9LivesLabs Security Review

Reviewers
0xnirlin, Lead
Amar Fares, Lead

1 Executive Summary

MEM-Tech reached out and engaged with 9LivesLabs to review MEM-Tech.

Repository	Commit
MEM-Tech	3c75e10e846874119486350b38fd36c375ff8f4c

Summary

Type of Project	Bridge
Engagement Date	May 5th, 2024
Methods	Manual Review
Available Documentation	Sufficient

Total Issues

Critical Risk	1
High Risk	0
Medium Risk	2
Low Risk	0
Gas Optimizations and Informational	0

Contents

2 9LivesLabs

9LivesLabs is a team of smart contract security researchers comprising of 0xnirlin & Amar Fares. Together, we help secure the Web3 ecosystem. We offer security reviews and related services to Web3 projects.

3 Introduction

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of brink according to the specific commit by a two person team. Any modifications to the code will require a new security review.

4 Findings

4.1 Critical Risk

4.1.1 Upgradeable pattern is not implemented correctly which breaks the upgradeability of contracts

Severity: Critical

Context: `bridge.sol#L148-L180`

Description:

When building upgradeable contracts, the implementation should not set the storage variables in the constructor as it sets them in implementation storage instead of proxy storage, and since we want the storage variables to persist through upgrades, this is problematic.

Currently, there is the following constructor in the bridge contract:

```
constructor(
     IERC20 _btoken,
     address _oracleAddress,
     address _linkTokenAddr,
     address _treasuryAddr,
     address _cronjobAddr,
     string memory _jobId,
     string memory _baseEndpoint,
     uint256 _ofee,
     uint256 _bLockfee,
     uint256 _minBAmount,
     uint256 _unlockFlatFee
 ) ConfirmedOwner(msg.sender) {
     address uinitialized = address(0);
     require(
         _oracleAddress != uinitialized &&
             _linkTokenAddr != uinitialized &&
             _treasuryAddr != uinitialized &&
             _cronjobAddr != uinitialized &&
             address(_btoken) != uinitialized
     );
     token = _btoken;
     treasury = _treasuryAddr;
     cronjobAddress = _cronjobAddr;
     minBamount = _minBAmount;
     setChainlinkToken(_linkTokenAddr);
     setChainlinkOracle(_oracleAddress);
     setJobId(_jobId);
     setFeeInHundredthsOfLink(_ofee);
     bridgeLockFee = _bLockfee;
     unlockFlatFee = _unlockFlatFee;
     baseEndpoint = _baseEndpoint;
 }
```

This will set the variables in the implementation storage but the correct way is for them to be stored in the proxy's storage.

Recommendation:

Implement the initialize function that will be called via proxy after deployment and inherit from OZ initializer.

4.2 Medium Risk

4.2.1 Wrong function signatures are used for Chainlink Client v0.8

Severity: Medium

Context: bridge.sol#L173-L174

Description:

Chainlink Client v0.8 uses different signatures from v0.6, but in the bridge contract, signatures from v0.6 are also used which will never work.

For example, in this snippet in the constructor:

```
    setChainlinkToken(_linkTokenAddr);
    setChainlinkOracle(_oracleAddress);
```

The correct signatures are `_setChainlinkToken` and `_setChainlinkOracle` for v0.8.

Recommendation: Use the correct signatures for v0.8.

4.2.2 No transfer limit set in bridge.js can lead to small amounts of tokens bridged back getting stuck

Severity: Medium

Context: bridge.sol#L289

Description:

In the solidity code, when unlocking the tokens that are bridged back to EVM from MEM, there is following check:

```
    require(amount > unlockFlatFee, "err_invalid_amount");
```

This means the amount is less than `unlockFlatFee`, now the architecture is such that if such a small amount is bridged, more amount cannot be added to the same transaction so it will be forever locked, which is possible because there is no such minimum transfer check in MEM `bridge.js` code. Any amount can be transferred as shown in the following snippet:

```
if (input.function === "initiateUnlock") {
  const { caller, sig, amount } = input;

  const bigIntAmount = BigInt(amount);

  const normalizedCaller = _normalizeCaller(caller);
  ContractAssert(
    bigIntAmount <= BigInt(state.balances[normalizedCaller]),
    "err",
  );

  await _moleculeSignatureVerification(normalizedCaller, sig);

  state.unlocks.push({
    address: normalizedCaller,
    mid: sig,
    amount: amount,
  });

  const newBalance = BigInt(state.balances[normalizedCaller]) - bigIntAmount;

  state.balances[normalizedCaller] = newBalance.toString();

  return { state };
}
```

Recommendation: Implement minimum amount check in `bridge.js`.