IMMUNEFI AUDIT

Immunefi / LAYERB

DATE August 19, 2025

AUDITOR Neumo, Security Researcher

REPORT BY Immunefi

ABOUT IMMUNEFI

Immunefi is the leading onchain security platform, having directly prevented hacks worth more than $25 billion USD. Immunefi security researchers have earned over $120M USD for responsibly disclosing over 4,000 web2 and web3 vulnerabilities, more than the rest of the industry combined.

Through Magnus, Immunefi delivers a comprehensive suite of best-in-class security services through a single command center to more than 300 projects — including Sky (formerly MakerDAO), Optimism, Polygon, GMX, Reserve, Chainlink, TheGraph, Gnosis Chain, Lido, LayerZero, Arbitrum, StarkNet, EigenLayer, AAVE, ZKsync, Morpho, Ethena, USDT0, Stacks, Babylon, Fuel, Sei, Scroll, XION, Wormhole, Firedancer, Jito, Pyth, Eclipse, PancakeSwap and many more.

Magnus unifies SecOps across the entire onchain lifecycle, combining Immunefi's market leading products and community of elite security researchers with a curated set of the very best security products and technologies provided by top security firms — including Runtime Verification, Dedaub, Fuzzland, Nexus Mutual, Failsafe, OtterSec and others.

Magnus is powered by Immunefi's proprietary vulnerabilities dataset — the largest and most comprehensive in web3, ensuring that security leaders and teams have the best possible tools for identifying and mitigating life threats before they cause catastrophic harm, all while reducing operational overhead and complexity.

Learn how you can benefit too at immunefi.com.

TERMINOLOGY

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

LIKELIHOOD	IMPACT		
	HIGH	MEDIUM	LOW
CRITICAL	Critical	Critical	High
HIGH	High	High	Medium
MEDIUM	Medium	Medium	Low
LOW	Low		
NONE	None		

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

EXECUTIVE SUMMARY

Over the course of 2 days in total, Layer3.xyz engaged with Immunefi to review the tge-contracts. In this period of time a total of 6 issues were identified.

SUMMARY

Name	Layer3.xyz
Repository	https://github.com/layer3xyz/tge-contracts
Audit Commit	9d3616b27735e2ae9a00a7dd0a8dba7302bc18b8
Type of Project	Infrastructure, NFT, Staking
Audit Timeline	Aug 4th - Aug 5th
Fix Period	Aug 6th - Aug 14th

ISSUES FOUND

Severity	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	4	4	0
Insights	2	1	1

CATEGORY BREAKDOWN

Bug	4
Gas Optimization	0
Informational	2

IMM-LOW-01

Rewards until `rewardPerShare` is greater than zero are not claimable #6

Id	IMM-LOW-01
Severity	LOW
Category	Bug
Status	Fixed in 6dd3940f20baf08f48dd4aec13cfaed223930740

Description

Function `_calculatedRewardPerShare` does not increment the value of reward per share when `totalWeights` is zero:

```
TypeScript
  function _calculatedRewardPerShare() internal view returns (uint256 _rewardPerShare) {
    if (totalWeights == 0) {
      return rewardPerShare;
    }

    uint256 _timeSinceLastUpdate = _lastTimeRewardApplicable() - lastUpdateTime;

    _rewardPerShare = rewardPerShare + _timeSinceLastUpdate * rewardPerSecond * _BASE /
totalWeights;
  }
```

This, added to the fact that when `_rewardPerShare` is zero, the `lastUpdateTime` is updated:

```
TypeScript
...
    if (_rewardPerShare == 0 || _rewardPerShare > rewardPerShare) {
      rewardPerShare = _rewardPerShare;
      lastUpdateTime = _lastTimeRewardApplicable();
    }
...
```

means all the seconds the contract is at the beginning in a state where `rewardPerShare` is zero do not count toward reward distribution. However, the calculation of `rewardPerSecond` assumes that rewards are being distributed for every second of `rewardsDuration`. As a result, once `periodFinish` is reached, there will still be undistributed rewards in the contract, even if all stakers have claimed their rewards.

The remaining amount becomes unclaimable and can only be recovered by the owner via `emergencyWithdraw`.

Let's put some numbers to see how this would unfold:

- After contract deployment and initialization, `periodFinish`, `rewardPerSecond`, `lastUpdateTime`, and `totalRewards` are zero.
- The owner calls `setRewardAmount` to set the variables above. To keep it simple, we assume `rewardsDuration` is 3,600 seconds and the reward passed to `setRewardAmount` is 3,600e18.
- Also, for simplicity, assume `block.timestamp = 0`.
    - Now: `periodFinish = 3_600`, `rewardPerSecond = 1e18`, `lastUpdateTime = 0`, and `totalRewards = 3_600e18`.
- Until the last minute of the period, nobody stakes. So `rewardPerShare` remains 0 during this time. Alice then stakes 10e18 with a zero lockup period. She gets a weight of 2.5e18, and total weights is also 2.5e18.
    - After the call, `lastUpdateTime = 3_540` and `rewardPerShare` is still zero. The user's `pendingRewards` is zero, and so is her `rewardPerShareSnapshot`.
- After `periodFinish`, Alice calls `getReward`. The call to `_updateReward` at the top sets:
    - `rewardPerShare = _timeSinceLastUpdate * rewardPerSecond * _BASE / totalWeights = (60 / 2.5) * 1e18 = 24e18`
    - `lastUpdateTime = 3_600`
    - Alice's `pendingRewards = _staker.pendingRewards + _rewardsSinceSnapshot = _staker.weight * _rateDifferenceSinceSnapshot / _BASE = 2.5e18 * 24e18 / 1e18 = 60e18`
    - Alice's `rewardPerShareSnapshot = rewardPerShare = 2.5e18`

After the call to `getReward`, `totalRewards = 3_540e18`.

In the end, the value of `totalRewards` includes an amount that is reserved for claiming but never will be, and can only be recovered by a call to `emergencyWithdraw`.

This was an extreme example using convenient numbers to improve readability. In a real-world scenario, the

rewards corresponding to the time between the call to `setRewardAmount` and the first user staking would get stuck in the contract.

Recommendation

Do not update the value of `lastUpdateTime` when `_rewardPerShare` is zero.

```
TypeScript
...
-    if (_rewardPerShare == 0 || _rewardPerShare > rewardPerShare) {
+    if (_rewardPerShare > rewardPerShare) {
      rewardPerShare = _rewardPerShare;
      lastUpdateTime = _lastTimeRewardApplicable();
    }
...
```

IMM-LOW-02

Function `_increaseStake` should revert when the calculated weight is zero #5

Id	IMM-LOW-02
Severity	LOW
Category	Bug
Status	Fixed in c24d86d5139d4e99687c96fa632c08e41d2c26ce

Description

Calls to `_inscreasedStake` should be disallowed when the calculated weight is zero, just as it is implemented in the `_stake` function:

```
TypeScript
...
    if (_weight == 0) revert ZeroWeight();
...
```

This is an extreme case because it can only happen when staking a tiny amount (1 to 3 wei, depending on the lockup period), but all the deposits that fall into this category would not yield rewards to the depositors, as their weight would be zero.

Recommendation

Consider applying this fix:

```
TypeScript
  function _increaseStake(uint256 _index, uint256 _amount, address _user) internal {
    Deposit storage _deposit = deposits[_user][_index];

    if (_deposit.amount == 0) revert InvalidDepositIndex();
    if (_deposit.lockupPeriod > 0) revert CannotIncreaseLockedStake();
    if (_deposit.withdrawAt > 0) revert WithdrawalAlreadyInitiated();
```

```
    // Because the deposit is unlocked, we're calculating the weight with a lockup period of
0
    uint256 _weight = _calculateWeight(0, _amount);
+
+    if (_weight == 0) revert ZeroWeight();
```

Alternatively, consider computing the combined weight of the current position amount (which is already checked to be nonzero) plus the increased amount.

IMM-LOW-03

Function `listDeposits` should not return deleted deposits #4

Id	IMM-LOW-03
Severity	LOW
Category	Bug
Status	Fixed in cbeefd1a9cf055ea6a08bcb912e7f24c08dd78d2

Description

Function `listDeposits` should not return deleted deposits, which are empty and are not actually deposits in the first place.

```
TypeScript
...
    _list = new Deposit[](_batchSize);


    uint256 _index;
    while (_index < _batchSize) {
      _list[_index] = deposits[_user][_startFrom + _index];
      ++_index;
    }
...
```

The deposit's `amount` must be positive, otherwise it should not be included in the list.

Recommendation

Consider applying the following fix:

```
TypeScript
...
-    uint256 _index;
-    while (_index < _batchSize) {
```

```
-      _list[_index] = deposits[_user][_startFrom + _index];
-      ++_index;
-    }
+    uint256 _batchIndex;
+    uint256 _index;
+    while (_batchIndex < _batchSize) {
+      Deposit memory deposit = deposits[_user][_startFrom + _batchIndex];
+      if (deposit.amount > 0) {
+        _list[_index] = deposit;
+        ++_index;
+      }
+      ++_batchIndex;
+    }
+    assembly { mstore(_list, _index) }
...
```

IMM-LOW-04

Deposits with an initiated withdrawal get a wrong value in `calculateAPY` #2

Id	IMM-LOW-04
Severity	LOW
Category	Bug
Status	Fixed in dfd5fa1683a8f378e7076d74a1250b705dcea76a

Description

When a user initiates a withdrawal calling `initiateWithdrawal`, the weight that corresponds to the deposit is subtracted from the total weights through the inner call to `_decreaseStake`.

initiateWithdrawal:
https://github.com/layer3xyz/tge-contracts/blob/9d3616b27735e2ae9a00a7dd0a8dba7302bc18b8/src/contracts/Staking.sol#L193-L209

_decreaseStake:
https://github.com/layer3xyz/tge-contracts/blob/9d3616b27735e2ae9a00a7dd0a8dba7302bc18b8/src/contracts/Staking.sol#L500-L514

`calculateAPY` will return a higher APY value for such a deposit, because `totalWeights` is less than it should be.

```
TypeScript
  function calculateAPY(address _user, uint256 _index) external view returns (uint256 _apy) {
    Deposit memory _deposit = deposits[_user][_index];
    uint256 _weight = _calculateWeight(_deposit.lockupPeriod, _deposit.amount);
    uint256 _rewardPerYear = rewardPerSecond * _12_MONTHS * _BASE * 100;

    _apy = Math.mulDiv(_weight, _rewardPerYear, _deposit.amount * totalWeights);
  }
```

This could cause confusion to the user who initiated the withdrawal, who would see an inflated APY for that

deposit.

Recommendation

Consider handling this special case:

```
TypeScript
   function calculateAPY(address _user, uint256 _index) external view returns (uint256 _apy)
{
    Deposit memory _deposit = deposits[_user][_index];
    uint256 _weight = _calculateWeight(_deposit.lockupPeriod, _deposit.amount);
    uint256 _rewardPerYear = rewardPerSecond * _12_MONTHS * _BASE * 100;
-    _apy = Math.mulDiv(_weight, _rewardPerYear, _deposit.amount * totalWeights);
+    if (_deposit.withdrawAt > 0){
+      _apy = Math.mulDiv(_weight, _rewardPerYear, _deposit.amount * (totalWeights +
_weight));
+    }
+    else{
+      _apy = Math.mulDiv(_weight, _rewardPerYear, _deposit.amount * totalWeights);
+    }
  }
```

IMM-INSIGHT-01

Checks to return an empty list in `listDeposits` can be improved #3

Id	IMM-INSIGHT-01
Severity	INSIGHT
Category	Informational
Status	Fixed in d1d976878c2189099bf5b775efca613cb5852154

Description

Calls to `listDeposits` return early if the start index is greater than the total deposits:

```
TypeScript
...
    // Return an empty array if non-existent user or no deposits
    if (_startFrom > _totalDeposits) {
      return _list;
    }
...
```

This check is wrong, because the case where `_startFrom == _totalDeposits` should also return an empty array. And also the case where `_batchSize == 0` could be added to the if statement so that the function also returns early in that case.

Recommendation

Consider the following fix:

```
TypeScript
...
-    if (_startFrom > _totalDeposits) {
+    if (_startFrom >= _totalDeposits || _batchSize == 0) {
       return _list;
```

```
    }
...
```

IMM-INSIGHT-02

Misleading error message in `initiateWithdrawal` function #1

Id	IMM-INSIGHT-02
Severity	INSIGHT
Category	Informational
Status	Acknowledged

Description

There is a check in function `initiateWithdrawal` that makes sure `lockupPeriod` is zero:

```
TypeScript
    if (_deposit.lockupPeriod > 0) revert DepositLocked();
```

But the error thrown can be misleading for the caller, because a deposit with `lockupPeriod` greater than zero but `unlockAt <= block.timestamp` is actually not locked, but would cause the revert anyway.

Note that the other places in the contract where the `DepositLocked` error is thrown, the condition to meet is that `deposit.unlockAt > block.timestamp`.

Recommendation

Consider using a new error for this case.