QuillAudits

Audit Report
November, 2024

For

magpie

Table of Content

Table of Content

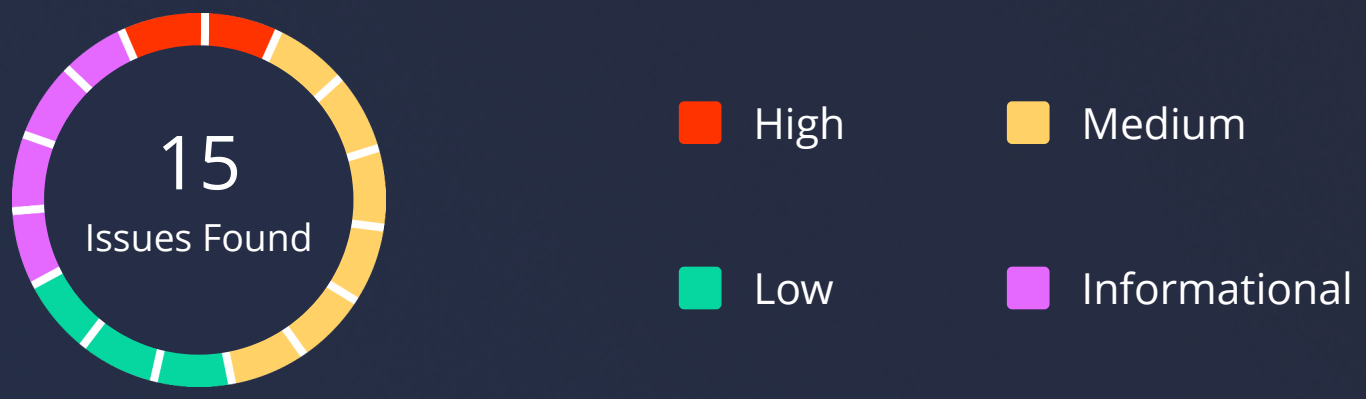
Executive Summary

Project Name	Magpie V2
Project URL	https://www.magpiefi.xyz/
Overview	The Intent-based protocol by Magpie transforms trade execution by allowing users to define desired outcomes through signed “Intents,” which specify transaction details while maintaining asset control. Solvers in a competitive network fulfill these Intents by routing orders across diversified liquidity sources and covering gas fees, providing users with optimal trade prices, cross-chain support, and gas fee savings, all without manual swaps or bridge complexities.
Audit Scope	https://github.com/magpieprotocol/magpie-contracts/tree/intents
Contracts In-Scope	**Branch: Intents** contracts/MagpieSymbiosisBridge.sol contracts/MagpieStargateBridgeV3.sol contracts/MagpieRouterV3.sol contracts/MagpieCCTPBridge.sol contracts/MagpieCelerBridgeV2.sol contracts/libraries/LibRouter.sol contracts/libraries/LibAsset.sol contracts/libraries/LibBridge.sol contracts/interfaces/celer/ILiquidityBridge.sol contracts/interfaces/celer/IMessageBus.sol contracts/interfaces/IMagpieRouterV3.sol contracts/interfaces/IMagpieStargateBridgeV3.sol contracts/interfaces/IMagpieSymbiosisBridge.sol contracts/interfaces/IWETH.sol contracts/interfaces/IBridge.sol contracts/interfaces/IMagpieCelerBridgeV2.sol contracts/interfaces/IMagpieCCTPBridge.sol contracts/interfaces/stargate/IOFT.sol contracts/interfaces/cctp/IReceiver.sol contracts/interfaces/stargate/IStargate.sol contracts/interfaces/axelar/IAxelarGasService.sol contracts/interfaces/axelar/IAxelarGateway.sol

Executive Summary

Commit Hash	49bd796b7bfbf137ad7430c833d21feae9ea925d
Language	Solidity
Blockchain	Ethereum, Polygon, BSC, Avalanche, Arbitrum, Optimism, Polygon zkEVM, Base, zkSync, Blast, Manta, Scroll, Metis, Fantom, Taiko
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	2nd Octoberber 2024 - 31st October 2024
Updated Code Received	17th November 2024
Second Review	18th November 2024 - 20th November 2024
Fixed In	https://github.com/magpieprotocol/magpie-contracts/tree/intents
Third Review Scope	https://github.com/magpieprotocol/magpie-contracts/pull/318
Third Review	13th may 2025
Note	starting **May 2025** Magpie has been rebranded as fly.trade . Same platform, new name - reflecting a refreshed identity and direction.

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	5	2	4

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. The Refund Address Can Be Set To Internal Caller In MagpieCelerBridgeV2

Path

MagpieCelerBridgeV2.sol

Function

swapIn

Description

In MagpieCelerBridgeV2, within the swapIn function the refundAddresses[depositDataHash] is always set to msg.sender. However, the swapIn can be called either by swapInWithMagpieSignature or swapInWithUserSignature which have different authorisation. Therefore, whenever swapInWithUserSignature is called, the refundAddresses can be set to the internal caller. As a result, in the event of the bridge transfer failure, the tokens will be transferred back to the incorrect address.

```
  /// @dev See {IMagpieCelerBridgeV2-swapInWithMagpieSignature}
  function swapInWithMagpieSignature(bytes calldata) external payable whenNotPaused
returns (uint256 amountOut) {
      SwapData memory swapData = LibRouter.getData();
      amountOut = swapIn(swapData, true);
  }


  /// @dev See {IMagpieCelerBridgeV2-swapInWithUserSignature}
  function swapInWithUserSignature(bytes calldata) external payable onlyInternalCaller
returns (uint256 amountOut) {
      SwapData memory swapData = LibRouter.getData();
      if (swapData.fromAssetAddress.isNative()) {
          revert InvalidAddress();
      }
      amountOut = swapIn(swapData, false);
  }
...
  function swapIn(SwapData memory swapData, bool useCaller) private returns (uint256
amountOut) {
...
itDataHash] = msg.sender;
...
```

Recommendation

It is recommended to set the refund address to the fromAddress.

Status

Resolved

2. MagpieStargateBridgeV3 Lacks Bridge Slippage

Path

MagpieStargateBridgeV3.sol

Function

bridgeIn()

Description

In MagpieStargateBridgeV3, within the bridgeIn function, the quoteOFT is called to determine the expected amount of tokens received for the user after bridging (the minAmountLD variable). However, this amount is taken as is, without any extra check against the user's slippage. On the contrary, the MagpieCelerBridgeV2 has user supplied slippage implemented.

```
...
IStargate stargate = IStargate(currentStargateAddress);


    (, , OFTReceipt memory receipt) = stargate.quoteOFT(sendParam);
    sendParam.minAmountLD = receipt.amountReceivedLD;


    stargate.sendToken{value: valueToSend}(
        sendParam,
        MessagingFee({nativeFee: bridgeFee, lzTokenFee: 0}),
        refundAddress
    );
...
```

Recommendation

It is recommended to implement and use user's supplied slippage in the MagpieStargateBridgeV3.

Status

Resolved

Medium Severity Issues

3. MagpieCelerBridgeV2 Does Not Support Native Token Bridging

Path

MagpieCelerBridgeV2.sol

Function

bridgeIn

Description

Within the MagpieCelerBridgeV2, the bridgeIn function has an ‘if’ condition suggesting that native tokens can be bridged as well. However, the final call to the sendMessageWithTransfer only passes bridgeFee and not the bridgeFee + amount (amountOut from swapIn). Also, on the contrary the MagpieStargateBridgeV3 calculates the valueToSend, to determine the amount that should be sent. Ultimately, within the Celer Network the separate function exists to send native tokens: sendNative. Thus, this bridge only supports ERC20 tokens and wrapped native token, but not native.

```
...
        address liquidityBridgeAddress = messageBus.liquidityBridge();
        if (!toAssetAddress.isNative()) {
            toAssetAddress.approve(liquidityBridgeAddress, amount);
        }
...
```

Recommendation

It is recommended to either implement native token bridging or explicitly disable it.

Status

Resolved

4. Permit Front-run Affects the Legitimate Execution Call

Path

magpie-contracts/contracts/MagpieRouterV3.sol

Function

estimateSwapGas, swapWithMagpieSignature, swapWithUserSignature

Description

Within the MagpieRouterV3 contract three functions: estimateSwapGas, swapWithMagpieSignature, swapWithUserSignature make use of permit functionality for ERC20 tokens to set gas-less allowance. However, the permit function can be called by anyone anytime whenever the permit's signature becomes public. Thus, an adversary can extract the permit signature from the functions' calldata, front-run legitimate call and execute ERC20 token's permit. As a result, the legitimate call will revert as the permit's nonce is incremented and the signature is not valid anymore.

Recommendation

It is recommended to wrap the call to the permit function within the try-catch block. With such security control any attempt to permit front-run will not affect the execution of the legitimate call.

Status

Resolved

5. Signature Replay Possible for swapWithMagpieSignature

Path

magpie-contracts/contracts/MagpieRouterV3.sol

Function

swapWithMagpieSignature

Description

Within the MagpieRouterV3 contract the swapWithMagpieSignature allows the user to trigger swap operation, if only the user has been given the valid signature signed by the internalCaller. However, this signature-based authorisation is flawed as it does not include nonce in the signed message. As a result, user can replay the signature multiple times triggering multiple swap operations.

This state persists until the internalCaller is not revoked.

Recommendation

It is recommended to include the per-signature nonce that is included in the signed message. After every use, the used nonce should be marked as spent and protocol should expect incremented value in the subsequent signature checks.

Status

Acknowledged

6. Signature Is Not Bound To Particular User in swapWithMagpieSignature

Path

magpie-contracts/contracts/MagpieRouterV3.sol

Function

swapWithMagpieSignature

Description

Within the MagpieRouterV3 contract the swapWithMagpieSignature allows the user to trigger swap operation, if only the user has been given the valid signature signed by the internalCaller. However, this signature-based authorisation is flawed as it does not include from address in the signed message - thus, it is not bound to a particular user. As a result, any user can use any signature generated by the internal caller, triggering swap operation.

Recommendation

It is recommended to include the from address that is included in the signed message. Whenever signature verification is performed, it should check whether the signature is created for a particular msg.sender.

Status

Resolved

7. Ineffective Implementation of the Emergency-Stop Pattern

Path

magpie-contracts/contracts/MagpieRouterV3.sol

Function

swapWithMagpieSignature

Description

The MagpieRouterV3 contract implements the Pausable library. Functions: estimateSwapGas and swapWithMagpieSignature are marked as whenNotPaused. However, there is no public or external function that allows the protocol's owner to toggle paused state on or off. Thus, it is not possible to use the emergency-stop pattern in the event of an emergency situation.

Recommendation

It is recommended to expose functions for protocol's owner to update the pausability state.

Status

Resolved

8. The executeMessageWithTransferRefund can revert due to various reasons

Path

MagpieCelerBridgeV2.sol

Function

executeMessageWithTransferRefund

Description

The MagpieCelerBridgeV2 contract implements the executeMessageWithTransferRefund function, which is designed to handle bridge's refund in the event of unsuccessful bridging. However, this function attempts to immediately transfer refunded assets to the user. This feature may revert in following use cases:

- Whenever a refunded asset is a native token and the receiver address is a contract with custom receive()/fallback function, the transaction can revert due to Out Of Gas error.
- Whenever a refunded asset is a USDC or similar token implementing the blacklisting, whenever the address get blacklisted between bridging and refund, it will revert due to ERC20 transfer revert.

```
function executeMessageWithTransferRefund(
    address assetAddress,
    uint256 amount,
    bytes calldata payload,
    address
) external payable onlyCeler returns (IMessageBus.TxStatus) {
    bytes32 depositDataHash = LibBridge.decodeDepositDataHash(payload);

    address receiver = refundAddresses[depositDataHash];

    if (receiver == address(0)) {
        revert InvalidRefundAddress();
    }

    refundAddresses[depositDataHash] = address(0);

    if (msg.value >= amount) {
        assetAddress = address(0);
    }
```

```
assetAddress.transfer(receiver, amount);

        emit Refund(receiver, assetAddress, depositDataHash, amount);

        return IMessageBus.TxStatus.Success;
    }

    function transfer(address self, address recipient, uint256 amount) internal {
        if (self.isNative()) {
            (bool success, ) = payable(recipient).call{value: amount}("");
            if (!success) {
                revert TransferFailed();
            }
        } else {
            uint256 ptr;
            assembly {
                ptr := mload(0x40)
                mstore(0x40, add(ptr, 68))
                mstore(ptr,
0xa9059cbb00000000000000000000000000000000000000000000000000000000)
                mstore(add(ptr, 4), recipient)
                mstore(add(ptr, 36), amount)
            }
            if (!execute(self, 0, ptr, 68, 0, 0)) {
                revert TransferFailed();
            }
        }
    }
```

Recommendation

It is recommended to either change the approach from push to pull or alternatively, implement the escrow mechanism which will held refunded tokens until user claims them.

Status

Resolved

Low Severity Issues

9. The MagpieStargateBridgeV3 Lacks quoteSend

Path

MagpieStargateBridgeV3.sol

Function

bridgeIn

Description

The MagpieStargateBridgeV3 contract allows to perform swap and then bridge the obtained token to the Stargate. However, it does not check whether the provided bridge fee in native tokens is sufficient to fulfill the request. Thus, the transaction call can revert in the late processing consuming more Gas than required. Additionally, the contract does not expose any function to calculate the required amount of Gas off-chain.

The Stargate implements LayerZero's OFT standard. This protocol exposes quoteSend function, which can provide an amount of bridge fee required to fulfill the request. This function takes into account the size of the SendParam.

Recommendation

It is recommended to implement an external view function that calculates the Stargate's bridge fee required to fulfill the request by means of the quoteSend. Additionally, it is recommended to check whether quoteSend received a sufficient amount of native tokens.

Status

Acknowledged

10. The swapOut Can Finish Without Consuming Allowance

Path

contracts/libraries/LibBridge.sol

Function

swapOut

Description

The swapOut function in the LibBridge library attempts to swap tokens considering an exchange between native and non-native tokens. Whenever a swap occurs for not equal non-native tokens, the function firstly calls approve for router contract for fromAssetAddress token with amountIn as amount. Then, it attempts to swap tokens, which in the event of failure returns a boolean flag. If the swap is unsuccessful, the fromAssetAddress tokens are transferred to the toAddress instead. However, the allowance set before the swap operation is not reset.

```
...
        } else {
            // We dont need signature, we validate against crosschain message
            uint256 nativeAmount = 0;
            if (swapData.fromAssetAddress.isNative()) {
                nativeAmount = swapData.amountIn;
            } else {
                swapData.fromAssetAddress.approve(routerAddress, swapData.amountIn);
            }
            bool success = false;
            (amountOut, success) = swap(routerAddress, nativeAmount);
            if (!success) {
                swapData.fromAssetAddress.transfer(swapData.toAddress, swapData.amountIn);
                swapData.toAssetAddress = swapData.fromAssetAddress;
                amountOut = swapData.amountIn;
            }
        }
...
```

Recommendation

It is recommended to reset the allowance to 0 in the event of an unsuccessful swap operation.

Status

Resolved

11. MagpieCelerBridgeV2's swapOut Function Lacks Condition Check

Path

MagpieCelerBridgeV2.sol

Function

swapOut()

Description

The swapOut function in the MagpieCelerBridgeV2 contract has a condition check to handle the event when Celer changes the mapping for native token address. However, this if and else if conditions lack final else fallback, to handle citation when depositAmount is equal to 0 and it is not native asset.

```
...
        uint256 depositAmount = deposit[depositDataHash][swapData.fromAssetAddress];
        if (depositAmount > 0) {
            deposit[depositDataHash][swapData.fromAssetAddress] = 0;
        } else if (swapData.fromAssetAddress.isNative()) {
            // If celer changes the mapping it makes sure that the user can still swap the from
asset as native token
            depositAmount = deposit[depositDataHash][weth];
            if (depositAmount == 0) {
                revert DepositIsNotFound();
            }
            weth.unwrap(depositAmount);
            deposit[depositDataHash][weth] = 0;
        }
...
```

Recommendation

It is recommended to implement else fallback in the aforementioned conditions check, to revert the transaction early in the processing.

Status

Resolved

Informational Issues

12. The UniswapV3SwapCallback Does Not Check Caller

Path

MagpieRouterV3.sol

Function

fallback()

Description

The MagpieRouterV3 implements fallback() function to handle the incoming uniswapV3SwapCallback. The Uniswap's documentation states that The caller of this method must be checked to be a UniswapV3Pool deployed by the canonical UniswapV3Factory. However, no such check is implemented, which is noted in the developer's comments. The router contract is meant to not hold any ERC20 tokens. Nevertheless, in the event of having positive balance for some ERC20 tokens, this function can be abused to sweep them. The finding was reported as a deviation from leading security practices.

Recommendation

It is recommended to check whether the caller is a UniswapV3Pool contract.

Status

Resolved

13. Not Optimized Storage Usage Consumes More Gas

Path

MagpieCCTPBridge.sol, MagpieCelerBridgeV2.sol, MagpieStargateBridgeV3.sol, MagpieSymbiosisBridge.sol

Function

swapSequence state variable

Description

The *Bridge contracts have defined the swapSequence variable with uint64 type. Thus, this state variable occupies only 8 bytes from the storage slot. However, it is declared in between two other variables that occupy full slots. Eventually, such order of variables impacts the storage slots used, as swapSequence starts occupying a new slot, when it could be attached to other variable which occupies only 20 bytes out of 32.

```
mapping(address => bool) public internalCaller;
address public weth;
bytes32 public networkIdAndRouterAddress;
uint64 public swapSequence;
mapping(bytes32 => mapping(address => uint256)) public deposit;
mapping(address => address) public assetToStargate;
mapping(address => address) public stargateToAsset;
address public lzAddress;
```

```
forge inspect MagpieStargateBridgeV3  storage-layout —pretty
| Name                | Type                                | Slot | Offset | Bytes | Contract
|
|———————————————|———————————————————————————|————|
———————|————|———————————————————————————————|
| _owner              | address                             | 0    | 0      | 20    | contracts/
MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| _pendingOwner       | address                             | 1    | 0      | 20    | contracts/
MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| _paused             | bool                                | 1    | 20     | 1     | contracts/
MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| internalCaller      | mapping(address => bool)            | 2    | 0      | 32    |
contracts/MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| weth                | address                             | 3    | 0      | 20    |
```

contracts/MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| networkIdAndRouterAddress | bytes32 | 4 | 0 | 32 |
contracts/MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| swapSequence | uint64 | 5 | 0 | 8 | contracts/
MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| deposit | mapping(bytes32 => mapping(address => uint256)) | 6 | 0 | 32
| contracts/MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| assetToStargate | mapping(address => address) | 7 | 0 | 32 |
contracts/MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| stargateToAsset | mapping(address => address) | 8 | 0 | 32 |
contracts/MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |
| lzAddress | address | 9 | 0 | 20 | contracts/
MagpieStargateBridgeV3.sol:MagpieStargateBridgeV3 |

Recommendation

It is recommended to change the order of variables declared, so less storage's slots are occupied by the contract.

Status

Resolved

14. The execute Function Can Be Front-Runned

Path

MagpieCCTPBridge.sol, MagpieSymbiosisBridge.sol

Function

execute()

Description

The MagpieCCTPBridge and MagpieSymbiosisBridge contracts implement the execute() function which is meant to be called by the gateway. However, this function can be called by anyone. If the function is front-run, the legitimate call by the gateway will be reverted, as a second call with the same data to validateContractCall function will revert. No identified risk is associated with this behavior in on-chain processing, however, it may have an impact on the off-chain processing. If off-chain expects that this function is called only by specific address and processes the emitted event only in such a case, it may pose a risk that the call will be unhandled.

Recommendation

It is recommended to review the off-chain services, whether they are prepared for the aforementioned behavior.

Status

Resolved

15. MagpieCCTPBridge and MagpieSymbiosisBridge Do Not Disable Native Transfers Entirely

Path

MagpieCCTPBridge.sol, MagpieSymbiosisBridge.sol

Function

swapInWithMagpieSignature()

Description

The MagpieCCTPBridge is meant to bridge only USDC tokens. It reverts whenever native asset is attempted to be bridged within the swapInWithUserSignature. However, there is no similar check implemented in the swapInWithMagpieSignature function. Thus, in the event of such a transfer attempt, the transaction reverts in late processing.

Similar behavior can be observed within the MagpieSymbiosisBridge contract.

```
  /// @dev See {IMagpieCCTPBridge-swapInWithMagpieSignature}
  function swapInWithMagpieSignature(bytes calldata) external payable whenNotPaused
returns (uint256 amountOut) {
      SwapData memory swapData = LibRouter.getData();
      amountOut = swapIn(swapData, true);
  }

  /// @dev See {IMagpieCCTPBridge-swapInWithUserSignature}
  function swapInWithUserSignature(bytes calldata) external payable onlyInternalCaller
returns (uint256 amountOut) {
      SwapData memory swapData = LibRouter.getData();
      if (swapData.fromAssetAddress.isNative()) {
          revert InvalidAddress();
      }
      amountOut = swapIn(swapData, false);
  }
```

Recommendation

It is recommended to revert an attempt to transfer native assets in every case within the MagpieCCTPBridge contract.

Status

Resolved

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Magpie V2. We performed our audit according to the procedure described above.

Some issues of High,low,medium and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture. In The End, Magpie Team Resolved almost all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Magpie V2. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Magpie V2. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of Magpie V2 to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2024

For

magpie

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com