

Preliminary Security Assessment

AI competition

Apr 25, 2024

This smart contract audit report was created by Bunzz Audit. It utilized a database of over 1000 contract vulnerability patterns, comparing the project's contract against this database with AI to conduct a comprehensive diagnosis of vulnerabilities

Notes on Preliminary Security Assessment

This report presents the results of a vulnerability diagnosis of the code conducted non-biasedly, without considering the specific operations or circumstances of the project. Therefore, the report may highlight issues that your project has intentionally set. Please rest assured that these details will be filtered out in the Finalized Security Assessment.

Table of Contents

- Summary
- Coverage
- Findings
- Details

Summary

The smart contract audit revealed multiple security vulnerabilities that could potentially compromise the integrity, availability, and confidentiality of the contract's operations. Key issues include oracle manipulation, external dependency risks, reentrancy vulnerabilities, arithmetic overflows, unvalidated external service calls, insufficient access controls, and susceptibility to front-running and denial-of-service attacks. These vulnerabilities stem from inadequate validation mechanisms, improper handling of external calls, and lack of robust access control measures. Addressing these issues is crucial to ensure the security and reliability of the contract's functions.

Coverage

In this audit, we checked a total of 241 types of vulnerability patterns.



Title	Found
Front Running	ISSUE-4
Back Running	-
Sandwiching	-
Transaction order dependency	-
Fake tokens	-
Fake contracts	-
On-chain oracle manipulation	ISSUE-1
Governance attack	-
Token standard incompatibility	ISSUE-11
Flash liquidity borrow, purchase, mint, or deposit	-
Unsafe call to phantom function	-
One DeFi protocol dependency	ISSUE-2
Unfair slippage protection	-
Unfair liquidity providing	-
Unsafe or infinite token approval	-
Delegatecall injection	-
Unhandled or mishandled exception	ISSUE-18

Locked or frozen tokens	ISSUE-12
Absence of code logic or sanity check	ISSUE-19
Casting	ISSUE-20
Unbounded operation, including gas limit and call-stack depth	-
Arithmetic mistakes	ISSUE-5
Inconsistent access control	ISSUE-13
Visibility errors, including unrestricted action	ISSUE-21, ISSUE-22
Direct call to untrusted contract	ISSUE-6
Function Default Visibility	-
Integer Overflow and Underflow	-
Outdated Compiler Version	-
Floating Pragma	ISSUE-23
Unchecked Call Return Value	ISSUE-24
Unprotected Ether Withdrawal	ISSUE-7
Wrong Comparison Operator	-
UINT256 could be more gas efficient than smaller types	-
Magic numbers should be replaced with constants	ISSUE-25
Unprotected SELFDESTRUCT Instruction	-
Reentrancy	ISSUE-3
State Variable Default Visibility	ISSUE-26
Uninitialized Storage Pointer	-
Assert Violation	ISSUE-14
Use of Deprecated Solidity Functions	-
Delegatecall to Untrusted Callee	-
DoS with Failed Call	ISSUE-8
Transaction Order Dependence	ISSUE-9
Authorization through tx.origin	-
Block values as a proxy for time	-
Signature Malleability	-
Incorrect Constructor Name	-
Shadowing State Variables	-
Weak Sources of Randomness from Chain Attributes	-
Missing Protection against Signature Replay Attacks	-
Lack of Proper Signature Verification	-
Requirement Violation	ISSUE-27
Write to Arbitrary Storage Location	-
Incorrect Inheritance Order	-
Insufficient Gas Griefing	ISSUE-15
Arbitrary Jump with Function Type Variable	-
DoS With Block Gas Limit	-
Typographical Error	-
Right-To-Left-Override control character (U+202E)	-

Presence of unused variables	-
Unexpected Ether balance	-
Hash Collisions With Multiple Variable Length Arguments	-
Message call with hardcoded gas amount	-
Code With No Effects	ISSUE-16
Unencrypted Private Data On-Chain	ISSUE-17
Constant variables should be marked as private	-
Reading array length in for-loops	-
Checked arithmetic for for-loops	-
++i costs less gas than i++	-
IERC20.transfer does not support all ERC20 tokens	ISSUE-28, ISSUE-29, ISSUE-30
Unrestricted minting	-
Trust Issue Of Admin Roles	ISSUE-10
Replace `abi.encodeWithSignature` and `abi.encodeWithSelector` with `abi.encodeCall` which keeps the code typo/type safe	-
abicodec v2 is enabled by default	-
Missing checks for `address(0)` when assigning values to address state variables	L-2, NC-1
Array indices should be referenced via `enum`s rather than via numeric literals	-
`require()` should be used instead of `assert()`	NC-2
Use `string.concat()` or `bytes.concat()` instead of `abi.encodePacked`	NC-3
Constants should be in CONSTANT_CASE	-
`constant`s should be defined rather than using magic numbers	NC-4
Control structures do not follow the Solidity Style Guide	-
Critical Changes Should Use Two-step Procedure	-
Dangerous `while(true)` loop	-
Default Visibility for constants	-
Delete rogue `console.log` imports	-
Consider disabling `renounceOwnership()`	-
Draft Dependencies	-
Duplicated `require()` / `revert()` Checks Should Be Refactored To A Modifier Or Function	NC-5
`else`-block not required	-
Unused `error` definition	-
Event is never emitted	-
Events should use parameters to convey information	-
Event missing indexed field	NC-6
Events that mark critical parameter changes should contain both the old and the new value	-
Function ordering does not follow the Solidity style guide	NC-7
Functions should not be longer than 50 lines	NC-8
Change int to int256	NC-9
Change uint to uint256	-
Interfaces should be indicated with an `I` prefix in the contract name	-

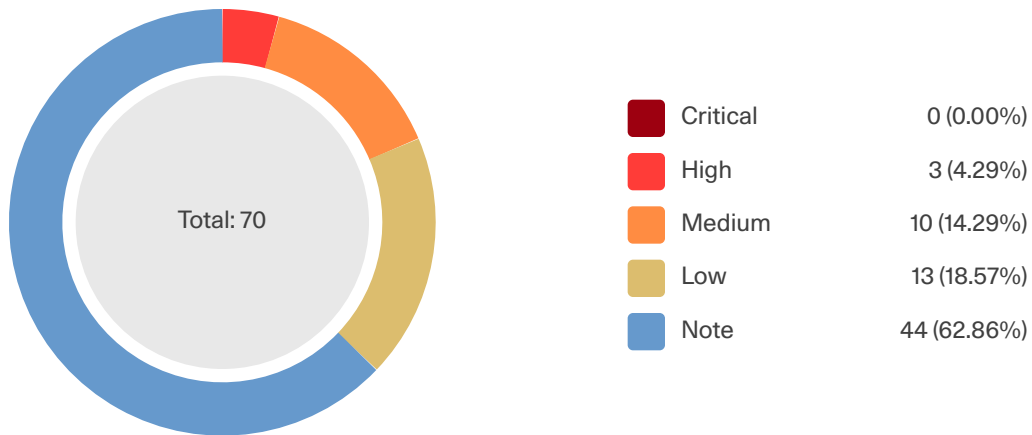
Interfaces should be defined in separate files from their usage	-
Lack of checks in setters	NC-10
Lines are too long	-
`mapping` definitions do not follow the Solidity Style Guide	-
`type(uint<n>).max` should be used instead of `uint<n>(-1)`	-
`type(uint256).max` should be used instead of `2 ** 256 - 1`	-
Missing Event for critical parameters change	NC-11
NatSpec is completely non-existent on functions that should have them	-
Incomplete NatSpec: `@param` is missing on actually documented functions	-
Incomplete NatSpec: `@return` is missing on actually documented functions	NC-12
File's first line is not an SPDX Identifier	-
Use a `modifier` instead of a `require/if` statement for a special `msg.sender` actor	NC-13
Constant state variables defined more than once	-
Consider using named mappings	NC-14
`address`s shouldn't be hard-coded	-
The `nonReentrant` `modifier` should occur before all other modifiers	-
Numeric values having to do with time should use time units for readability	-
Variable names that consist of all capital letters should be reserved for `constant` / `immutable` variables	-
Owner can renounce while system is paused	-
Adding a `return` statement when the function defines a named return variable, is redundant	NC-15
`require()` / `revert()` statements should have descriptive reason strings	-
Take advantage of Custom Error's return value property	-
Deprecated library used for Solidity `>= 0.8` : SafeMath	-
Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)	-
Use scientific notation for readability reasons for large multiples of ten	-
Avoid the use of sensitive terms	-
Strings should use double quotes rather than single quotes	NC-16
Function writing that does not comply with the Solidity Style Guide	-
Contract does not follow the Solidity style guide's suggested layout ordering	NC-17
TODO Left in the code	-
Some require descriptions are not clear	-
Use Underscores for Number Literals (add an underscore every 3 digits)	NC-18
Internal and private variables and functions names should begin with an underscore	NC-19
Event is missing `indexed` fields	NC-20
Constants should be defined rather than using magic numbers	NC-21
`override` function arguments that are unused should have the variable name removed or commented out to avoid compiler warnings	-
`public` functions not called by the contract should be declared `external` instead	NC-22
Variables need not be initialized to zero	-
No need to check that `v == 27` or `v == 28` with `erecover`	-
Contracts are vulnerable to fee-on-transfer accounting-related issues	M-1
NFT contract redefines `_mint()` / `_safeMint()`, but not both	-

<code>`approve()`,`safeApprove()``</code> may revert if the current approval is not zero	-
Use of <code>`tx.origin``</code> is unsafe in almost every context	-
<code>`block.number``</code> means different things on different L2s	-
Centralization Risk for trusted owners	M-2
Use of deprecated chainlink function: <code>`latestAnswer()``</code>	-
<code>`call()``</code> should be used instead of <code>`transfer()``</code> on an <code>`address payable``</code>	-
<code>`_safeMint()``</code> should be used rather than <code>`_mint()``</code> wherever possible	-
Using <code>`transferFrom``</code> on ERC721 tokens	-
Fees can be set to be greater than 100%.	M-3
<code>`increaseAllowance/decreaseAllowance``</code> won't work on mainnet for USDT	-
Lack of EIP-712 compliance: using <code>`keccak256()``</code> directly on an array or struct variable	-
Library function isn't <code>`internal``</code> or <code>`private``</code>	-
Solady's SafeTransferLib does not check for token contract's existence	-
Solmate's SafeTransferLib does not check for token contract's existence	-
Chainlink's <code>`latestRoundData``</code> might return stale or incorrect results	-
Missing checks for whether the L2 Sequencer is active	-
Direct <code>`supportsInterface()``</code> calls may cause caller to revert	-
Return values of <code>`transfer()`,`transferFrom()``</code> not checked	-
Unsafe use of <code>`transfer()`,`transferFrom()``</code> with <code>`IERC20``</code>	-
Use a 2-step ownership transfer pattern	-
Precision Loss due to Division before Multiplication	-
NFT doesn't handle hard forks	-
Some tokens may revert when zero value transfers are made	L-1
Missing checks for <code>`address(0)``</code> when assigning values to address state variables	L-2, NC-1
Use of <code>`ecrecover``</code> is susceptible to signature malleability	-
<code>`abi.encodePacked()``</code> should not be used with dynamic types when passing the result to a hash function such as <code>`keccak256()``</code>	-
Use of <code>`tx.origin``</code> is unsafe in almost every context	-
<code>`calc_token_amount()``</code> has slippage added on top of Curve's calculated slippage	-
<code>`decimals()``</code> is not a part of the ERC-20 standard	-
<code>`decimals()``</code> should be of type <code>`uint8``</code>	-
Deprecated <code>approve()``</code> function	-
Do not use deprecated library functions	-
<code>`safeApprove()``</code> is deprecated	-
Deprecated <code>_setupRole()``</code> function	-
Do not leave an implementation contract uninitialized	-
Division by zero not prevented	-
<code>`domainSeparator()``</code> isn't protected against replay attacks in case of a future chain split	-
Duplicate import statements	-
Empty Function Body - Consider commenting why	-
Empty <code>`receive()/payable fallback()``</code> function does not authenticate requests	-
External calls in an un-bounded <code>`for-`` loop may result in a DOS</code>	-

External call recipient may consume all transaction gas	-
Fallback lacking `payable`	-
Initializers could be front-run	-
Signature use at deadlines should be allowed	-
Lack of Slippage check	-
`Math.max(<x>,0)` used with `int` cast to `uint`	-
Prevent accidentally burning tokens	-
NFT ownership doesn't support hard forks	-
Owner can renounce while system is paused	-
Possible rounding issue	-
`pragma experimental ABIEncoderV2` is deprecated	-
Loss of precision	L-3
Solidity version 0.8.20+ may not work on other chains due to `PUSH0`	L-4
Use `Ownable2Step.transferOwnership` instead of `Ownable.transferOwnership`	-
File allows a version of solidity that is susceptible to an assembly optimizer bug	L-5
Sweeping may break accounting if tokens with multiple addresses are used	-
`symbol()` is not a part of the ERC-20 standard	-
Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows when downcasting	-
Unsafe ERC20 operation(s)	L-6
Unsafe solidity low-level call can cause gas grief attack	-
Unspecific compiler version pragma	-
Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions	-
Upgradeable contract not initialized	-
Use of ecrecover is susceptible to signature malleability	-
Use `initializer` for public-facing functions only. Replace with `onlyInitializing` on internal functions.	-
A year is not always 365 days	-
Incorrect comparison implementation	-
Using `delegatecall` inside a loop	-
`get_dy_underlying()` is not a flash-loan-resistant price	-
Using `msg.value` in a loop	-
`wstETH`'s functions operate on units of stEth, not Eth	-
Use ERC721A instead ERC721	-
Don't use `_msgSender()` if not supporting EIP-2771	-
`a = a + b` is more gas effective than `a += b` for state variables (excluding arrays and mappings)	GAS-1
Use assembly to check for `address(0)`	-
`array[index] += amount` is cheaper than `array[index] = array[index] + amount` (or related variants)	-
Comparing to a Boolean constant	-
Using bools for storage incurs overhead	GAS-2
Cache array length outside of loop	-
State variables should be cached in stack variables rather than re-reading them from storage	-

Use calldata instead of memory for function arguments that do not get mutated	GAS-3
For Operations that will not overflow, you could use unchecked	GAS-4
Use Custom Errors instead of Revert Strings to save Gas	GAS-5
Avoid contract existence checks by using low level calls	GAS-6
Stack variable used as a cheaper cache for a state variable is only used once	-
State variables only set in the constructor should be declared <code>immutable</code>	GAS-7
Don't initialize variables with default value	-
Reduce the size of error messages (Long revert Strings)	-
Functions guaranteed to revert when called by normal users can be marked <code>payable</code>	GAS-8
<code>++i</code> costs less gas compared to <code>i++</code> or <code>i += 1</code> (same for <code>--i</code> vs <code>i--</code> or <code>i -= 1</code>)	-
Using <code>private</code> rather than <code>public</code> for constants, saves gas	-
Use shift right/left instead of division/multiplication if possible	-
Splitting <code>require()</code> statements that use <code>&&</code> saves gas	-
Superfluous event fields	-
Use of <code>this</code> instead of marking as <code>public</code> an <code>external</code> function	-
<code>uint256</code> to <code>bool</code> mapping: Utilizing Bitmaps to dramatically save on Gas	-
Increments/decrements can be unchecked in for-loops	-
Use <code>!= 0</code> instead of <code>> 0</code> for unsigned integer comparison	GAS-9
<code>internal</code> functions not called by the contract should be removed	-
WETH address definition can be use directly	-








Findings



ID	Title	Severity	Status
ISSUE-1	Oracle Manipulation Vulnerability in validateUnlock Function	High	-
ISSUE-2	External Dependency in Token Unlocking	High	-
ISSUE-3	Reentrancy Vulnerability in executeUnlock Function	High	-
ISSUE-4	Front Running Vulnerability in validateUnlock Function	Medium	-
ISSUE-5	Arithmetic Overflow Vulnerability in Lock Function	Medium	-
ISSUE-6	Unvalidated External Service Call in validateUnlock Function	Medium	-
ISSUE-7	Insufficient Access Control in withdrawFees Function	Medium	-
ISSUE-8	DoS Vulnerability Due to External Call Failure	Medium	-
ISSUE-9	Transaction Order Dependence in validateUnlock Function	Medium	-
ISSUE-10	Privileged Account Control Over Critical Functions	Medium	-
ISSUE-11	Token Standard Incompatibility Risk	Low	-
ISSUE-12	Premature Token Redemption Vulnerability	Low	-

ISSUE-13	Inadequate Access Control and Assert Misuse in withdrawFees Function	Low	-
ISSUE-14	Improper Use of Assert for External State Check	Low	-
ISSUE-15	Insufficient Gas Griefing in validateUnlock Function	Low	-
ISSUE-16	Premature State Change in MemBridge Contract	Low	-
ISSUE-17	Public Exposure of Sensitive MEM IDs	Low	-
ISSUE-18	Improper Exception Handling in withdrawFees Function	Note	-
ISSUE-19	Missing Input Validation in validateUnlock Function	Note	-
ISSUE-20	Type Casting Vulnerability in setJobId Function	Note	-
ISSUE-21	Inefficient Function Visibility	Note	-
ISSUE-22	Inefficient Gas Usage Due to Public Visibility	Note	-
ISSUE-23	Floating Pragma Vulnerability	Note	-
ISSUE-24	Proper Handling of Transfer Return Value	Note	-
ISSUE-25	Use of Magic Numbers in Fee Calculation	Note	-
ISSUE-26	Proper Visibility Declaration of State Variables	Note	-
ISSUE-27	Restrictive Zero Amount Handling in fulfill Function	Note	-
ISSUE-28	Proper Use of SafeERC20 for Token Transfers	Note	-
ISSUE-29	Proper Use of SafeERC20 for Non-Boolean Token Transfers	Note	-
ISSUE-30	Proper Use of SafeERC20 for Token Transfers	Note	-
M-1	Contracts are vulnerable to fee-on-transfer accounting-related issues	Medium	-
M-2	Centralization Risk for trusted owners	Medium	-
M-3	Fees can be set to be greater than 100%.	Medium	-
L-1	Some tokens may revert when zero value transfers are made	Low	-
L-2	Missing checks for address(0) when assigning values to address state variables	Low	-
L-3	Loss of precision	Low	-
L-4	Solidity version 0.8.20+ may not work on other chains due to PUSH0	Low	-
L-5	File allows a version of solidity that is susceptible to an assembly optimizer bug	Low	-
L-6	Unsafe ERC20 operation(s)	Low	-

GAS-1	a = a + b is more gas effective than a += b for state variables (excluding arrays and mappings)	● Note	-
GAS-2	Using bools for storage incurs overhead	● Note	-
GAS-3	Use calldata instead of memory for function arguments that do not get mutated	● Note	-
GAS-4	For Operations that will not overflow, you could use unchecked	● Note	-
GAS-5	Use Custom Errors instead of Revert Strings to save Gas	● Note	-
GAS-6	Avoid contract existence checks by using low level calls	● Note	-
GAS-7	State variables only set in the constructor should be declared immutable	● Note	-
GAS-8	Functions guaranteed to revert when called by normal users can be marked payable	● Note	-
GAS-9	Use != 0 instead of > 0 for unsigned integer comparison	● Note	-
NC-1	Missing checks for address(0) when assigning values to address state variables	● Note	-
NC-2	require() should be used instead of assert()	● Note	-
NC-3	Use string.concat() or bytes.concat() instead of abi.encodePacked	● Note	-
NC-4	constants should be defined rather than using magic numbers	● Note	-
NC-5	Duplicated require()/revert() Checks Should Be Refactored To A Modifier Or Function	● Note	-
NC-6	Event missing indexed field	● Note	-
NC-7	Function ordering does not follow the Solidity style guide	● Note	-
NC-8	Functions should not be longer than 50 lines	● Note	-
NC-9	Change int to int256	● Note	-
NC-10	Lack of checks in setters	● Note	-
NC-11	Missing Event for critical parameters change	● Note	-
NC-12	Incomplete NatSpec: @return is missing on actually documented functions	● Note	-
NC-13	Use a modifier instead of a require/if statement for a special msg.sender actor	● Note	-
NC-14	Consider using named mappings	● Note	-
NC-15	Adding a return statement when the function defines a named return variable, is redundant	● Note	-

NC-16	Strings should use double quotes rather than single quotes	 Note	-
NC-17	Contract does not follow the Solidity style guide's suggested layout ordering	 Note	-
NC-18	Use Underscores for Number Literals (add an underscore every 3 digits)	 Note	-
NC-19	Internal and private variables and functions names should begin with an underscore	 Note	-
NC-20	Event is missing indexed fields	 Note	-
NC-21	Constants should be defined rather than using magic numbers	 Note	-
NC-22	public functions not called by the contract should be declared external instead	 Note	-

Details

ISSUE-1: Oracle Manipulation Vulnerability in validateUnlock Function

Severity

High

Location

contracts/MemBridge.sol:MemBridge:validateUnlock

Description

The function `validateUnlock` constructs a URL to fetch data from an external source without any validation on the source or the data it returns, which can lead to oracle manipulation. The function does not implement any checks to validate the authenticity and integrity of the data received from the oracle, potentially leading to the contract acting on false information.

How to fix

Implement checks to validate the authenticity and integrity of the data received from the oracle. This could include using signatures, requiring multiple confirmations from different sources, or other similar mechanisms to ensure data reliability.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    // Add validation logic here
    if (!isValidData(req)) {
        revert("Invalid data received from oracle");
    }
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Status

N/A

Comment

N/A

ISSUE-2: External Dependency in Token Unlocking

Severity

High

Location

contracts/MemBridge.sol:MemBridge.validateUnlock

Description

The contract's critical functionality, such as token unlocking, relies on data from an external Chainlink oracle, which in turn fetches data from an external API (0xmem.net). If either of these external dependencies is compromised, it could lead to incorrect or malicious data being used to unlock tokens, potentially causing financial losses.

How to fix

Implement additional checks and balances around the data received from the external oracle. Consider using multiple independent oracles to cross-verify data before processing unlocks. Implement circuit breakers that can halt operations based on anomaly detection in oracle data.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    require(!midIsRedeemed[_memid], "MEM ID already redeemed");
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    // Additional oracle addresses for cross-verification
    address[] memory oracleAddresses = new address[](3);
    oracleAddresses[0] = oracleAddress; // existing oracle
    oracleAddresses[1] = address(0xAnotherOracle); // additional oracle
    oracleAddresses[2] = address(0xAnotherOracle2); // additional oracle
    // Cross-verify data from all oracles
    for (uint i = 0; i < oracleAddresses.length; i++) {
        _setChainlinkOracle(oracleAddresses[i]);
        string memory url = string.concat("https://0xmem.net/vu/", _memid);
        req._add("method", "GET");
        req._add("url", url);
        req._add("path", "amount");
        bytes32 tempRequestId = _sendOperatorRequest(req, oracleFee);
        // Logic to cross-verify data from all oracles
    }
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Status

N/A

Comment

N/A

ISSUE-3: Reentrancy Vulnerability in executeUnlock Function

Severity

High

Location

contracts/MemBridge.sol:MemBridge:executeUnlock

Description

The function `executeUnlock` in `MemBridge.sol` is vulnerable to reentrancy attacks as it makes an external call to `token.safeTransfer` before finalizing all state changes, such as updating the caller's balance and the treasury balance. This order of operations can allow reentrancy attacks where the state can be manipulated by a recursive call to the function.

How to fix

To prevent reentrancy, ensure that all state changes happen before calling external contracts. Use the Checks-Effects-Interactions pattern and consider using reentrancy guards like OpenZeppelin's `ReentrancyGuard`.

Code

```
function executeUnlock(bytes32 _requestId) public {
    // retrieve request amount and mem id from maps
    uint256 amount = requests[_requestId];
    string memory memid = reqToMemId[_requestId];
    // fee calculation
    uint256 net_amount = computeNetAmount(amount);
    uint256 generateFees = amount - net_amount;
    // validate that the request owner is the function caller
    require(reqToCaller[_requestId] == msg.sender, "err_invalid_caller");
    // do balances checks
    require(
        balanceOf[msg.sender] >= amount && balanceOf[msg.sender] > 0,
        "Insufficient funds"
    );
    // seal this memid and make its reusage not possible
    midIsRedeemed[memid] = true;
    // update the caller balance
    balanceOf[msg.sender] -= amount;
    // update the treasury balance
    balanceOf[treasury] += generateFees;
    // update stats: cumulative fees
    cumulativeFees += generateFees;
    // update stats: total locked tokens
    totalLocked -= net_amount;
    //transfer the tokens
    token.safeTransfer(msg.sender, net_amount);
    // emit event
    emit Unlock(msg.sender, net_amount);
}
```


Code Suggestion

```
function executeUnlock(bytes32 _requestId) public {
    // retrieve request amount and mem id from maps
    uint256 amount = requests[_requestId];
    string memory memid = reqToMemId[_requestId];
    // fee calculation
    uint256 net_amount = computeNetAmount(amount);
    uint256 generateFees = amount - net_amount;
    // validate that the request owner is the function caller
    require(reqToCaller[_requestId] == msg.sender, "err_invalid_caller");
    // do balances checks
    require(
        balanceOf[msg.sender] >= amount && balanceOf[msg.sender] > 0,
        "Insufficient funds"
    );
    // update the caller balance
    balanceOf[msg.sender] -= amount;
    // update the treasury balance
    balanceOf[treasury] += generateFees;
    // update stats: cumulative fees
    cumulativeFees += generateFees;
    // update stats: total locked tokens
    totalLocked -= net_amount;
    // seal this memid and make its reuse not possible
    midIsRedeemed[memid] = true;
    //transfer the tokens
    token.safeTransfer(msg.sender, net_amount);
    // emit event
    emit Unlock(msg.sender, net_amount);
}
```

Status

N/A

Comment

N/A

ISSUE-4: Front Running Vulnerability in validateUnlock Function

Severity

Medium

Location

contracts/MemBridge.sol:MemBridge:validateUnlock

Description

The validateUnlock function is vulnerable to front running as it involves a transaction that can be seen and potentially exploited by an attacker. This allows an attacker to manipulate or anticipate the state change before the original transaction is confirmed.

How to fix

To mitigate front running, consider using techniques such as commit-reveal schemes, or adding randomness to transaction ordering, or using a more private method for transaction submission like private transactions if supported by the blockchain.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    bytes32 commit = keccak256(abi.encodePacked(_memid, msg.sender,
block.timestamp));
    require(!commits[commit], "Commit already used");
    commits[commit] = true;
    // Rest of the function remains the same
    // Add reveal function that checks the commit
}
```

Status

N/A

Comment

N/A

ISSUE-5: Arithmetic Overflow Vulnerability in Lock Function

Severity

Medium

Location

contracts/MemBridge.sol:MemBridge:lock

Description

The lock function performs arithmetic operations without safe math checks, which could lead to overflows if the input values are large enough.

How to fix

Use SafeMath library or Solidity 0.8 built-in overflow checks for all arithmetic operations to prevent overflows.

Code

```
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 cumulative fee
    cumulativeFees += generateFees;
    // emit event
    emit Lock(msg.sender, net_amount);
}
```

Code Suggestion

```
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] = balanceOf[msg.sender].add(net_amount);
    // update treasury balance from fee cut
    balanceOf[treasury] = balanceOf[treasury].add(generateFees);
    // update totalLocked amount
    totalLocked = totalLocked.add(net_amount);
    //update treasury cumulative fee
    cumulativeFees = cumulativeFees.add(generateFees);
    // emit event
    emit Lock(msg.sender, net_amount);
}
```

Status

N/A

Comment

N/A

ISSUE-6: Unvalidated External Service Call in validateUnlock Function

Severity

Medium

Location

contracts/MemBridge.sol:MemBridge.validateUnlock

Description

The function 'validateUnlock' constructs a URL dynamically using external input (_memid and msg.sender) and makes an HTTP GET request without validating the URL against a list of trusted services, potentially allowing interactions with malicious or unintended external endpoints.

How to fix

Implement a whitelist of trusted external services and validate the URL against this list before making the call. Additionally, consider implementing circuit breakers or other security mechanisms to handle unexpected or malicious data returned from external calls.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    require(trustedServices.contains("https://0xmem.net"), "Untrusted external
service");
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Status

N/A

Comment

N/A

ISSUE-7: Insufficient Access Control in withdrawFees Function

Severity

Medium

Location

contracts/MemBridge.sol:MemBridge:withdrawFees

Description

The function 'withdrawFees' is vulnerable due to its reliance solely on the treasury address for authorization, which poses a risk if the treasury address is compromised, potentially leading to unauthorized asset withdrawals.

How to fix

Implement more robust access controls and possibly multi-signature requirements for critical functions such as withdrawals to enhance security.

Code

```
function withdrawFees() public {
    uint256 amount = balanceOf[treasury];
    assert(amount > 0);
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Code Suggestion

```
function withdrawFees() public onlyOwner {
    uint256 amount = balanceOf[treasury];
    require(amount > 0, "No fees to withdraw");
    require(msg.sender == treasury || isAuthorized(msg.sender), "Unauthorized");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}

function isAuthorized(address _addr) private view returns(bool) {
    return authorizedAddresses[_addr];
}
```

Status

N/A

Comment

N/A

ISSUE-8: DoS Vulnerability Due to External Call Failure

Severity

Medium

Location

contracts/MemBridge.sol:MemBridge:validateUnlock

Description

The function `validateUnlock` makes an external call within a transaction that also changes the state. If the external call fails, it could revert the transaction, leading to a DoS condition.

How to fix

Implement a pull mechanism where the state change is separated from the external call. Allow the transaction to complete even if the external call fails, and handle failures explicitly without reverting the entire transaction.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    require(!midIsRedeemed[_memid], "MEM ID already redeemed");
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    try {
        requestId = _sendOperatorRequest(req, oracleFee);
    } catch {
        // Handle failure
    }
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Status

N/A

Comment

N/A

ISSUE-9: Transaction Order Dependence in validateUnlock Function

Severity

Medium

Location

contracts/MemBridge.sol:MemBridge:validateUnlock

Description

The validateUnlock function is vulnerable to front-running because it uses `_memid`, which can be seen in the mempool and exploited by sending a transaction with a higher gas fee to redeem it first.

How to fix

Implement a commit-reveal scheme to prevent front-running. This involves splitting the transaction into two phases: a commit phase where the user submits a hash of their intended action, and a reveal phase where the user reveals their action after some time has passed.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memidHash) public returns (bytes32
requestId) {
    require(!commitments[_memidHash].revealed, "Already revealed");
    commitments[_memidHash] = Commitment({
        revealed: false,
        memid: "",
        sender: msg.sender
    });
    return keccak256(abi.encodePacked(_memidHash, msg.sender));
}

function revealUnlock(string calldata _memid, string calldata _secret) public {
    bytes32 hash = keccak256(abi.encodePacked(_memid, _secret));
    require(commitments[hash].sender == msg.sender, "Invalid sender");
    require(!commitments[hash].revealed, "Already revealed");
    commitments[hash].revealed = true;
    commitments[hash].memid = _memid;
    // proceed with unlock
}
```

Status

N/A

Comment

N/A

ISSUE-10: Privileged Account Control Over Critical Functions

Severity

Medium

Location

contracts/MemBridge.sol:MemBridge:withdrawLink, contracts/MemBridge.sol:MemBridge:withdrawFees

Description

The functions 'withdrawLink' and 'withdrawFees' allow single privileged accounts (owner and treasury) to execute significant financial transactions without additional oversight or consensus, increasing the risk of misuse or abuse.

How to fix

Implement multi-signature mechanisms or decentralized governance for critical financial operations to ensure checks and balances.

Code

```
function withdrawLink() public onlyOwner {
    LinkTokenInterface link = LinkTokenInterface(_chainlinkTokenAddress());
    require(
        link.transfer(msg.sender, link.balanceOf(address(this))),
        "Unable to transfer"
    );
}
function withdrawFees() public {
    uint256 amount = balanceOf[treasury];
    assert(amount > 0);
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Code Suggestion

```
function withdrawLink() public onlyMultiSigOwners {
    LinkTokenInterface link = LinkTokenInterface(_chainlinkTokenAddress());
    require(
        link.transfer(msg.sender, link.balanceOf(address(this))),
        "Unable to transfer"
    );
}
function withdrawFees() public onlyMultiSigOwners {
    uint256 amount = balanceOf[treasury];
    assert(amount > 0);
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Status

N/A

Comment

N/A

ISSUE-11: Token Standard Incompatibility Risk

Severity

Low

Location

MemBridge.sol:MemBridge:lock

Description

The vulnerability arises from potential incompatibilities in token standards which could lead to unexpected behaviors during token transfers, approvals, or balance updates. Despite using SafeERC20, the complexity of cross-chain functionalities and multiple token operations in the contract could lead to reentrancy or unexpected token loss.

How to fix

Ensure that all token interactions are compliant with their respective standards. Conduct thorough testing and potentially engage in formal verification. Review and audit all parts of the code that interact with different token standards, especially those that handle token transfers, to ensure they adhere to expected behaviors.

Code

```
contract MemBridge is ChainlinkClient, ConfirmedOwner {
    using SafeERC20 for IERC20;
    using Chainlink for Chainlink.Request;
    IERC20 public immutable token;
    ...
    function lock(uint256 _amount) external {
        uint256 net_amount = computeNetAmount(_amount);
        uint256 generateFees = _amount - net_amount;
        token.safeTransferFrom(msg.sender, address(this), _amount);
        balanceOf[msg.sender] += net_amount;
        balanceOf[treasury] += generateFees;
        totalLocked += net_amount;
        cumulativeFees += generateFees;
        emit Lock(msg.sender, net_amount);
    }
    ...
}
```

Code Suggestion

Consider implementing checks or safeguards that verify and enforce standard compliance for all token operations. For example:

```
function ensureStandardCompliance(address token) private view returns(bool) {  
    // Example check for ERC20 compliance  
    return IERC20(token).totalSupply() > 0;  
}
```

```
// Use this function to verify compliance in critical operations  
require(ensureStandardCompliance(address(token)), "Token does not comply with  
expected standard");
```

Status

N/A

Comment

N/A

ISSUE-12: Premature Token Redemption Vulnerability

Severity

Low

Location

contracts/MemBridge.sol:MemBridge:validateUnlock

Description

The function 'validateUnlock' sets the 'midlsRedeemed[_memid]' flag to true immediately after sending a Chainlink request, before the oracle response is validated. This premature setting can lead to tokens being marked as redeemed without proper validation, potentially locking them.

How to fix

Modify the code to delay setting the 'midlsRedeemed' status to true until after the Chainlink oracle has successfully validated and fulfilled the unlock request.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    // Delay setting the redeemed status
    // midIsRedeemed[_memid] = true;
    return requestId;
}

function fulfill(bytes32 _requestId, uint256 _result) public
recordChainlinkFulfillment(_requestId) returns (uint256) {
    require(_result > 0, "err_zero_amount");
    string memory memid = reqToMemId[_requestId];
    require(!midIsRedeemed[memid], "err_mid_redeemed");
    requests[_requestId] = _result;
    midIsRedeemed[memid] = true;
    emit Request(_requestId, _result);
    return _result;
}
```

Status

N/A

Comment

N/A

ISSUE-13: Inadequate Access Control and Assert Misuse in withdrawFees Function

Severity

Low

Location

contracts/MemBridge.sol:MemBridge:withdrawFees

Description

The function 'withdrawFees' uses a simple equality check against 'msg.sender' to restrict access, which is vulnerable if the treasury's private key is compromised. Additionally, the use of 'assert' for balance checking is inappropriate for input validation as it consumes all gas if it fails without providing error messages.

How to fix

Replace 'assert' with 'require' for balance checking and implement a more robust access control mechanism using modifiers.

Code

```
function withdrawFees() public {
    uint256 amount = balanceOf[treasury];
    assert(amount > 0);
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```


Code Suggestion

```
function withdrawFees() public onlyTreasury {
    uint256 amount = balanceOf[treasury];
    require(amount > 0, "No funds to withdraw");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}

modifier onlyTreasury() {
    require(msg.sender == treasury, "err_invalid_caller");
    _;
}
```

Status

N/A

Comment

N/A

ISSUE-14: Improper Use of Assert for External State Check

Severity

Low

Location

contracts/MemBridge.sol:MemBridge:withdrawFees

Description

The misuse of `assert()` for checking the treasury's non-zero balance is vulnerable because `assert()` should be used for invariants. The treasury balance can change due to external interactions, making it inappropriate for an invariant assertion.

How to fix

Replace `assert()` with `require()` to properly handle conditions that can change and are not contract invariants.

Code

```
function withdrawFees() public {
    uint256 amount = balanceOf[treasury];
    assert(amount > 0);
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Code Suggestion

```
function withdrawFees() public {
    uint256 amount = balanceOf[treasury];
    require(amount > 0, "No funds to withdraw");
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Status

N/A

Comment

N/A

ISSUE-15: Insufficient Gas Griefing in validateUnlock Function

Severity

Low

Location

contracts/MemBridge.sol:MemBridge.validateUnlock

Description

The validateUnlock function in MemBridge.sol makes a sub-call to a Chainlink oracle without ensuring sufficient gas, potentially causing the sub-call to fail and affecting the transaction's integrity.

How to fix

Add a check for sufficient gas before making the sub-call and handle the sub-call's failure gracefully.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    require(gasleft() > requiredGas, "Insufficient gas for the operation");
    assert(!midIsRedeemed[_memid]);
    Chainlink.Request memory req = _buildOperatorRequest(jobId,
this.fulfill.selector);
    string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
    string memory caller = string.concat("/",
Strings.toHexString(uint256(uint160(msg.sender)), 20));
    string memory url = string.concat(arg1, caller);
    req._add("method", "GET");
    req._add("url", url);
    req._add("path", "amount");
    req._add("headers", '["content-type", "application/json", "set-cookie",
"sid=14A52"]');
    req._add("body", "");
    req._add("contact", "https://t.me/decentland");
    req._addInt("multiplier", 1);
    requestId = _sendOperatorRequest(req, oracleFee);
    reqToCaller[requestId] = msg.sender;
    reqToMemId[requestId] = _memid;
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Status

N/A

Comment

N/A

ISSUE-16: Premature State Change in MemBridge Contract

Severity

Low

Location

contracts/MemBridge.sol:MemBridge:validateUnlock

Description

The vulnerability arises from setting 'midIsRedeemed[_memid]' to true before the Chainlink request is fulfilled, potentially leading to inconsistencies if the request fails or is not processed. This premature action could allow a memid to be marked as redeemed erroneously.

How to fix

Move the line 'midIsRedeemed[_memid] = true;' to the 'fulfill' function after the Chainlink request is successfully fulfilled to ensure that the memid is only marked as redeemed upon successful completion of the request.

Code

```
midIsRedeemed[_memid] = true;
```

Code Suggestion

```
function fulfill(bytes32 _requestId, uint256 _result) public
recordChainlinkFulfillment(_requestId) returns (uint256) {
    string memory memid;
    require(_result > 0, "err_zero_amount");
    memid = reqToMemId[_requestId];
    require(!midIsRedeemed[memid], "err_mid_redeemed");
    requests[_requestId] = _result;
    midIsRedeemed[memid] = true;
    emit Request(_requestId, _result);
    return _result;
}
```

Status

N/A

Comment

N/A

ISSUE-17: Public Exposure of Sensitive MEM IDs

Severity

Low

Location

contracts/MemBridge.sol:MemBridge:midIsRedeemed

Description

The vulnerability involves storing sensitive MEM IDs in a public mapping without encryption, making them accessible to anyone. This exposure could lead to unauthorized access and potential misuse of this data.

How to fix

Encrypt the MEM IDs or use a commitment scheme to obscure the actual values until they are needed for processing within the contract.

Code

```
mapping(string => bool) public midIsRedeemed;
```

Code Suggestion

```
Consider changing the visibility of 'midIsRedeemed' to private and providing a controlled access method if external access is necessary. Alternatively, implement a cryptographic commitment scheme to hide the actual values.
```

Status

N/A

Comment

N/A

ISSUE-18: Improper Exception Handling in withdrawFees Function

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:withdrawFees

Description

The use of 'assert' for checking 'amount > 0' is inappropriate for business logic validation, leading to potential gas wastage and poor error handling. 'Require' should be used instead for conditions that can fail under normal operation.

How to fix

Replace 'assert' with 'require' to check if the amount is greater than 0, providing a clear error message for better user experience and efficient gas usage.

Code

```
function withdrawFees() public {
    uint256 amount = balanceOf[treasury];
    assert(amount > 0);
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Code Suggestion

```
function withdrawFees() public {
    uint256 amount = balanceOf[treasury];
    require(amount > 0, "No funds to withdraw");
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Status

N/A

Comment

N/A

ISSUE-19: Missing Input Validation in validateUnlock Function

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:validateUnlock

Description

The function lacks input validation for the `_memid` parameter, which could lead to processing invalid or unexpected inputs, potentially causing errors or unexpected behavior in subsequent operations.

How to fix

Add input validation checks for the `_memid` parameter to ensure it meets the expected format and is not empty.

Code

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    assert(!midIsRedeemed[_memid]);
    // chainlink request
    Chainlink.Request memory req = _buildOperatorRequest(
        jobId,
        this.fulfill.selector
    );
    // other code...
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Code Suggestion

```
function validateUnlock(string calldata _memid) public returns (bytes32 requestId) {
    require(bytes(_memid).length > 0, "Invalid memid");
    assert(!midIsRedeemed[_memid]);
    // chainlink request
    Chainlink.Request memory req = _buildOperatorRequest(
        jobId,
        this.fulfill.selector
    );
    // other code...
    midIsRedeemed[_memid] = true;
    return requestId;
}
```

Status

N/A

Comment

N/A

ISSUE-20: Type Casting Vulnerability in setJobId Function

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:setJobId

Description

The function setJobId contains a type casting vulnerability where a string is cast to bytes32. This can lead to data loss if the string exceeds 32 bytes, potentially causing the jobId to be set incorrectly, which is a security concern.

How to fix

Ensure the string length does not exceed 32 bytes before casting, or handle strings longer than 32 bytes appropriately.

Code

```
function setJobId(string memory _jobId) public onlyOwner {  
    jobId = bytes32(bytes(_jobId));  
}
```

Code Suggestion

```
function setJobId(string memory _jobId) public onlyOwner {  
    require(bytes(_jobId).length <= 32, "JobId exceeds maximum length");  
    jobId = bytes32(bytes(_jobId));  
}
```

Status

N/A

Comment

N/A

ISSUE-21: Inefficient Function Visibility

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:setOracleAddress

Description

The function `setOracleAddress` is marked as `public`, which is less gas efficient than marking it as `external` since it is only called externally. The `onlyOwner` modifier restricts access control, but the visibility setting does not pose a security risk, only a potential for gas optimization.

How to fix

Change the visibility of the function from `public` to `external` to optimize gas usage.

Code

```
function setOracleAddress(address _oracleAddress) public onlyOwner {  
    oracleAddress = _oracleAddress;  
    _setChainlinkOracle(_oracleAddress);  
}
```

Code Suggestion

```
function setOracleAddress(address _oracleAddress) external onlyOwner {  
    oracleAddress = _oracleAddress;  
    _setChainlinkOracle(_oracleAddress);  
}
```

Status

N/A

Comment

N/A

ISSUE-22: Inefficient Gas Usage Due to Public Visibility

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:withdrawFees

Description

The function `withdrawFees` is set to public visibility, which is not optimal for gas usage since it is only meant to be called externally. Changing the visibility to external can optimize gas costs.

How to fix

Change the visibility of the function from public to external.

Code

```
function withdrawFees() public {
    uint256 amount = balanceOf[treasury];
    assert(amount > 0);
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Code Suggestion

```
function withdrawFees() external {
    uint256 amount = balanceOf[treasury];
    assert(amount > 0);
    require(msg.sender == treasury, "err_invalid_caller");
    token.safeTransfer(treasury, amount);
    balanceOf[treasury] = 0;
}
```

Status

N/A

Comment

N/A

ISSUE-23: Floating Pragma Vulnerability

Severity

Note

Location

contracts/MemBridge.sol:MemBridge

Description

The use of a floating pragma allows the MemBridge contract to be compiled with any version of the compiler from 0.8.12 onwards, potentially leading to inconsistencies and vulnerabilities with future compiler versions that behave differently.

How to fix

Lock the pragma to a specific compiler version that has been thoroughly tested with the contract.

Code

```
// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.12;
```

Code Suggestion

```
// SPDX-License-Identifier: MIT  
pragma solidity 0.8.12;
```

Status

N/A

Comment

N/A

ISSUE-24: Proper Handling of Transfer Return Value

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:withdrawLink

Description

The code correctly checks the return value of the transfer function to ensure that the transfer was successful, which is a necessary check to prevent errors or malicious activity from causing unintended behavior.

How to fix

No action needed as the code already properly checks the return value.

Code

```
require(link.transfer(msg.sender, link.balanceOf(address(this))), "Unable to transfer");
```

Code Suggestion

```
require(link.transfer(msg.sender, link.balanceOf(address(this))), "Unable to transfer");
```

Status

N/A

Comment

N/A

ISSUE-25: Use of Magic Numbers in Fee Calculation

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:computeNetAmount

Description

The code uses a hardcoded value (10000) directly in the computation of bridge fees, which can lead to errors and makes the code less maintainable and readable. Using named constants instead of magic numbers is a best practice in programming.

How to fix

Define a constant at the beginning of the contract to replace the magic number 10000. This constant should be named to reflect its purpose, such as BASIS_POINTS_TOTAL.

Code

```
uint256 bfee = (_amount * bridgeFee) / 10000;
```

Code Suggestion

```
uint256 private constant BASIS_POINTS_TOTAL = 10000;  
  
uint256 bfee = (_amount * bridgeFee) / BASIS_POINTS_TOTAL;
```

Status

N/A

Comment

N/A

ISSUE-26: Proper Visibility Declaration of State Variables

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:StateVariables

Description

The state variables 'jobId', 'oracleFee', 'bridgeFee', 'oracleAddress', and 'treasury' are declared with explicit visibility, which is a good practice and aligns with security recommendations to prevent unintended access.

How to fix

No action needed as the state variables are already declared with explicit visibility which is a best practice.

Code

```
bytes32 private jobId;  
uint256 private oracleFee;  
uint256 private bridgeFee;  
address private oracleAddress;  
address private treasury;
```

Code Suggestion

N/A

Status

N/A

Comment

N/A

ISSUE-27: Restrictive Zero Amount Handling in fulfill Function

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:fulfill

Description

The ``require(_result > 0, "err_zero_amount");`` condition in the ``fulfill`` function is overly restrictive, potentially blocking valid zero results from the oracle which could be valid in certain contexts, such as when there is no balance to unlock. This could lead to unnecessary transaction reverts.

How to fix

Adjust the ``require()`` condition to allow zero results if they are valid in the given context.

Code

```
function fulfill(
    bytes32 _requestId,
    uint256 _result
) public recordChainlinkFulfillment(_requestId) returns (uint256) {
    string memory memid;
    // caller can't redeem memid with 0 amount
    require(_result > 0, "err_zero_amount");
    // retrieve the memid using the requestId and check its redeeming status
    memid = reqToMemId[_requestId];
    require(!midIsRedeemed[memid], "err_mid_redeemed");
    // map the chainlink request result to the corresponding requestId
    requests[_requestId] = _result;
    emit Request(_requestId, _result);
    return _result;
}
```

Code Suggestion

```
function fulfill(  
  bytes32 _requestId,  
  uint256 _result  
) public recordChainlinkFulfillment(_requestId) returns (uint256) {  
  string memory memid;  
  // Validate non-negative result, consider allowing zero if contextually  
  appropriate  
  require(_result >= 0, "err_negative_amount");  
  // retrieve the memid using the requestId and check its redeeming status  
  memid = reqToMemId[_requestId];  
  require(!midIsRedeemed[memid], "err_mid_redeemed");  
  // map the chainlink request result to the corresponding requestId  
  requests[_requestId] = _result;  
  emit Request(_requestId, _result);  
  return _result;  
}
```

Status

N/A

Comment

N/A

ISSUE-28: Proper Use of SafeERC20 for Token Transfers

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:lock

Description

The code correctly uses SafeERC20's `safeTransferFrom` method to handle ERC20 tokens that might not return a boolean on transfer, mitigating the potential vulnerability of incorrect token transfer handling.

How to fix

No action needed as the code already uses SafeERC20 which is designed to handle such cases.

Code

```
token.safeTransferFrom(msg.sender, address(this), _amount);
```

Code Suggestion

```
token.safeTransferFrom(msg.sender, address(this), _amount);
```

Status

N/A

Comment

N/A

ISSUE-29: Proper Use of SafeERC20 for Non-Boolean Token Transfers

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:executeUnlock

Description

The code correctly utilizes SafeERC20's `safeTransfer` method to address the issue of ERC20 tokens that might not return a boolean on transfer, mitigating potential vulnerabilities.

How to fix

No action needed as the code already uses SafeERC20 which is designed to handle such cases.

Code

```
token.safeTransfer(msg.sender, net_amount);
```

Code Suggestion

```
token.safeTransfer(msg.sender, net_amount);
```

Status

N/A

Comment

N/A

ISSUE-30: Proper Use of SafeERC20 for Token Transfers

Severity

Note

Location

contracts/MemBridge.sol:MemBridge:withdrawFees

Description

The code correctly uses SafeERC20's safeTransfer method to handle ERC20 tokens that might not return a boolean on transfer, mitigating the potential vulnerability of unhandled return values.

How to fix

No action needed as the code already uses SafeERC20 which is designed to handle such cases.

Code

```
token.safeTransfer(treasury, amount);
```

Code Suggestion

```
token.safeTransfer(treasury, amount);
```

Status

N/A

Comment

N/A

M-1: Contracts are vulnerable to fee-on-transfer accounting-related issues

Severity

Medium

Location

File: bridge.sol

Description

Consistently check account balance before and after transfers for Fee-On-Transfer discrepancies. As arbitrary ERC20 tokens can be used, the amount here should be calculated every time to take into consideration a possible fee-on-transfer or deflation.

Also, it's a good practice for the future of the solution.

Use the balance before and after the transfer to calculate the received amount instead of assuming that it would be equal to the amount passed as a parameter. Or explicitly document that such tokens shouldn't be used and won't be supported

Code

```
File: bridge.sol
```

```
158:         token.safeTransferFrom(msg.sender, address(this), _amount);
```

Status

N/A

Comment

N/A

M-2: Centralization Risk for trusted owners

Severity

Medium

Location

File: bridge.sol

Description

Impact:

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

Code

```
File: bridge.sol

214:     function withdrawLink() public onlyOwner {

240:     function setOracleAddress(address _oracleAddress) public onlyOwner {

247:     function getOracleAddress() public view onlyOwner returns (address) {

254:     function setJobId(string memory _jobId) public onlyOwner {

260:     function getJobId() public view onlyOwner returns (string memory) {

267:     function setFeeInJuels(uint256 _feeInJuels) public onlyOwner {

278:     ) public onlyOwner {
```

Status

N/A

Comment

N/A

M-3: Fees can be set to be greater than 100%.

Severity

Medium

Location

File: bridge.sol

Description

There should be an upper limit to reasonable fees.

A malicious owner can keep the fee rate at zero, but if a large value transfer enters the mempool, the owner can jack the rate up to the maximum and sandwich attack a user.

Code

File: bridge.sol

```
267:     function setFeeInJuels(uint256 _feeInJuels) public onlyOwner {
        oracleFee = _feeInJuels;

276:     function setFeeInHundredthsOfLink(
        uint256 _feeInHundredthsOfLink
    ) public onlyOwner {
        setFeeInJuels((_feeInHundredthsOfLink * LINK_DIVISIBILITY) / 100);
```

Status

N/A

Comment

N/A

L-1: Some tokens may revert when zero value transfers are made

Severity

Low

Location

File: bridge.sol

Description

Example: <https://github.com/d-xo/weird-erc20#revert-on-zero-value-transfers>.

In spite of the fact that EIP-20 states (<https://github.com/ethereum/EIPs/blob/46b9b698815abbfa628cd1097311deee77dd45c5/EIPS/eip-20.md?plain=1#L116>) that zero-valued transfers must be accepted, some tokens, such as LEND will revert if this is attempted, which may cause transactions that involve other tokens (such as batch operations) to fully revert. Consider skipping the transfer if the amount is zero, which will also save gas.

Code

File: bridge.sol

```
158:         token.safeTransferFrom(msg.sender, address(this), _amount);  
  
200:         token.safeTransfer(msg.sender, net_amount);  
  
229:         token.safeTransfer(treasury, amount);
```

Status

N/A

Comment

N/A

L-2: Missing checks for address(0) when assigning values to address state variables

Severity

Low

Location

File: bridge.sol

Description

Code

```
File: bridge.sol  
  
77:         treasury = _treasury; // 0x747D50C93e6821277805a2B80FE9CBF72EFCe6Cd  
  
241:         oracleAddress = _oracleAddress;
```

Status

N/A

Comment

N/A

L-3: Loss of precision

Severity

Low

Location

File: bridge.sol

Description

Division by large numbers may result in the result being zero, due to solidity not supporting fractions. Consider requiring a minimum amount for the numerator to ensure that it is always larger than the denominator

Code

```
File: bridge.sol
```

```
290:         return (oracleFee * 100) / LINK_DIVISIBILITY;
```

Status

N/A

Comment

N/A

L-4: Solidity version 0.8.20+ may not work on other chains due to PUSH0

Severity

Low

Location

File: bridge.sol

Description

The compiler for Solidity 0.8.20 switches the default target EVM version to Shanghai (<https://blog.soliditylang.org/2023/05/10/solidity-0.8.20-release-announcement/#important-note>), which includes the new PUSH0 op code. This op code may not yet be implemented on all L2s, so deployment on these chains will fail. To work around this issue, use an earlier EVM (<https://docs.soliditylang.org/en/v0.8.20/using-the-compiler.html?ref=zaryabs.com#setting-the-evm-version-to-target>) version (https://book.getfoundry.sh/reference/config/solidity-compiler#evm_version). While the project itself may or may not compile with 0.8.20, other projects with which it integrates, or which extend this project may, and those projects will have problems deploying these contracts/libraries.

Code

```
File: bridge.sol  
  
2: pragma solidity ^0.8.12;
```

Status

N/A

Comment

N/A

L-5: File allows a version of solidity that is susceptible to an assembly optimizer bug

Severity

Low

Location

File: bridge.sol

Description

In solidity versions 0.8.13 and 0.8.14, there is an optimizer bug (https://github.com/ethereum/solidity-blog/blob/499ab8abc19391be7b7b34f88953a067029a5b45/_posts/2022-06-15-inline-assembly-memory-side-effects-bug.md) where, if the use of a variable is in a separate assembly block from the block in which it was stored, the mstore operation is optimized out, leading to uninitialized memory. The code currently does not have such a pattern of execution, but it does use mstores in assembly blocks, so it is a risk for future changes. The affected solidity versions should be avoided if at all possible.

Code

```
File: bridge.sol  
  
2: pragma solidity ^0.8.12;
```

Status

N/A

Comment

N/A

L-6: Unsafe ERC20 operation(s)

Severity

Low

Location

File: bridge.sol

Description

Code

File: bridge.sol

```
217:          link.transfer(msg.sender, link.balanceOf(address(this))),
```

Status

N/A

Comment

N/A

GAS-1: $a = a + b$ is more gas effective than $a += b$ for state variables (excluding arrays and mappings)

Severity

Note

Location

File: bridge.sol

Description

This saves 16 gas per instance.

Code

```
File: bridge.sol

160:         balanceOf[msg.sender] += net_amount;

162:         balanceOf[treasury] += generateFees;

164:         totalLocked += net_amount;

166:         cumulativeFees += generateFees;

194:         balanceOf[treasury] += generateFees;

196:         cumulativeFees += generateFees;
```

Status

N/A

Comment

N/A

GAS-2: Using bools for storage incurs overhead

Severity

Note

Location

File: bridge.sol

Description

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past. See source (<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27>).

Code

```
File: bridge.sol
```

```
55:     mapping(string => bool) public midIsRedeemed;
```

Status

N/A

Comment

N/A

GAS-3: Use calldata instead of memory for function arguments that do not get mutated

Severity

Note

Location

File: bridge.sol

Description

When a function with a memory array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the calldata to the memory index. Each iteration of this for-loop costs at least 60 gas (i.e. $60 * \text{<mem_array>.length}$). Using calldata directly bypasses this loop.

If the array is passed to an internal function which passes the array to another internal function where the array is modified and therefore memory is used in the external call, it's still more gas-efficient to use calldata when the external function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one.

Saves 60 gas per instance

Code

```
File: bridge.sol
```

```
254:     function setJobId(string memory _jobId) public onlyOwner {
```

Status

N/A

Comment

N/A

GAS-4: For Operations that will not overflow, you could use unchecked

Severity

Note

Location

File: bridge.sol

Description

Code

```
File: bridge.sol

5: import {LinkTokenInterface} from "../lib/chainlink/contracts/src/v0.8/shared/
interfaces/LinkTokenInterface.sol";

6: import {ConfirmedOwner} from "../lib/chainlink/contracts/src/v0.8/shared/access/
ConfirmedOwner.sol";

7: import {Chainlink, ChainlinkClient} from "../lib/chainlink/contracts/src/v0.8/
ChainlinkClient.sol";

8: import {Strings} from "../lib/openzeppelin/contracts/utils/Strings.sol";

9: import "../lib/openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

76:         token = _btoken; // 0x779877A7B0D9E8603169DdbD7836e478b4624789 $LINK

77:         treasury = _treasury; // 0x747D50C93e6821277805a2B80FE9CBF72EFCe6Cd

78:         _setChainlinkToken(_linkTokenAddr); //
0x779877A7B0D9E8603169DdbD7836e478b4624789

79:         _setChainlinkOracle(_oracleAddress); //
0x0FaCf846af22BCE1C7f88D1d55A038F27747eD2B

80:         setJobId(_jobId); // "a8356f48569c434eaa4ac5fcb4db5cc0"

81:         setFeeInHundredthsOfLink(_ofee); // sepolia is zero $LINK fee

82:         bridgeFee = _bfee; // 0.25% for the launch so uint256(25)

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

104:         "/",

115:         '[ "content-type", "application/json", "set-cookie", "sid=14A52"] '

118:         req._add("contact", "https://t.me/decentland");

119:         req._addInt("multiplier", 1); // MEM store balances in BigInt as well

156:         uint256 generateFees = _amount - net_amount;

160:         balanceOf[msg.sender] += net_amount;

162:         balanceOf[treasury] += generateFees;

164:         totalLocked += net_amount;
```

```
166:         cumulativeFees += generateFees;

181:         uint256 generateFees = amount - net_amount;

192:         balanceOf[msg.sender] -= amount;

194:         balanceOf[treasury] += generateFees;

196:         cumulativeFees += generateFees;

198:         totalLocked -= net_amount;

207:         uint256 bfee = (_amount * bridgeFee) / 10000;

208:         return _amount - bfee;

279:         setFeeInJuels((_feeInHundredthsOfLink * LINK_DIVISIBILITY) / 100);

290:         return (oracleFee * 100) / LINK_DIVISIBILITY;
```

Status

N/A

Comment

N/A

GAS-5: Use Custom Errors instead of Revert Strings to save Gas

Severity

Note

Location

File: bridge.sol

Description

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas (<https://gist.github.com/IIIIII000/ad1bd0d29a0101b25e57c293b4b0c746>) each time they're hit by avoiding having to allocate and store the revert string (<https://blog.soliditylang.org/2021/04/21/custom-errors/#errors-in-depth>). Not defining the strings also save deployment gas

Additionally, custom errors can be used inside and outside of contracts (including interfaces and libraries).

Source: <https://blog.soliditylang.org/2021/04/21/custom-errors/> (<https://blog.soliditylang.org/2021/04/21/custom-errors/>):

Starting from Solidity v0.8.4 (<https://github.com/ethereum/solidity/releases/tag/v0.8.4>), there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to give more information about failures (e.g., `revert("Insufficient funds.");`), but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them.

Consider replacing all revert strings with custom errors in the solution, and particularly those that have multiple occurrences:

Code

```
File: bridge.sol

141:         require(_result > 0, "err_zero_amount");

144:         require(!midIsRedeemed[memid], "err_mid_redeemed");

183:         require(reqToCaller[_requestId] == msg.sender, "err_invalid_caller");

228:         require(msg.sender == treasury, "err_invalid_caller");
```

Status

N/A

Comment

N/A

GAS-6: Avoid contract existence checks by using low level calls

Severity

Note

Location

File: bridge.sol

Description

Prior to 0.8.10 the compiler inserted extra code, including EXTCODESIZE (100 gas), to check for contract existence for external function calls. In more recent solidity versions, the compiler will not insert these checks if the external call has a return value. Similar behavior can be achieved in earlier versions by using low-level calls, since low level calls never check for contract existence

Code

File: bridge.sol

```
217:                link.transfer(msg.sender, link.balanceOf(address(this))),
```

Status

N/A

Comment

N/A

GAS-7: State variables only set in the constructor should be declared immutable

Severity

Note

Location

File: bridge.sol

Description

Variables only set in the constructor and never edited afterwards should be marked as immutable, as it would avoid the expensive storage-writing operation in the constructor (around 20 000 gas per variable) and replace the expensive storage-reading operations (around 2100 gas per reading) to a less expensive value reading (3 gas)

Code

```
File: bridge.sol

76:         token = _btoken; // 0x779877A7B0D9E8603169DdbD7836e478b4624789 $LINK

77:         treasury = _treasury; // 0x747D50C93e6821277805a2B80FE9CBF72EFCe6Cd

82:         bridgeFee = _bfee; // 0.25% for the launch so uint256(25)
```

Status

N/A

Comment

N/A

GAS-8: Functions guaranteed to revert when called by normal users can be marked payable

Severity

Note

Location

File: bridge.sol

Description

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

Code

```
File: bridge.sol

214:     function withdrawLink() public onlyOwner {
240:     function setOracleAddress(address _oracleAddress) public onlyOwner {
247:     function getOracleAddress() public view onlyOwner returns (address) {
254:     function setJobId(string memory _jobId) public onlyOwner {
260:     function getJobId() public view onlyOwner returns (string memory) {
267:     function setFeeInJuels(uint256 _feeInJuels) public onlyOwner {
```

Status

N/A

Comment

N/A

GAS-9: Use != 0 instead of > 0 for unsigned integer comparison

Severity

Note

Location

File: bridge.sol

Description

Code

```
File: bridge.sol

141:         require(_result > 0, "err_zero_amount");

186:         balanceOf[msg.sender] >= amount && balanceOf[msg.sender] > 0,

227:         assert(amount > 0);
```

Status

N/A

Comment

N/A

NC-1: Missing checks for address(0) when assigning values to address state variables

Severity

Note

Location

File: bridge.sol

Description

Code

```
File: bridge.sol

77:         treasury = _treasury; // 0x747D50C93e6821277805a2B80FE9CBF72EFCe6Cd

241:         oracleAddress = _oracleAddress;
```

Status

N/A

Comment

N/A

NC-2: require() should be used instead of assert()

Severity

Note

Location

File: bridge.sol

Description

Prior to solidity version 0.8.0, hitting an assert consumes the remainder of the transaction's available gas rather than returning it, as require()/revert() do. assert() should be avoided even past solidity version 0.8.0 as its documentation (<https://docs.soliditylang.org/en/v0.8.14/control-structures.html#panic-via-assert-and-error-via-require>) states that "The assert function creates an error of type Panic(uint256). ... Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix. Additionally, a require statement (or a custom error) are more friendly in terms of understanding what happened."

Code

```
File: bridge.sol

94:         assert(!midIsRedeemed[_memid]);

227:         assert(amount > 0);
```

Status

N/A

Comment

N/A

NC-3: Use string.concat() or bytes.concat() instead of abi.encodePacked

Severity

Note

Location

File: bridge.sol

Description

Solidity version 0.8.4 introduces bytes.concat() (vs abi.encodePacked(<bytes>,<bytes>))

Solidity version 0.8.12 introduces string.concat() (vs abi.encodePacked(<str>,<str>), which catches concatenation errors (in the event of a bytes data mixed in the concatenation))

Code

```
File: bridge.sol
```

```
261:         return string(abi.encodePacked(jobId));
```

Status

N/A

Comment

N/A

NC-4: constants should be defined rather than using magic numbers

Severity

Note

Location

File: bridge.sol

Description

Even assembly (<https://github.com/code-423n4/2022-05-opensea-seaport/blob/9d7ce4d08bf3c3010304a0476a785c70c0e90ae7/contracts/lib/TokenTransferrer.sol#L35-L39>) can benefit from using readable constants instead of hex/numeric literals

Code

```
File: bridge.sol

105:         Strings.toHexString(uint256(uint160(msg.sender)), 20)

207:         uint256 bfee = (_amount * bridgeFee) / 10000;

279:         setFeeInJuels((_feeInHundredthsOfLink * LINK_DIVISIBILITY) / 100);

290:         return (oracleFee * 100) / LINK_DIVISIBILITY;
```

Status

N/A

Comment

N/A

NC-5: Duplicated require()/revert() Checks Should Be Refactored To A Modifier Or Function

Severity

Note

Location

File: bridge.sol

Description

Code

```
File: bridge.sol

183:         require(reqToCaller[_requestId] == msg.sender, "err_invalid_caller");

228:         require(msg.sender == treasury, "err_invalid_caller");
```

Status

N/A

Comment

N/A

NC-6: Event missing indexed field

Severity

Note

Location

File: bridge.sol

Description

Index event fields make the field more quickly accessible to off-chain tools (<https://ethereum.stackexchange.com/questions/40396/can-somebody-please-explain-the-concept-of-event-indexing>) that parse events. This is especially useful when it comes to filtering based on an address. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Where applicable, each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three applicable fields, all of the applicable fields should be indexed.

Code

File: bridge.sol

```
25:     event Lock(address address_, uint256 amount_);
```

```
26:     event Unlock(address address_, uint256 amount_);
```

Status

N/A

Comment

N/A

NC-7: Function ordering does not follow the Solidity style guide

Severity

Note

Location

File: bridge.sol

Description

According to the Solidity style guide (<https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-functions>), functions should be laid out in the following order :constructor(), receive(), fallback(), external, public, internal, private, but the cases below do not follow this pattern

Code

```
File: bridge.sol

1:
  Current order:
  public validateUnlock
  public fulfill
  external lock
  public executeUnlock
  internal computeNetAmount
  public withdrawLink
  public withdrawFees
  public setOracleAddress
  public getOracleAddress
  public setJobId
  public getJobId
  public setFeeInJuels
  public setFeeInHundredthsOfLink
  public getFeeInHundredthsOfLink

  Suggested order:
  external lock
  public validateUnlock
  public fulfill
  public executeUnlock
  public withdrawLink
  public withdrawFees
  public setOracleAddress
  public getOracleAddress
  public setJobId
  public getJobId
  public setFeeInJuels
  public setFeeInHundredthsOfLink
  public getFeeInHundredthsOfLink
  internal computeNetAmount
```

Status

N/A

Comment

N/A

NC-8: Functions should not be longer than 50 lines

Severity

Note

Location

File: bridge.sol

Description

Overly complex code can make understanding functionality more difficult, try to further modularize your code to ensure readability

Code

```
File: bridge.sol

175:     function executeUnlock(bytes32 _requestId) public {

206:     function computeNetAmount(uint256 _amount) internal view returns (uint256)
{

240:     function setOracleAddress(address _oracleAddress) public onlyOwner {

247:     function getOracleAddress() public view onlyOwner returns (address) {

254:     function setJobId(string memory _jobId) public onlyOwner {

260:     function getJobId() public view onlyOwner returns (string memory) {

267:     function setFeeInJuels(uint256 _feeInJuels) public onlyOwner {
```

Status

N/A

Comment

N/A

NC-9: Change int to int256

Severity

Note

Location

File: bridge.sol

Description

Throughout the code base, some variables are declared as int. To favor explicitness, consider changing all instances of int to int256

Code

```
File: bridge.sol
```

```
119:         req._addInt("multiplier", 1); // MEM store balances in BigInt as well
```

Status

N/A

Comment

N/A

NC-10: Lack of checks in setters

Severity

Note

Location

File: bridge.sol

Description

Be it sanity checks (like checks against 0-values) or initial setting checks: it's best for Setter functions to have them

Code

File: bridge.sol

```
240:     function setOracleAddress(address _oracleAddress) public onlyOwner {
        oracleAddress = _oracleAddress;
        _setChainlinkOracle(_oracleAddress);

254:     function setJobId(string memory _jobId) public onlyOwner {
        jobId = bytes32(bytes(_jobId));

267:     function setFeeInJuels(uint256 _feeInJuels) public onlyOwner {
        oracleFee = _feeInJuels;

276:     function setFeeInHundredthsOfLink(
        uint256 _feeInHundredthsOfLink
    ) public onlyOwner {
        setFeeInJuels((_feeInHundredthsOfLink * LINK_DIVISIBILITY) / 100);
```

Status

N/A

Comment

N/A

NC-11: Missing Event for critical parameters change

Severity

Note

Location

File: bridge.sol

Description

Events help non-contract tools to track changes, and events prevent users from being surprised by changes.

Code

File: bridge.sol

```
240:     function setOracleAddress(address _oracleAddress) public onlyOwner {
        oracleAddress = _oracleAddress;
        _setChainlinkOracle(_oracleAddress);

254:     function setJobId(string memory _jobId) public onlyOwner {
        jobId = bytes32(bytes(_jobId));

267:     function setFeeInJuels(uint256 _feeInJuels) public onlyOwner {
        oracleFee = _feeInJuels;

276:     function setFeeInHundredthsOfLink(
        uint256 _feeInHundredthsOfLink
    ) public onlyOwner {
        setFeeInJuels((_feeInHundredthsOfLink * LINK_DIVISIBILITY) / 100);
```

Status

N/A

Comment

N/A

NC-12: Incomplete NatSpec: @return is missing on actually documented functions

Severity

Note

Location

File: bridge.sol

Description

The following functions are missing @return NatSpec comments.

Code

```
File: bridge.sol

85:      /// @notice The function that reads data from MEM partof the bridge
      /// @dev After issuing an unlock on MEM function, use the memid of that
unlock req to fetch the unlockable amount
      /// This function send the request to the LinkWellNodes Chainlink's oracle
and receive the amount that the user
      /// can unlock for a given mem id.
      /// @param _memid The mem id of the issued unlock on the MEM serverless
function
      function validateUnlock(
          string calldata _memid
      ) public returns (bytes32 requestId) {

131:      /// @notice This function is called by the Chainlink oracle to resolve a
request
      /// @dev The fulfill function is self-desriptive within the Chainlink
usage context
      /// @param _requestId The oracle request ID
      /// @param _result The result of the requestId resolved by the oracle
function fulfill(
    bytes32 _requestId,
    uint256 _result
) public recordChainlinkFulfillment(_requestId) returns (uint256) {
```

Status

N/A

Comment

N/A

NC-13: Use a modifier instead of a require/if statement for a special msg.sender actor

Severity

Note

Location

File: bridge.sol

Description

If a function is supposed to be access-controlled, a modifier should be used instead of a require/if statement for more readability.

Code

```
File: bridge.sol

183:         require(reqToCaller[_requestId] == msg.sender, "err_invalid_caller");

228:         require(msg.sender == treasury, "err_invalid_caller");
```

Status

N/A

Comment

N/A

NC-14: Consider using named mappings

Severity

Note

Location

File: bridge.sol

Description

Consider moving to solidity version 0.8.18 or later, and using named mappings (<https://ethereum.stackexchange.com/questions/51629/how-to-name-the-arguments-in-mapping/145555#145555>) to make it easier to understand the purpose of each mapping

Code

```
File: bridge.sol

49:     mapping(address => uint) public balanceOf;

51:     mapping(bytes32 => uint256) public requests;

53:     mapping(bytes32 => string) public reqToMemId;

55:     mapping(string => bool) public midIsRedeemed;

57:     mapping(bytes32 => address) public reqToCaller;
```

Status

N/A

Comment

N/A

NC-15: Adding a return statement when the function defines a named return variable, is redundant

Severity

Note

Location

File: bridge.sol

Description

Code

```
File: bridge.sol

85:      /// @notice The function that reads data from MEM partof the bridge
      /// @dev After issuing an unlock on MEM function, use the memid of that
unlock req to fetch the unlockable amount
      /// This function send the request to the LinkWellNodes Chainlink's oracle
and receive the amount that the user
      /// can unlock for a given mem id.
      /// @param _memid The mem id of the issued unlock on the MEM serverless
function
    function validateUnlock(
        string calldata _memid
    ) public returns (bytes32 requestId) {
        // memid can be redeemed once
        assert(!midIsRedeemed[_memid]);
        // chainlink request
        Chainlink.Request memory req = _buildOperatorRequest(
            jobId,
            this.fulfill.selector
        );

        // construct the API req full URL
        string memory arg1 = string.concat("https://0xmem.net/vu/", _memid);
        string memory caller = string.concat(
            "/",
            Strings.toHexString(uint256(uint160(msg.sender)), 20)
        );
        string memory url = string.concat(arg1, caller);

        // Set Chain req object
        req._add("method", "GET");
        req._add("url", url);
        req._add("path", "amount");
        req._add(
            "headers",
            ["content-type", "application/json", "set-cookie", "sid=14A52"]
        );
        req._add("body", "");
        req._add("contact", "https://t.me/decentland");
        req._addInt("multiplier", 1); // MEM store balances in BigInt as well

        // 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 true
        midIsRedeemed[_memid] = true;
```

```
return requestId;
```

Status

N/A

Comment

N/A

NC-16: Strings should use double quotes rather than single quotes

Severity

Note

Location

File: bridge.sol

Description

See the Solidity Style Guide: <https://docs.soliditylang.org/en/v0.8.20/style-guide.html#other-recommendations>

Code

File: bridge.sol

```
115:          '['content-type', 'application/json', 'set-cookie', 'sid=14A52']'
```

Status

N/A

Comment

N/A

NC-17: Contract does not follow the Solidity style guide's suggested layout ordering

Severity

Note

Location

File: bridge.sol

Description

The style guide (<https://docs.soliditylang.org/en/v0.8.16/style-guide.html#order-of-layout>) says that, within a contract, the ordering should be:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

However, the contract(s) below do not follow this ordering

Code

File: bridge.sol

```
1:
  Current order:
  UsingForDirective.IERC20
  UsingForDirective.Chainlink.Request
  VariableDeclaration.token
  EventDefinition.Lock
  EventDefinition.Unlock
  EventDefinition.Request
  VariableDeclaration.jobId
  VariableDeclaration.oracleFee
  VariableDeclaration.bridgeFee
  VariableDeclaration.oracleAddress
  VariableDeclaration.treasury
  VariableDeclaration.cumulativeFees
  VariableDeclaration.totalLocked
  VariableDeclaration.balanceOf
  VariableDeclaration.requests
  VariableDeclaration.reqToMemId
  VariableDeclaration.midIsRedeemed
  VariableDeclaration.reqToCaller
  FunctionDefinition.constructor
  FunctionDefinition.validateUnlock
  FunctionDefinition.fulfill
  FunctionDefinition.lock
  FunctionDefinition.executeUnlock
  FunctionDefinition.computeNetAmount
  FunctionDefinition.withdrawLink
  FunctionDefinition.withdrawFees
  FunctionDefinition.setOracleAddress
  FunctionDefinition.getOracleAddress
  FunctionDefinition.setJobId
  FunctionDefinition.getJobId
  FunctionDefinition.setFeeInJuels
  FunctionDefinition.setFeeInHundredthsOfLink
  FunctionDefinition.getFeeInHundredthsOfLink
```

```
Suggested order:
UsingForDirective.IERC20
UsingForDirective.Chainlink.Request
VariableDeclaration.token
VariableDeclaration.jobId
VariableDeclaration.oracleFee
VariableDeclaration.bridgeFee
VariableDeclaration.oracleAddress
VariableDeclaration.treasury
VariableDeclaration.cumulativeFees
VariableDeclaration.totalLocked
```



```
VariableDeclaration.balanceOf  
VariableDeclaration.requests  
VariableDeclaration.reqToMemId  
VariableDeclaration.midIsRedeemed  
VariableDeclaration.reqToCaller  
EventDefinition.Lock  
EventDefinition.Unlock  
EventDefinition.Request  
FunctionDefinition.constructor  
FunctionDefinition.validateUnlock  
FunctionDefinition.fulfill  
FunctionDefinition.lock  
FunctionDefinition.executeUnlock  
FunctionDefinition.computeNetAmount  
FunctionDefinition.withdrawLink  
FunctionDefinition.withdrawFees  
FunctionDefinition.setOracleAddress  
FunctionDefinition.getOracleAddress  
FunctionDefinition.setJobId  
FunctionDefinition.getJobId  
FunctionDefinition.setFeeInJuels  
FunctionDefinition.setFeeInHundredthsOfLink  
FunctionDefinition.getFeeInHundredthsOfLink
```

Status

N/A

Comment

N/A

NC-18: Use Underscores for Number Literals (add an underscore every 3 digits)

Severity

Note

Location

File: bridge.sol

Description

Code

```
File: bridge.sol

207:         uint256 bfee = (_amount * bridgeFee) / 10000;
```

Status

N/A

Comment

N/A

NC-19: Internal and private variables and functions names should begin with an underscore

Severity

Note

Location

File: bridge.sol

Description

According to the Solidity Style Guide, Non-external variable and function names should begin with an underscore (<https://docs.soliditylang.org/en/latest/style-guide.html#underscore-prefix-for-non-external-functions-and-variables>)

Code

```
File: bridge.sol

32:     bytes32 private jobId;

34:     uint256 private oracleFee;

36:     uint256 private bridgeFee;

38:     address private oracleAddress;

40:     address private treasury;

206:     function computeNetAmount(uint256 _amount) internal view returns (uint256)
    {
```

Status

N/A

Comment

N/A

NC-20: Event is missing indexed fields

Severity

Note

Location

File: bridge.sol

Description

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

Code

```
File: bridge.sol

25:     event Lock(address address_, uint256 amount_);

26:     event Unlock(address address_, uint256 amount_);

27:     event Request(bytes32 indexed requestId_, uint256 result_);
```

Status

N/A

Comment

N/A

NC-21: Constants should be defined rather than using magic numbers

Severity

Note

Location

File: bridge.sol

Description

Code

```
File: bridge.sol
```

```
82:         bridgeFee = _bfee; // 0.25% for the launch so uint256(25)
```

Status

N/A

Comment

N/A

NC-22: public functions not called by the contract should be declared external instead

Severity

Note

Location

File: bridge.sol

Description

Code

```
File: bridge.sol

90:     function validateUnlock(
135:     function fulfill(
175:     function executeUnlock(bytes32 _requestId) public {
214:     function withdrawLink() public onlyOwner {
225:     function withdrawFees() public {
240:     function setOracleAddress(address _oracleAddress) public onlyOwner {
247:     function getOracleAddress() public view onlyOwner returns (address) {
260:     function getJobId() public view onlyOwner returns (string memory) {
284:     function getFeeInHundredthsOfLink()
```

Status

N/A

Comment

N/A

Disclaimer

We conducted our review of the smart contract codes solely based on the materials and documentation provided by the project under audit (the "Project").

Our audit employed a fine-tuned Artificial Intelligence (AI) system, which incorporates (i) a database of known vulnerability patterns that we have collected up until the date of our review for this audit, and (ii) results from a selection of existing contract analysis tools available as of the date of our review for this audit. While we endeavor to ensure the highest possible quality and accuracy of the results produced by our fine-tuned AI, we must clarify that the results are based on the state of the AI technology and understanding of smart contracts as of the date mentioned, and consequently absolute completeness and infallibility of the AI-generated results cannot be guaranteed.

In addition to the aforementioned AI-driven analysis, we may conduct manual audits. These are grounded in the knowledge and expertise we have accumulated up to the date of our review for this audit. The purpose of these manual audits is to identify and assess vulnerabilities specific to the project under audit.

Blockchain and smart contract technologies continue to evolve and may be susceptible to unforeseen risks and flaws. Consequently, while it is possible to minimize smart contract security risks, their complete elimination is inherently unattainable. Therefore, our audit does not claim to provide an exhaustive or all-encompassing review of all potential vulnerabilities.

Lastly, it is important to clarify that the scope of our audit is strictly limited to the analysis of smart contracts. Our audit does not extend to other layers or components, including but not limited to hardware, operating systems, programming languages, compilers, protocols, platforms, virtual machines, and imported libraries.

DISCLAIMER: You agree to the terms set forth in this disclaimer by reading this report or any part of it. Bunzz Pte. Ltd. and its affiliates (including shareholders, subsidiaries, employees, directors, and other representatives if any) ("Bunzz") provide this report for information purposes only. No party, including third parties, has the right to rely on this report or its contents. Bunzz assumes no responsibility or duty of care to any person and makes no warranty or representation about the accuracy or completeness of this report to any person. Bunzz provides this report "as-is," without any conditions, warranties, or other terms of any kind, and hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, fitness for purpose, merchantability, or non-infringement). Except to the extent that it is prohibited by law, Bunzz excludes all liability and responsibility, and you or any other person will not have any claim against Bunzz for any amount or kind of loss or damage that may result to you or any other person (including, without limitation, any direct, indirect, special, punitive, consequential, or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, whether in tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report, its use, the inability to use it, the results of its use, and any reliance on this report. For the avoidance of doubt, no one should consider or rely upon this report, its content, access, or usage as any form of financial, investment, tax, legal, regulatory, or other advice.