9LivesLabs Security Review

Reviewers
0xnirlin, Lead
cats, Lead

March 17, 2024

1 Executive Summary

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

Repository	Commit
MEM-tech	1c16878e5f8de3471fdcd42e0eee99fb0510f3b2

Summary

Type of Project	Bridge
Engagement Date	March 9th 2024
Methods	Manual Review
Available Documentation	High

Total Issues

Critical Risk	0
High Risk	2
Medium Risk	2
Low Risk	4
JS Issues	2
Gas Optimizations and Informational	7

Contents

2 9LivesLabs

9LivesLabs is a team of smart contract security researchers comprising of 0xnirlin & cats. 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 High Risk

4.1.1 Permissionless `validateUnlock()` allows anyone to spam empty requests and use up all designated `oracleFee` funds

Severity: High

Context: `bridge.sol#L90-L130`

Description: The function to validate an unlock is permissionless and the only check made is that the `_memid` is not yet redeemed.

```
    function validateUnlock(
        string calldata _memid
    ) public returns (bytes32 requestId) {
        // memid can be redeemed once
@>      assert(!midIsRedeemed[_memid]);
```

A Chainlink Client request is then created and filled with data, after which the request is pushed to the Chainlink operator.

```
    // Sends the request
    requestId = sendOperatorRequest(req, oracleFee);
    // map requestId to caller
    reqToCaller[requestId] = msg.sender;
    // map the chainlink requestId to memid
    reqToMemId[requestId] = _memid;
    // map the memid redeeming status to false
    midIsRedeemed[_memid] = false;
    return requestId;
```

When a request is sent to an operator, Chainlink charges an `oracleFee`. An issue arises from the fact that since the function is permissionless and doesn't validate if the caller even has an existing lock, anyone could spam this function with random `_memIds` and use up all available `$LINK` token balance designated for paying oracle fees in the contract. When the funds run out the system will cease to function properly and will be temporarily DOS'd until new funds are added by the protocol team.

Recommendation: At the very least, implement access control to validate that only users that have locked funds can call the function. Additional input validation like a mapping from addr => memId is required. Otherwise if a user has locked 1 wei, they can still pass the access control check and spam different `_memIds` which will achieve the same result.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.1.2 A malicious user can grief contract out of oracle balance if the total lock/unlock fee per tx is less than oracle fee per tx

Severity: High

Context: `bridge.sol#L122`

Description:

The bridge fee is currently set at 0.25% of the bridged asset, paid twice when locking and unlocking tokens. There is also an additional oracle fee when validating an unlock.

```
    requestId = sendOperatorRequest(req, oracleFee);
```

If the oracle fee that is paid is not `>= bridgeFee`, the function can be griefed at virtually no cost by anyone using the protocol as intended. For example, a user can split up a 100USDC lock/unlock many multiple different transactions, and since on L2 the gas is negligible, their total bridge fee will still be `0.5% of amount + gas`. But if the oracle fee does not cover that, a malicious user can grief all of the remaining oracle fee balance in the contract while retaining functionality and losing nothing.

Recommendation:

Require for every tx that `bridgeFee >= oracleFee`.

MEM-Tech: For the MEM Bridge launch, the first Bridge instance is to bridge `USDT`, so `bridge.sol` is `usdt.sol` (constructor variables), so to fix the bridgeFees >= gas+oracle fee, we implemented a min bridgeable amount of 5 `USDT`. and the fee changed to 0.25% on `lock` (no oracle fees here), and 2.5 USDT flat fees on `executeUnlock`. Since we plan to launch on Polygon and OP mainnets, the 2.5 `USDT` should cover the chainlink fees (0.06 `$LINK` per req by our provider, and the gas fees). However, this fee is variable.

9LivesLabs: Fix approved

4.2 Medium Risk

4.2.1 Calling `executeUnlock()` before `fulfill()` will render all funds stuck

Severity: Medium

Context: `bridge.sol#L154-L169`

Description:

If a user calls `validateUnlock()` for a memId with intended amount to transfer 100 WETH, the oracle does the request side validation validation on chain and pass in the amount as result into the requests mapping.

```
    // Sends the request
    requestId = sendOperatorRequest(req, oracleFee);
    // map requestId to caller
    reqToCaller[requestId] = msg.sender;
    // map the chainlink requestId to memid
    reqToMemId[requestId] = _memid;
    // map the memid redeeming status to false
    midIsRedeemed[_memid] = false;
    return requestId;
```

After this a user can call the `executeUnlock()` for that memId and unlock their tokens. However, in the case that a user calls it before validating first, the transaction will still pass but the default value for the request will be 0 since chainlink has not assigned it an amount, it will be marked redeemed. It will revert even if the oracle tries to fullfill it after this call, hence locking in all user funds

Recommendation:

In `executeUnlock()` check that the amount returned from requests mapping is non zero.

MEM-Tech: Fixed

9LivesLabs: Fix Approved

4.2.2 Bridging funds is incompatible with smart contract wallets

Severity: Medium

Context: `bridge.sol#L175-L203`

Description:

When funds are to be bridged from EVM <> MEM, users call `lock()` which calculates the lock fee, transfers the tokens in and adjusts user & treasury balances accordingly:

```
    function lock(uint256 _amount) external {
        uint256 net_amount = computeNetAmount(_amount);
        uint256 generateFees = _amount - net_amount;
        // ERC20 token transfer
        token.safeTransferFrom(msg.sender, address(this), _amount);
        // update balances map
        balanceOf[msg.sender] += net_amount;
        // update treasury balance from fee cut
        balanceOf[treasury] += generateFees;
        // update totalLocked amount
        totalLocked += net_amount;
        //update treasury cumultive fee
        cumulativeFees += generateFees;
        // emit event
        emit Lock(msg.sender, net_amount);
    }
```

The issue is that if a user is using a smart contract wallet instead of an EOA, the address will be different on different networks. When the locked in amount is incremented as `balanceOf[msg.sender]`, but the user's address on the receiving side is different, this will lead to the funds being attributed to another address and potentially lost.

Recommendation:

Let the user pass an argument for receiving address `_to`.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.3 Low Risk

4.3.1 Implement treasury setter function in case of emergency

Severity: Low

Description:

Currently the contract is missing functionality to change the treasury address once set in the constructor. In case of emergency or compromised treasury EOA, the protocol will lose all accumulated and future fees and will likely need to be re-deployed. Consider adding such functionality to the contract.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.3.2 Restrict fee setting to MIN/MAX range

Severity: Low

Description:

There are currently no `MIN/MAX` ranges when setting fees in the protocol. Consider adding a `>MIN_FEE && <MAX_FEE` constants range when setting fees in the constructor and in `setFeeInJuels()`.

MEM-Tech: Acknowledged

9LivesLabs: Acknowledged

4.3.3 Subtract `amount` instead of `net_amount`

Severity: Low

Description:

Currently the wrong value is subtracted from the `totalLocked` variable when unlocking. Make sure to subtract `amount` instead of `net_amount`.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.3.4 Hardcoded MEM URL is not recommended

Severity: Low

Description:

The URL is currently hardcoded as following:

```
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
```

There is no way to set a new URL, so in case the domain is lost or changed, there is no way to adjust it to the new one. Consider adding a function to set a new one.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.4 JS Issues

4.4.1 Current address check is not compatible with all EVM compatible chains

Severity: JS Low

Context: `bridge.js#L93-L98`

Description:

Bridge.js validates the account address by checking it's format, but the problem is not all the EVM compatible chains have same address format that we see on ethereum, for example on tron which is EVM compatible chain an EOA address is like this:

```
TTUCJ3dKKikv3AhScBQ9aRsGeJ2EuBXQhd
```

So for such chain this check will fail:

```
  function _validateEoaSyntax(address) {
    ContractAssert(
      /^(0x)?[0-9a-fA-F]{40}$/.test(address),
      "ERROR_INVALID_EOA_ADDR",
    );
  }
```

MEM-Tech: Will work on chains intended for deployment (OP & Polygon)

9LivesLabs: Acknowledged

4.4.2 Max value a js integer can hold is 9007199254740991 which can lead to loss of funds

Severity: JS Low

Description:

In solidity largest type of integer is uint256 which can hold max value of `uint256.max(1157`
but in JS the max value for integer is:

```
2^53-1, or
+/- 9,007,199,254,740,991,
```

And adding any number to it won't overflow it to zero like old solidity versions but instead it remains same .

So consider the scnerio of token being bridged with total supply of uint256.max, and in many cases the top holder hold 15-30% of supply and in some cases more than that. For example in WETH case top holder hold almost 19% of total supply.

So for a token of uint256.max supply if the top holder hold the 1% of supply(1157920892373161954235709850086879078532699846656405640394575840079
and decides to bridge it all everything above 9,007,199,254,740,991 will be lost.

Point worth noting it is even feasible if supply is not uint256.max, it will also work for something like uint160.

Consider using bigInt.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.5 Info/Gas Optimizations

4.5.1 Redundant code can be removed

Severity: Info

Description:

Redundant code can be removed on L#164.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.5.2 No address(0) check on setter functions

Severity: Info

Description:

There are currently no `address(0)` checks when setting addresses in the constructor or when updating the oracle's address in `setOracleAddress()`. Consider implementing these checks.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.5.3 Use locked instead of unlocked pragma

Severity: Info

Description:

Currently, the pragma version is unlocked, it is recommended to lock the pragma to the latest compatible version since there are usually gas optimizations and bug fixes.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.5.4 There is no way to query MemID to RequestID

Severity: Info

Description:

In the contract there is a mapping for `reqToMemId` to get the memId for a request, consider adding a mapping for `memIdToRequestId`.

MEM-Tech: Fixed

9LivesLabs: Fix approved

4.5.5 Implement errors instead of revert string for gas savings

Severity: Gas

Description:

Reverting with a custom error instead of a string in require statements saves gas, it is recommended to change all instances of strings to errors.

All instances are on lines `L141`, `L144`, `L183`, `L185`, `L216`, `L228`.

MEM-Tech: Acknowledged

9LivesLabs: Acknowledged

4.5.6 Use solady string library for more gas optimized hex conversion

Severity: Gas

Description:

Solady provides a more gas-optimized string library than OpenZeppelin's whilst having the same `toHexString` function. Saves a lot of gas.

Solady String Library

MEM-Tech: Acknowledged

9LivesLabs: Acknowledged

4.5.7 Use solady safeTransferLib library to save gas

Severity: Gas

Description:

Solady also has a gas-optimised safeTransferLib library written in the assembly that can save a lot of gas for the user. Use that in your codebase for better UX.

Solady Safe Transfer Library

MEM-Tech: Acknowledged

9LivesLabs: Acknowledged