issue of volatility decay internally, SIR offers a more efficient and safer way to engage in leveraged trading in the crypto space.

3 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**.

4 Risk classification

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

4.1 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.

4.2 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.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	SIR-Trading
Repository	https://github.com/SIR-trading/Core
Commit hash	2a717c6bce0add4a3a8f18d02eb23e7bb010988c
Resolution	d6a85b6b4ad21eca9f738ad45744a03e6b499617
Documentation	https://docs.sir.trading/
Methods	Manual review

Scope

src/APE.sol
src/Contributors.sol
src/Oracle.sol
src/SIR.sol
src/Staker.sol
src/SystemControl.sol
src/SystemControlAccess.sol
src/SystemState.sol
src/TEA.sol
src/Vault.sol
src/libraries/TickMathPrecision.sol

Issues Found

Critical risk	0
High risk	0
Medium risk	1
Low risk	2
Informational	3

6 Findings

6.1 Medium risk

6.1.1 User may receive less tokens than expected on minting and burning

Severity: *Medium risk*

Context: Vault.sol#L399

Description: It may be some time between user submitting his mint/burn transaction and the function being executed on-chain. This means that the underlying oracle price may change and result in minting/withdrawing be less than the user expects.

For burning, a gentleman may expect to receive X tokens agains all his lp, but when the transaction hits execution, we may have a new oracle being set with some significant price change,
which will result in user receiving less collatreal than expected.

Recommendation: Consider introducing `deadline`, or `minAmount` func params.

Resolution: Fixed by implementing a deadline.

6.2 Low risk

6.2.1 Possible stuck funds in case of min/max tick

Severity: *Low risk*

Context: Vault.sol

Description: If the tick range is hit (min/max in our case), the actual swap might not need all the tokens we are sending and it might need less in order to fulfill the swap.

For example if we want to swap 100 tokens, but the pool needs only 80, because it hit min/max tick, it'll request only 80 tokens when the callback is called, the rest of the 20 tokens it won't need and they don't need to be transferred.

This is only problematic if the user uses ETH, as the ETH is first wrapped then the Vault transfers the tokens requested to the Uniswap pool, so a situation can occur where the user is wrapping 100 ETH, but the pool requests only 80 ETH, the Vault will transfer 80 ETH as requested, but the other 20 ETH will be stuck inside the Vault, since it's the holder of the tokens in this case, not the user himself.

This was actually reported in 2022 for Uniswaps Router, Uniswap decided it was not a bug, but how the system is supposed to work.

To sum up the potential issue: The bug is that SwapRouter doesn't refund unspent ETH. If you sell ETH and set a limit price and it's reached during the swap, then the input amount will be reduced, but you have already sent a bigger amount. The remaining amount will be left in the router.

In our case the Vault acts semantically as the SwapRouter, the user sends his ETH to the Vault and then the Vault sends it to the pool, any unspent ETH is stuck inside the Vault.

We believe this is extremely unlikely, as hitting the min/max ticks and not using the entire input amount would mean there isn't enough liquidity in the pool, which is close to impossible, we investigated if this issue ever occurred in the wild, but we found nothing except the above article.

Recommendation: Document the risk

Resolution: Acknowledged

6.2.2 MKR token cannot be used when creating a vault

Severity: *Low risk*

Context: VaultExternal.sol#L210

Description: `VaultExternal::_generateName` - Some tokens like MKR return `bytes32` instead of string for name and symbol. In this case if MKR is either one of the tokens, the function will revert, since it's expecting a string the call won't be able to decoding resulting in a revert.

Recommendation: Consider implementing a low-level call or notifying users that tokens like MKR are not supported.

Resolution: Fixed

6.3 Informational

6.3.1 Safe PERMIT_TYPEHASH as constant

Severity: *Informational*

Context: Staker.sol#L238-L242

Description: Safe `PERMIT_TYPEHASH` as a constant in `Staker`, insteaf of computing it inside `permit` each time:

```
    bytes32 private constant PERMIT_TYPEHASH =
        keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,
            uint256 deadline)");
```

6.3.2 CARDINALITY_DELTA should be different for different chains

Severity: *Informational*

Context: Oracle.sol#L52-L53

Description: If the protocol decides to deploy on chains with different block production CARDINALITY_DELTA should be modified to account for that.

Currently it's only being deployed on Mainnet and the current

```
CARDINALITY_DELTA = uint16((TWAP_DELTA - 1) / (12 seconds)) + 1;
```

is correct, but if it's deployed on Optimism it should be:

```
CARDINALITY_DELTA = uint16((TWAP_DELTA - 1) / (2 seconds)) + 1;
```

The denominator has to be modified in order to correctly account for the faster block production on Optimism.

6.3.3 CREATE3 isn't supported on ZkSync

Severity: *Informational*

Context: AddressClone.sol#L7-L9

Description: If protocol decides to be deployed on zkSync in future, Create3 contract won't work. Ref1, Ref2