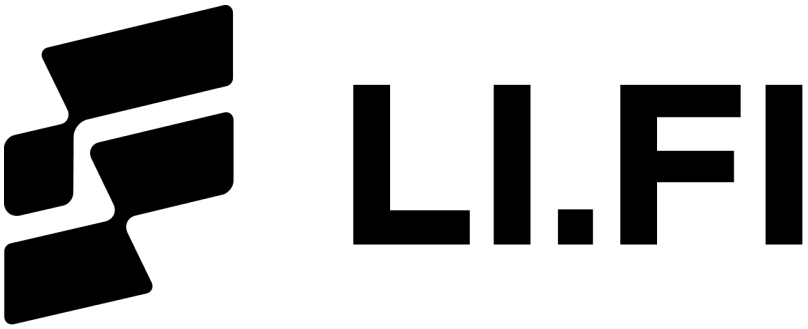

LI.FI Security Review

Patcher(v1.0.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

August 1, 2025

Contents

1 About Researcher **2**

2 Disclaimer **2**

3 Scope **2**

4 Risk classification **2**
4.1 Impact . . . 2
4.2 Likelihood . . . 3
4.3 Action required for severity levels . . . 3

5 Executive Summary **3**

6 Findings **4**
6.1 High Risk . . . 4
6.1.1 Frontrunning could lead to loss of caller funds . . . 4
6.2 Medium Risk . . . 4
6.2.1 Function `_getDynamicValue()` casts random return types into uint256 . . . 4
6.3 Gas Optimization . . . 5
6.3.1 Redundant return param in `_depositAndApprove()` . . . 5
6.3.2 Assign instead of copying data from calldata to memory save gas . . . 5
6.4 Informational . . . 5
6.4.1 Avoid unlimited approvals to unknown targets . . . 5
6.4.2 Remove unchecked loop increments . . . 6
6.4.3 Missing events . . . 6
6.4.4 Add documentation about usage of entire balance . . . 6
6.4.5 Add documentation about non refunds handling . . . 6

1 About Researcher

Sujith Somraaj is a distinguished security researcher and protocol engineer with over eight years of comprehensive experience in the Web3 ecosystem.

In addition to working as a Security researcher at Spearbit, Sujith is also the security researcher and advisor for leading bridge protocol LI.FI and also is a former founding engineer and current CISO at Superform, a yield aggregator with over $170M in TVL.

Sujith has experience working with protocols / funds including Layerzero, Edge Capital, Berachain, Optimism, Ondo, Sonic, Monad, Blast, ZkSync, Decent, Drips, SuperSushi Samurai, DistrictOne, Omni-X, Centrifuge, Superform-V2, Tea.xyz, Paintswap, Bitcorn, Sweep n' Flip, Byzantine Finance, Variational Finance, Satsbridge, Rova, Horizen, Earthfast and Angles

Learn more about Sujith on sujithsomraaj.xyz or on cantina.xyz

2 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of that given smart contract(s) or blockchain software. i.e., the evaluation result does not guarantee against a hack (or) the non existence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, I always recommend proceeding with several audits and a public bug bounty program to ensure the security of smart contract(s). Lastly, the security audit is not an investment advice.

This review is done independently by the reviewer and is not entitled to any of the security agencies the researcher worked / may work with.

3 Scope

- src/Periphery/Patcher.sol(v1.0.0)

4 Risk classification

Severity level	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 loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.

Medium global losses <10% or losses to only a subset of users, but still unacceptable.

Low losses will be annoying but bearable — applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

4.2 Likelihood

High almost certain to happen, easy to perform, or not easy but highly incentivized

Medium only conditionally possible or incentivized, but still relatively likely

Low requires stars to align, or little-to-no incentive

4.3 Action required for severity levels

Critical Must fix as soon as possible (if already deployed)

High Must fix (before deployment if not already deployed)

Medium Should fix

Low Could fix

5 Executive Summary

Over the course of 6 hours in total, LI.FI engaged with the researcher to audit the contracts described in section 3 of this document ("scope").

In this period of time a total of 9 issues were found. This review focussed only on the changes made from the previous version, not the code on its entirety.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	1536e96
Audit Timeline	July 30, 2025
Methods	Manual Review
Documentation	High
Test Coverage	Medium-High

Issues Found	
Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	0
Gas Optimizations	2
Informational	5
Total Issues	**9**

6 Findings

6.1 High Risk

6.1.1 Frontrunning could lead to loss of caller funds

Context: Patcher.sol#L72, Patcher.sol#L107

Description: The two variants of execution functions where tokens are deposited from `msg.sender` and executed with patches are prone to front-running attacks.

Callers should approve their tokens for use by `Patcher.sol`, which can be stolen by a malicious actor executing a front-run attack.

PoC: The following PoC demonstrates how a user's approval gets abused by a malicious actor.

```
function test_frontrunning() public {
        address user = address(0x1234);
        address malicious = address(0x420);

        uint256 tokenBalance = 1000 ether;
        token.mint(user, tokenBalance);
        valueSource.setValue(tokenBalance);

        bytes memory valueGetter = abi.encodeWithSelector(
            valueSource.getValue.selector
        );

        vm.prank(user);
        token.approve(address(patcher), tokenBalance);

        uint256[] memory offsets = new uint256[](1);
        offsets[0] = 68;

        bytes memory data = abi.encodeWithSelector(
            token.transferFrom.selector,
            user,
            malicious,
            tokenBalance
        );

        vm.prank(malicious);
        patcher.executeWithDynamicPatches(address(valueSource), valueGetter, address(token), 0, data,
        ↪  offsets, false);
    }
```

Recommendation: Since the execution targets are not whitelisted, the fix for this issue would be to add a note to users that any approvals to this contract would result in loss of funds and document this as accepted risk.

LI.FI: Acknowledged. Added clear documentation warnings about the frontrunning risk in both the contract NatSpec comments and the Patcher.md documentation in 68b91b10. The warnings explicitly state that this is a known and accepted risk when using the depositAndExecute functions.

Researcher: Documentation added as suggested.

6.2 Medium Risk

6.2.1 Function `_getDynamicValue()` casts random return types into uint256

Context: Patcher.sol#L195

Description: The function `_getDynamicValue()` is used to read an **uint256** value from a target address. However, if the target returns a datatype that's not uint256, e.g., bytes, the function casts them into random **uint256** values,

which can potentially lead to loss of user funds.

Recommendation: While a partial fix is to validate if the length of return data is 32, which will not protect against 32-byte encoded types like bool and address.

Nonetheless, the behavior should be documented.

LI.FI: Fixed in 68b91b10

Researcher: As mentioned in the description length based check provides better immunity, however other 32 length types like bool will always be interpreted as valid return values, which is an acknowledged risk.

6.3 Gas Optimization

6.3.1 Redundant return param in `_depositAndApprove()`

Context: Patcher.sol#L171

Description: The return parameter **amount** of the `_depositAndApprove()` function remains unused in the whole Patcher context and is redundant.

Recommendation: Consider removing the unused return parameter.

LI.FI: Fixed in 6b3671cf

Researcher: Verified fix

6.3.2 Assign instead of copying data from calldata to memory save gas

Context: Patcher.sol#L274-277, Patcher.sol#L310-L313

Description: The patcher contract uses yul to copy data from calldata into memory for applying patches. However a direct assignment is found to be cheaper in my local compiler / tests.

Recommendation: Consider removing the yul to reduce complexity and increase gas efficiency as follows:

```
- bytes memory patchedData = new bytes(data.length);
- assembly {
-     calldatacopy(add(patchedData, 0x20), data.offset, data.length)
- }
+ bytes memory patchedData = data;
```

LI.FI: Fixed in 69686f54

Researcher: Verified fix

6.4 Informational

6.4.1 Avoid unlimited approvals to unknown targets

Context: Patcher.sol#L184

Description: Although `Patcher.sol` is not intended to hold any funds, and there are alternative methods to withdraw locked funds from the contract, unlimited approvals are not always necessary. It's advisable to approve only when needed and reset approvals after the transaction to maintain clarity and security.

Recommendation: Consider documenting this behavior as potential risks, and if possible consider resetting approvals post external call.

LI.FI: Fixed in 699218d7

Researcher: Verified fix

6.4.2 Remove unchecked loop increments

Context: Patcher.sol#L358, Periphery/Patcher.sol#L340

Description: In newer compiler versions, loop increments are unchecked by default under the hood. Explicit usage results in no gas benefits but reduces code quality.

Recommendation: Consider removing the unchecked loop increments to improve code quality.

LI.FI: Fixed in 3fe22c81

Researcher: Verified fix

6.4.3 Missing events

Context: Global

Description: All four functions that patch and execute the data lack events.

Recommendation: Add events for off-chain tracking to the four execution functions.

LI.FI: Fixed in 68f8b1c0

Researcher: Verified fix

6.4.4 Add documentation about usage of entire balance

Context: Patcher.sol#L72, Patcher.sol#L107

Description: Both the `depositAndExecuteWithMultiplePatches()` and `depositAndExecuteWithDynamicPatches()` functions, which utilize the `_depositAndApprove()` internal method, will transfer the caller's entire balance, regardless of the amount needed for the transaction or specified in the calldata.

This behavior is undocumented and might result in loss of funds if used without caution.

Recommendation: Consider documenting this behavior for more clarity to integrators / users.

LI.FI: Fixed in f25baadc

Researcher: Verified fix

6.4.5 Add documentation about non refunds handling

Context: Global

Description: All functions in the Patcher contract, accepting native or ERC20 tokens, do not manage tokens sent in excess. These excess tokens are locked in the patcher contract and could be stolen.

Excessive token condition can happen:

- If the dynamic value patched is less than the deposited tokens
- If the target does not spend all the approved tokens
- If the `msg.value` is greater than the value parameter

**Recommendation: ** Please make sure to add explicit documentation about this behavior. If possible, ensure the `msg.value` always equals the value parameter as a sanity check.

LI.FI: Fixed in f25baadc

Researcher: Verified fix