Lila Finance

Security Review

February 21, 2024

Conducted by:
Blckhv, Independent Security Researcher
Slavcheww, Independent Security Researcher

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

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

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

Lila Finance employs a double-sided vault strategy connecting fixed-rate and variable-rate users. Fixed-rate users receive a fixed yield on their collateral paid by variable-rate users. Variable-rate users receive the actual yield generated from the fixed-rate users' collateral. All users receive a tradeable non-fungible token (NFT) representing their position.

Overview

Project	Lila Finance
Repository	https://github.com/Lila-Finance/protocol/
Commit Hash	def392e697ba12c33986b19bc3c45cd41dfe4927
Resolution	c34703e346415046901ea1e2c8f948f6f871f5e5
Date	February 10 – February 11, 2024 February 17 – February 21, 2024

Scope

LilaOracle.sol
LilaPoolsProvider.sol
LilaPosition.sol
/proxies/*

Issues Found

Critical Risk	2
High Risk	3
Medium Risk	5
Low/Info Risk	17

5. Findings

5.1. Critical severity

5.1.1. emergencyWithdraw is internal without a way to prematurely close a position

Severity: Critical Risk

Context: LilaPoolsProvider.sol#L132

Description: emergencyWithdraw is for users who want to close their position midway. However, the function has the wrong access modifier making it unavailable to be called.

```
function emergencyWithdraw(uint256 tokenId) internal whenNotPaused
```

Recommendation: Make the function external.

Resolution: Fixed

5.1.2. Stargate pool lacks approval which will make it impossible to open a position

Severity: Critical Risk

Context: StargateProxy.sol#L83

Description: All deposits made through the StargateProxy will fail because the LPStaking contract is not approved.

The deposit function in the StargateProxy first adds liquidity to the Stargate and then deposits into the corresponding pool. However, it needs to approve the pool in order to transfer the tokens from the StargateProxy to the LPStaking.

```
function deposit(uint256 amount, uint256 tokenId) external onlyProvider{
    require(amount > 0, "Invalid amount");
    address router = lpToken.router();
    uint256 poolId = lpToken.poolId();

    token.safeTransferFrom(_msgSender(), address(this), amount);

    if (token.allowance(address(this), router) < amount) {
        token.approve(router, type(uint256).max);
    }

    uint256 prevBalance = lpToken.balanceOf(address(this));
    IStargateRouter(router).addLiquidity(poolId, amount, address(this));
    uint256 lpTokenAmount = lpToken.balanceOf(address(this)) - prevBalance;

    // @audit missing approve
    pool.deposit(poolId, lpTokenAmount);

    totalShare += lpTokenAmount;
    userShares[tokenId] += lpTokenAmount;
}
```

```
function deposit(uint256 _pid, uint256 _amount) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    updatePool(_pid);
    if (user.amount > 0) {
        uint256 pending = user.amount.mul(pool.accStargatePerShare).div(1e12).sub(user.rewardDebt);
        safeStargateTransfer(msg.sender, pending);
    }
    pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount); // @audit this will fail
    user.amount = user.amount.add(_amount);
    user.rewardDebt = user.amount.mul(pool.accStargatePerShare).div(1e12);
    lpBalances[_pid] = lpBalances[_pid].add(_amount);
    emit Deposit(msg.sender, _pid, _amount);
}
```

Recommendation: Approve the pool before calling deposit().

Resolution: Fixed

5.2. High severity

5.2.1. Ownable constructor is not inherited in multiple contracts which will brick the deployment

Severity: High Risk

Context: LilaOracle.sol, LilaPoolsProviders.sol, LilaPosition.sol, /proxies/*

Description: An important change was introduced to Ownable from OpenZeppelin after version 5.0, owner has to be passed as a constructor argument instead of automatically being assigned to msg.sender.

In all Lila contracts:

- LilaOracle
- LilaPoolsProviders
- LilaPosition
- all proxies

Ownable's constructor is not called at all which will make it impossible to deploy the contracts, until the mistake is fixed.

Recommendation: Modify the contracts mentioned above to comply with the latest OZ version by passing the admin as a parameter to the constructor of Ownable.

Resolution: Fixed

5.2.2. First depositor can grief the users

Severity: High Risk

Context: AaveProxy.sol#L67

Description: In AaveProxy.sol, GranaryProxy.sol and RadiantProxy.sol, first depositor can inflate the shares calculation by depositing 1 wei of asset, then deposit to the proxies the correspondent lpToken from the chosen pool (AToken in these cases). By doing this all future deposits need to be more than the first depositor transfer, otherwise it will round down to 0.

Proof of Concept:

1. An attacker is the first depositor depositing 1 wei
2. Then transfer 100 aToken to the Proxy.
3. If Bob deposits 90 tokens, the share calculation will round down to 0, but his tokens will be transferred inside the Proxy

```
function deposit(uint256 amount, uint256 tokenId) external onlyProvider{
    require(amount > 0, "Invalid amount");
    token.safeTransferFrom(_msgSender(), address(this), amount);

    if (token.allowance(address(this), address(pool)) < amount) {
        token.approve(address(pool), type(uint256).max);
    }

    uint256 prevBalance = aToken.balanceOf(address(this));
    pool.supply(address(token), amount, address(this), 0);
    uint256 aTokenAmount = aToken.balanceOf(address(this)) - prevBalance;
    uint256 shareAmount = aTokenAmount;
    if (totalShare > 0 && prevBalance > 0) {
        shareAmount = (aTokenAmount * totalShare) / prevBalance; // @audit inflating the vault by depositing aToken
        // Second time: 90 tokens * 1wei / (1000 tokens + 1 wei) = 90 / (1000 + 1wei) = 0
    }
    totalShare += shareAmount;
    userBalance[tokenId] += aTokenAmount;
    userShares[tokenId] += shareAmount;
}
```

4. Bob cannot withdraw anything because aTokenAmount to withdraw is based on the users' shares, which will pass 0 as amount, causing the Aave pool to revert.
5. Attacker will only be able to withdraw his first deposited amount which is 1 wei while the aTokens, directly send to the AaveProxy, will be claimable from the contract owner, also Bob and all future depositors will not be able to withdraw their assets.

Recommendation: There are 3 possible solution:

- The first one is the team to make the first deposit.
- Second one is to utilize the approach used in ERC4626 from OZ implementation - https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC4626.sol#L232
- Third is to remove the share accounting, since its always 1:1 in the current implementation and just double the accounting logic.

Resolution: Acknowledged

5.2.3. Hardcoded decimals in LilaOracle will break the payment accounting

Severity: High Risk

Context: LilaOracle#L13

Description: LilaOracle uses hardcoded decimals for all of the assets that can be deposited in the protocol, some of them are:

USDC (6) DAI (18) wBTC (8)

Prices at which deposits and withdraws will happen will be wrong and all the accounting will be broken.

```
uint256 public decimals = 1e11;

function getReturn(address strategy, uint32 totalPayments, uint32 rateIndex, uint256 amount) external view override returns
    bytes32 key = getKey(strategy, totalPayments, rateIndex);
    return (amount * numeratorList[key]) / decimals;
}
```

Recommendation: Modify the LilaOracle contract to support multiple assets and scale the prices by the right decimals. It can be achieved with a mapping containing all the tokens, or the token can be passed as a parameter and divide on its decimals.

Resolution: Fixed

5.3. Medium severity

5.3.1. Position NFT can be locked into non-compliant smart contract

Severity: Medium Risk

Context: LilaPoolsProvider.sol#L66

Description: When a position is opened NFT representing the information about it is minted to the caller, but it uses mint instead of safeMint and therefore doesn't perform a checkOnERC721Received call, which is crucial to ensure that if the caller is a smart contract it is aware of receiving ERC721 tokens.

That can result in stuck tokens, without a way to close a position or sell it in a marketplace.

Recommendation: Use safeMint instead of normal mint it will guarantee that the receiver will always be able to perform the needed operations with it.

From OpenZeppelin ERC721 implementation regarding the mint:

WARNING: Usage of this method is discouraged, use {_safeMint} whenever possible

Resolution: Fixed

5.3.2. SafeApprove is removed from OpenZeppelin after 5.0

Severity: Medium Risk

Context: LilaPoolsProvider.sol#L70

Description: LilaPoolsProvider.sol::deposit uses safeApprove to authorize the proxy when depositing assets. However, since OpenZeppelin changed safeApprove to forceApprove in v5.0, LilaPoolsProvider.sol cannot be deployed.

Recommendation: Change safeApprove to forceApprove

Resolution: Fixed

5.3.3. After last payment, the position is not burnt which can make user continue receiving rewards

Severity: Medium Risk

Context: LilaPoolsProvider.sol#L103-104

Description: After receiving his last reward depositor in normal scenarios should not be able to receive more than the totalPayments, but the current implementation of the withdrawal functions, withdraw and emergencyWithdraw, doesn't burn the NFT representing the position but instead set the claimedPayments to be equal to totalPayments of the pool.

This is problematic because there is a function updatePool from which the admin can extend the totalPayments and intentionally or not allow users with already claimed positions to continue receiving rewards.

Recommendation: Make the LilaPosition NFT burnable, then in withdraw and emergencyWithdraw modify the if check responsible for closing the positions:

```
if (paymentsPassed >= pool.totalPayments) {
        paymentsPassed = pool.totalPayments;
        IProxy(pool.strategy).withdraw(tokenId);
        reward += position.amount;
+       lilaPosition.burn(tokenId);
    }
```

That will effectively prevent users from claiming more than they are allowed.

Resolution: Fixed

5.3.4. USDT markets will always revert on deposit

Severity: Medium Risk

Context: /proxies/*::deposit()

Description: All proxies use `ERC20::approve`. This will cause all deposits to `revert` if the token was a non-standard token that has different function signature for `approve()`. Tokens like USDT cannot be used as market tokens, because they do not return bool.

Recommendation: For the needs of the protocol and given that USDT will be one of the used tokens, consider use `SafeERC20`'s `forceApprove` method instead to support it.

Resolution: Fixed

5.3.5. LilaPosition inherits wrong ERC2981 and ERC165 implementations

Severity: Medium Risk

Context: LilaPosition.sol#L9

Description: Owners will not receive any royalty fees because ERC2981 is not implemented correctly.

Contract will not behave as expected due to not inheriting ERC2981 which will make supportsInterface return false on the calling contracts such as OpenSea and not indicate any royalties.

Recommendation: ERC2981 should be inherited by LilaPositions, also supportsInterface should be overridden. Also _setDefaultRoyalty should be called in the constructor to force apply the payments to all tokens minted.

Resolution: Fixed

5.4. Low/Info severity

5.4.1. User will not be able to withdraw if is blacklisted on USDC/USDT

Severity: Low Risk

Context: LilaPoolsProvider.sol#L119

Description: Users can be blocked from receiving rewards and closing their positions if they deposit into a USDC pool and they get blacklisted. If this happens safeTransfer in the LilaPoolsProvider will revert and the position won't be closeable.

Recommendation: Allow the user to specify another address to receive the initial deposit and reward.

Resolution: Acknowledged

5.4.2. Tokens with non-standard `permit()` can't be used in other chains

Severity: Low Risk

Context: LilaPoolsProvider.sol#L53

Description: LilaProtocol will be deployed on both Arbitrum and Optimism chains. Tokens that the team plans to use in both chains are the following:

- USDC: standart permit
- DAI: standart permit
- wBTC: no permit
- WETH: no permit
- Frax: standart permit
- USDC.e: no permit

- wstETH: no permit

All of these tokens have ERC2612-compliant permit functions in the Arbitrum chain, but on Optimism wBTC, wETH, USDC.e, and wstETH have no permit function, and will be impossible to open a position with them.

Recommendation: Having a normal deposit function has many more benefits than one with a permit signature + it is more widely adopted and offers more flexibility.

In our opinion using a permit in the context of LilaFinance brings only limitations and potential problems and its replacement has to be considered.

5.4.3. Admin can brick pool by updating it

Severity: Low Risk

Context: LilaPoolsProvider.sol#L201

Description: Admin can update existing pool:

```
function updatePool(
    uint32 _index,
    uint256 _maxAmount,
    address _strategy,
    uint64 _frequency,
    uint32 _totalPayments,
    uint32 _rateIndex
)
    public
    onlyOwner
{
    require(_index < poolList.length, "Index out of bounds");
    address _asset = address(IProxy(_strategy).token());
    poolList[_index] = Pool({
        maxAmount: _maxAmount,
        strategy: _strategy,
        asset: _asset,
        payoutFrequency: _frequency,
        totalPayments: _totalPayments,
        rateIndex: _rateIndex
    });
    emit PoolUpdated(_index, _maxAmount, _strategy, _asset, _frequency, _totalPayments, _rateIndex);
}
```

He can change all the important pool variables and cause various damage to both users and pool providers contract.

For example, he can change the underlying asset token which will allow users to receive rewards in different token - if user opens position with 10 USDC then pool.asset is changed to wETH in the system he will appear as a 10 wETH depositor instead and he will start receiving rewards in wETH. But when the position is being closed user's initial deposit amount will be locked because there are no shares in the userShares mapping of the updated strategy contract.

```
function _withdraw(uint256 shareAmount, uint256 tokenId) internal {
    ...
    userShares[tokenId] -= shareAmount; //@audit will revert there
    userBalance[tokenId] = 0;
    totalShare -= shareAmount;
}
```

Recommendation: Completely remove the option to change pool's strategy address or allow it only for pools without opened positions.

Resolution: Acknowledged

5.4.4. ERC777 tokens used as pool asset will lead to reentrancy and LilaPoolsProvider will be drained

Severity: Low Risk

Context: LilaPoolsProvider.sol#L167-168

Description: Certain patterns are not followed in emergencyWithdraw and can lead to malicious users draining the entire balance of LilaPoolsProvider if tokens with callbacks are used as pool assets.

- CEI is not followed - safeTransfer is called before claimedPayments are updated to be equal to the totalPayments, indicating that the position is closed.
- nonReentrant modifier is missing.

In case ERC777 is used, a user can open a position and wait only 1 payment period, then he will call emergencyWithdraw to prematurely close his position. The contract execution will begin taking fees and calculating the payments after that safeTransfer is called sending the initial deposit + payment for the 1 period, but in beforeTokenTransfer the recipient will call emergencyWithdraw until all the balance of the caller contract is drained because claimedPayments is still equal to the old value.

```
function emergencyWithdraw(uint256 tokenId) internal whenNotPaused {
...
    IERC20(pool.asset).safeTransfer(_msgSender(), amount);
    IERC20(pool.asset).safeTransfer(treasury, fee);

    lilaPosition.updatePoolPayouts(tokenId, uint32(pool.totalPayments));
}
```

Recommendation: Even if ERC777 tokens are going to be supported, inverse the order of the operations in the emergencyWithdraw function to first update the claimedPayments and then transfer the funds to the users. Also nonReentrant modifier should be applied.

5.4.5. LilaPoolsProvider::initialize can be frontrun

Severity: Low Risk

Context: LilaPoolsProvider.sol#L38

Description: initialize function can be frontrun on chains with public mempools, such as Optimism, where the Lila team plans to deploy. It is used to set protocol critical addresses such as lilaPosition and treasury. An eventual malicious address assigned to one of these variables can lead to bad consequences, such as draining the treasury pool.

Recommendation: We advise the team to write their deployment scripts to be able to atomically perform the contract instantiation instead of relying on submitting all the contracts in a certain sequence and not being frontrun.

5.4.6. admin can cause hash collision in LilaOracle::getKey by configuring pools with same strategy

Severity: Low Risk

Context: LilaOracle.sol#L21

Description: abi.encodePacked is prone to a hash collision if arguments of the same type are placed next to each other. For example:

- "AA", "BBB"

will produce the same hash as

- "AAB", "BB"

The same pattern can be seen in LilaOracle::getKey, the admin can cause that type of collision when creating a pool with the same strategy address, but.

```
function getKey(address strategy, uint32 totalPayments, uint32 rateIndex) public pure returns (bytes32) {
    return keccak256(abi.encodePacked(strategy, totalPayments, rateIndex)); //note encodePacked is discouraged, can lead to collisio
}
```

Recommendation: abi.encode should be used as it automatically adds padding for inputs shorter than 32 bytes to avoid collisions.

5.4.7. Missing zero address checks in several points

Severity: Low Risk

Context: LilaPoolsProvider.sol

Description: Zero address check must be performed in setter functions and the constructors, otherwise, there is a risk of leaving some variables uninitialized. Currently, only contracts passed as interfaces are checked for zero addresses. Treasury and provider are not verified and can lead to loss of rewards for the protocol.

Recommendation: Consider checking for zero address check in all the functions accepting address as argument.

- All the constructors
- setProvider
- setTreasury

5.4.8. No cap at emergencyWithdrawFee can cause users to lose all their assets

Severity: Low Risk

Context: LilaPoolsProvider.sol#L148

Description: Missing cap in the setEmergencyWithdrawFee can make the users lose all their deposited tokens and pay them to the treasury as fees. Admin can intentionally or unintentionally set emergencyWithdrawFee to any number and game the users.

There are 2 scenarios that can happen from weird fee assigned:

- Fee up to 99 - will make the users pay 99% of their initial deposit as a fee and receive the rest.
- Fee above 99 - will DoS the emergencyWithdraw function due to dividing by zero when calculating the proportional amount that the user has to pay as a fee.

```
uint256 amount = position.amount * (100 - emergencyWithdrawFee) / 100;
```

Recommendation: As mentioned in the docs that 4% will be taken from every user who wants to prematurely close his position and withdraw his tokens, consider adding a cap in the setEmergencyWithdrawFee to validate if the fee is in the acceptable ranges.

5.4.9. batchWithdraw is likely to run out-of-gas

Severity: Low Risk

Context: LilaPoolsProvider.sol#L178

Description: The function `batchWithdraw()` will repeat 72 times. First, it checks 64 times to see if there are any duplicate IDs. Then, it goes through all the IDs, 8 times and calls the `withdraw()` function for each. This could lead to an Out of Gas (OOG) error because the `withdraw()` logic calls the strategy, which in turn interacts with their external contracts.

Recommendation: The first loop can be simplified by setting up a mapping at the beginning of the function. Then, in the loop, we can check if each element exists in the mapping. If it doesn't, we set it; if it does, we revert it. This approach reduces the number of iterations in the loop to just 8.

5.4.10. Use Ownable2Step instead of Ownable

Severity: Low Risk

Context: LilaPoolsProvider.sol, LilaOracle.sol, LilaPosition.sol

Description: It is good practice to use `Ownable2Step` because it reduces the chance of transferring the ownership to an address that is not under the control of Lila team and lose access to all the `onlyOwner` functions. The benefit of using is that the ownership is not immediately transferred to the passed address but a confirmation from the new owner is required.

Recommendation: Consider replacing the `Ownable` used across all the contracts with `Ownable2Step` in case ownership will changed more frequently.

5.4.11. `removePool` can be called to one containing existing deposits and wipe out all payment information

Severity: Low Risk

Context: LilaPoolsProvider.sol#L228

Description: If the admin calls `removePool` all the deposits will be locked without a way to be withdrawn until the pool is added back.

```
function removePool(uint32 _index) public onlyOwner {
    require(_index < poolList.length, "Index out of bounds");
    delete poolList[_index];
    emit PoolRemoved(_index);
}
```

Recommendation: Before deleting a pool, consider withdrawing all funds belonging to the users.

5.4.12. Smart wallets such as Gnosis Safe can't deposit due to lack of non-permit deposit

Severity: Information Risk

Context: LilaPoolsProvider.sol#L53

Description: Having only one deposit function requiring a permit signature greatly limits the number of users, that can use the Lila functionality. Certain smart wallets such as Gnosis Safe and Argent have no functionality to produce such signatures and depositing from them will be impossible.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/8b12f83a702210cdeff862c837ffb338811c31a4/contracts/token/ERC20/extensions/IERC20Permit.sol#L39-L40

Recommendation: Consider adding a normal **deposit** function which requires normal approval from the user instead.

5.4.13. setProvider in LilaPositions can be moved into the constructor

Severity: Information Risk

Context: LilaPosition.sol#L21

Description: In LilaPosition contract setProvider function is not needed and lilaProvider can be set in the constructor. There is no need to have an additional setter function which can be called only once. Additionally, making the lilaProvider immutable and setting it at deployment will lower the gas cost.

Recommendation:

- Remove lilaProviderSet
- Make lilaProvider immutable
- Pass it as constructor argument
- Remove setProvider function

5.4.14. Unnecessary require check when withdrawing from LilaPoolsProviders

Severity: Information Risk

Context: LilaPoolsProvider.sol#L115

Description: Balance checks in `LilaPoolsProviders` withdraw functions are redundant and serve no purpose. They only verify that the `balanceOf` the contract is enough to cover the payment, but the effect will be the same if they are removed because Solidity has underflow protection and will revert if the balance is not enough. Every `require` statement only increases the code complexity and gas consumption.

Recommendation: Remove the balance checks performed in `withdraw` and `emergencyWithdraw` and leave the Solidity underflow protection to do the work.

5.4.15. uint256 can be used instead of uint

Severity: Information Risk

Context: AaveProxy.sol#L96

Description: Make sure to consistently use the same type semantics in the whole project, and change the `uint` to `uint256`.

5.4.16. Wrong and missing events in important functions

Severity: Information Risk

Context: LilaPosition.sol

Description: All described below functions lack events.

All proxies:

- setProvider
- setTreasury

LilaOracle.sol

- setDecimals

LilaPosition.sol

- setFee
- updatePoolPayouts

Emitting events in important functions is a good practice as the front-end applications rely only on them as a source of information.

Also, require statements have to be improved as the following mistakes were observed:

- ambiguous and wrong error messages
- no error messages at all

Recommendation: Add appropriate events to indicate when these functions are called, and also improve the existing ones by indexing the arguments of type address.

Regarding the require statements, try to give as much valuable info to the caller, avoid messages like "Too Early" for example.

5.4.17. public functions not called by the contract should be declared external instead

Severity: Information Risk

Context: LilaPoolsProvider.sol, LilaOracle.sol, LilaPosition.sol

Description: There are functions that are `public` but are never invoked in the contract.

Recommendation: Change the visibility to `external`.