SIR - MegaETH Update

Security review

Version 1.0

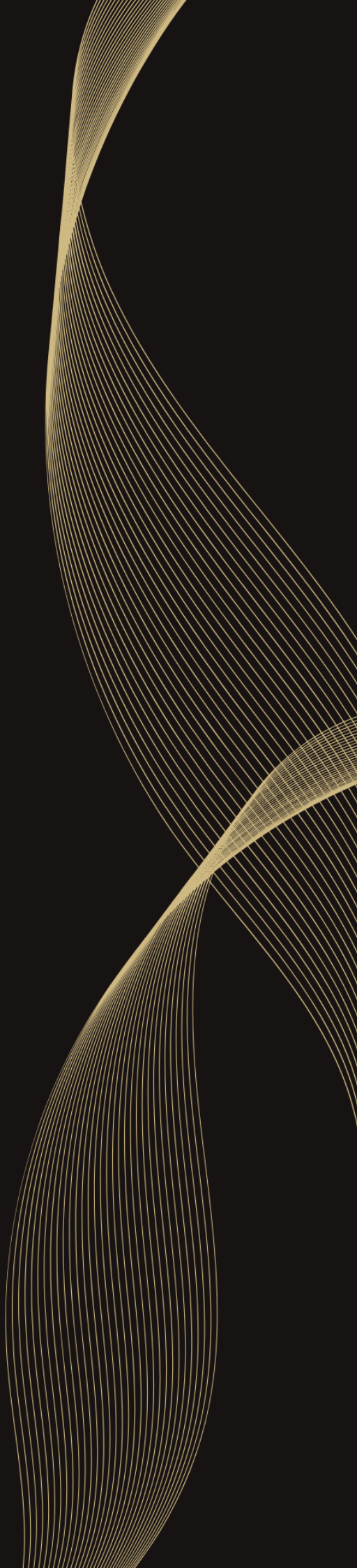

Table of Contents

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 MegaEth Update

SIR is a DeFi protocol designed to address the key challenges of leveraged trading, such as volatility decay and liquidation risks, which is already deployed on Mainnet. The current scope updates the smart contracts for a deployment on MegaETH chain. The scope of changes includes:

- Some parameters changes to config for better execution on MegaETH
- Allow LPers to choose between a fixed fee on deposit, a lock-up period during they cannot burn their TEA, or a combination of these.
- New implementation of `Contributors.sol` to scale up when a lot of addresses need an allocation

Project description to be filled manually.

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 MegaEth Update
Repository	https://github.com/SIR-trading/Core/tree/megaeth
Commit hash	884942ec55f1de873d1894c5a2bc4363d6b1a4eb
Resolution	5ffd439b2be0b662d7a3e3750aad447e32dac817
Documentation	https://docs.sir.trading/
Methods	Manual review

Scope

All changes in files under ‘src/‘ in the following PR

Issues Found

Critical risk	0
High risk	1
Medium risk	1
Low risk	1
Informational	5

6 Findings

6.1 High risk

6.1.1 User can skip lockup and fees when then have past lockup

Severity: *High risk*

Context: TEA.sol#L459

Description:

The `_updateLockEnd` function currently does not account for any existing positive `_lockEnd` timestamps in the past.

As a result, a user who previously locked an amount X of tokens could potentially mint additional tokens at a later time without paying fees or locking, because in the weighted average calculation:

$$newLockEnd = \left\lceil \frac{B_1 \cdot L_1 + A \cdot L_2}{B_1 + A} \right\rceil$$

the `B1` term corresponds to the old balance, whose associated lock (L1 = _lockEnd) may now be in the past. When L1 is earlier than the current block timestamp, the formula effectively treats it as non-locking, allowing the user to bypass fees or locking for the new deposit.

Impact:

Users can exploit past locks to minimize or completely avoid new lock-based fees, undermining the intended incentive mechanism

Recommendation: Update `_lockEnd` to `block.timestamp` before doing the weighted average, if it has expired.

Resolution: Fixed in 042693

6.2 Medium risk

6.2.1 Expired lock ordering prevents transfers between expired accounts

Severity: *Medium risk*

Context: TEA.sol#L181

Description:

The `safeTransferFrom` and `safeBatchTransferFrom` functions enforce a relative `lockEnd` ordering constraint that continues to apply even after locks have expired. As a result, a user whose lock expired later cannot transfer tokens to another user whose lock expired earlier, despite both locks being expired. This creates an unintended and permanent transfer restriction on expired funds.

```
if (_lockEnd[from][vaultId] > _lockEnd[to][vaultId])
    revert TransferToLowerLockEnd();
```

Impact:

Users with expired locks in the past cannot receive funds if they don't reset their lock by paying fees to mint, or re-locking their lp

Recommendation: If a lock has expired, update it's `lockEnd` to the current timestamp (See H-01)

Resolution: Fixed in 042693

6.3 Low risk

6.3.1 User can transfer it's locked tokens to the `TEA` contract

Severity: *Low risk*

Context: TEA.sol#L175

Description:

For TEA transfers, we enter the logic that checks if a sender lock has expired only when `(to != address(this))`:

```
if (to != address(this)) {
    // enforce lock expiration
}
```

This means a user could transfer tokens directly to the TEA contract, bypassing the lock enforcement. While there may be no immediate economic incentive to do so, it violates the intended invariant:

"Users should not be able to move tokens while _lockEnd > block.timestamp."

Impact:
Broken invariant

Recommendation: Don't allow users to transfer to TEA contract, if their lock hasn't expired

Resolution: Acknowledged
"That is because TEA tokens owned by the contract are
considered protocol owned liquidity, and the protocol is always happy to accept liquidity."

6.4 Informational risk

6.4.1 Rename of all occurrences of SIR to MegaSIR, including comments, natspec, etc.

Severity: *Informational risk*

Rename of all occurrences of SIR to MegaSIR, including comments, natspec, etc.

6.4.2 Exact 5% bid increase will be also rejected because the operator to trigger a revert is <=

Severity: *Informational risk*

Change the operator to `<`, or document explicitly that the bids should be strictly `> 5%`

6.4.3 Cache `_lockEnd[from][vaultId]` in `safeTransferFrom` to safe gas

Severity: *Informational risk*

Safe the value in `senderLockEnd` the same way, you've done it in `safeBatchTransferFrom`

6.4.4 Update the natspec inside `SystemState::systemParams()` to account for the new param `lpLockTime`

Severity: *Informational risk*

Update the natspec inside `SystemState::systemParams()` to account for the new param `lpLockTime`

6.4.5 Add `portionLockTime` to `Mint` event when minting lp

Severity: *Informational risk*

Add `portionLockTime` to `Mint` event when minting lp
