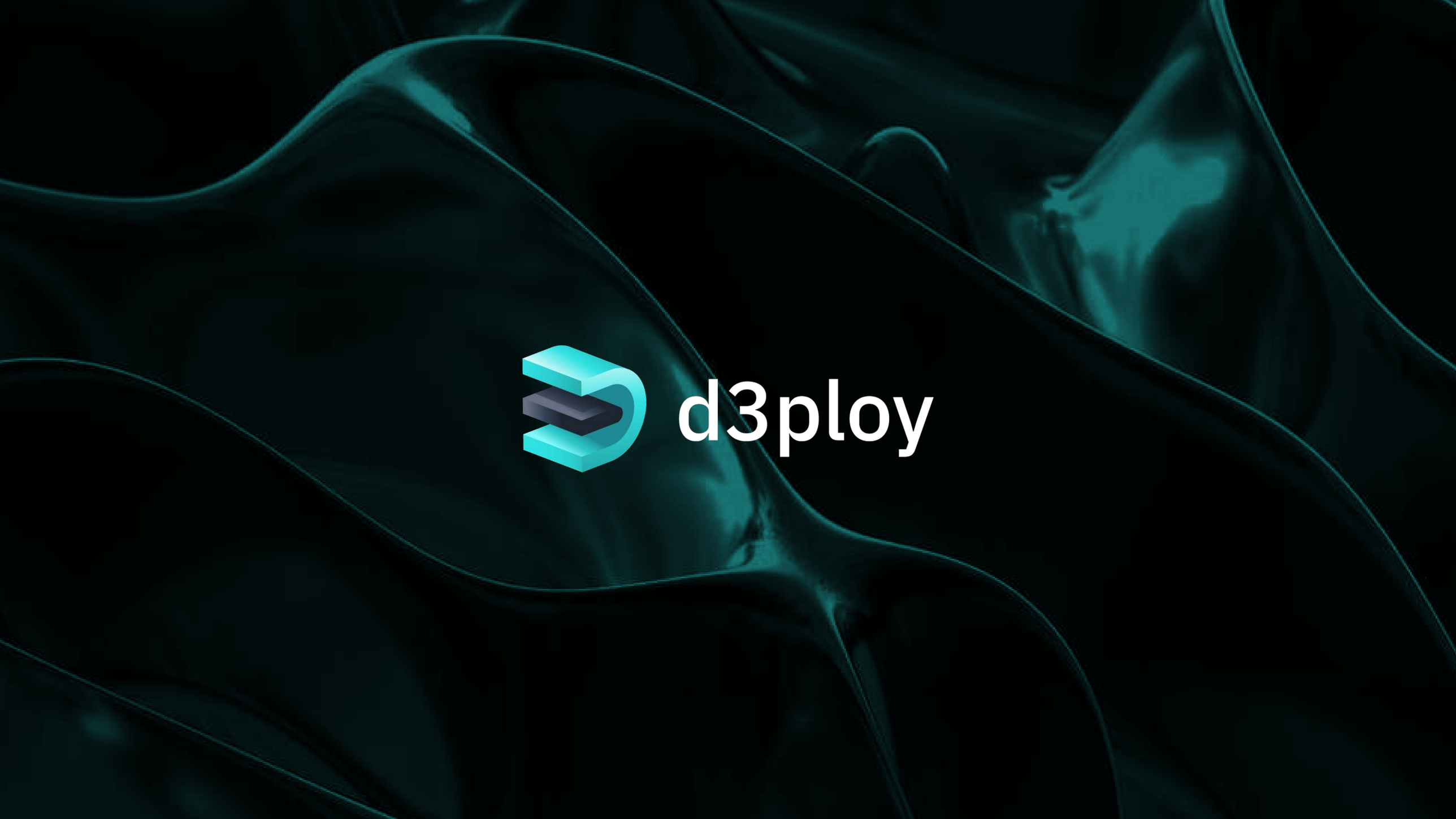
d3ploy

d3ploy

SECURITY ASSESSMENT

Magpie Protocol

October 20th 2023

TABLE OF

Contents

LEGAL

Disclaimer

D3ploy audits are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts d3ploy to perform a security review. D3ploy does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

D3ploy audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort. The report is provided only for the contract(s) mentioned in the report and does not include any other potential additions and/or contracts deployed by Owner. The report does not provide a review for contract(s), applications and/or operations, that are out of this report scope.

D3ploy’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

D3ploy represents an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. D3ploy’s position is that each company and individual are responsible for their own due diligence and continuous security. The security audit is not meant to replace functional testing done before a software release. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits and a public bug bounty program to ensure the security of the smart contracts.

D3PLOY

Introduction

D3ploy is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Secure your project with d3ploy

We offer field-proven audits with in-depth reporting and a range of suggestions to improve and avoid contract vulnerabilities. Industry-leading comprehensive and transparent smart contract auditing on all public and private blockchains.

Vunerability checking

A crucial manual inspection carried out to eliminate any code flaws and security loopholes. This is vital to avoid vulnerabilities and exposures incurring costly errors at a later stage.

Contract verification

A thorough and comprehensive review in order to verify the safety of a smart contract and ensure it is ready for launch and built to protect the end-user

Risk assessment

Analyse the architecture of the blockchain system to evaluate, assess and eliminate probable security breaches. This includes a full assessment of risk and a list of expert suggestions.

In-depth reporting

A truly custom exhaustive report that is transparent and depicts details of any identified threats and vulnerabilities and classifies those by severity.

Fast turnaround

We know that your time is valuable and therefore provide you with the fastest turnaround times in the industry to ensure that both your project and community are at ease.

Best-of-class blockchain engineers

Our engineers combine both experience and knowledge stemming from a large pool of developers at our disposal. We work with some of the brightest minds that have audited countless smart contracts over the last 4 years.

PROJECT

Introduction

Magpie protocol is a cross-chain liquidity aggregator that enables seamless cross-chain swaps with near-instant finality and cost efficiency on many of the top blockchains, all without the need to bridge any assets, making for an extremely fast, secure, easy, and gas efficient solution.

Magpie protocol incorporates a unique technical implementation that allows execution of cross-chain swaps without the need for the user to bridge assets from any of the top bridges. This saves time and cost by reducing the complexity and risks involved in using any of the bridging solutions to move assets across chains.

Project Name Magpie Protocol

Contract Name FLY Token

Contract Address -

Contract Chain Not Yet Deployed on Mainnet

Contract Type Smart Contract

Platform EVM

Language Solidity

Network Ethereum (ERC20)

Codebase Private GitHub Repository

Total Token Supply -

INFO

Social

https://www.magpiefi.xyz/

https://twitter.com/magpieprotocol

https://t.me/magpieprotocol

https://discord.gg/CwJuFeHp6f

https://medium.com/@Magpieprotocol

https://github.com/magpieprotocol/

contact@magpiefi.xyz

AUDIT

Score

Issues	12
Critical	0
Major	0
Medium	4
Minor	1
Informational	2
Discussion	5

All issues are described in further detail on the following pages.

AUDIT Scope

CODEBASE FILES

magpieprotocol/magpie-contracts/contracts/

LOCATION

- Private Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for Magpie Protocol to discover issues and vulnerabilities in the source code of the Magpie Protocol project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic, Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective in the comments below.

TIMESTAMP

Version *v1.0*

Date *2023/10/13*

Descrption *Layout project*

Architecture / Manual review / Static & dynamic security testing

Summary

Version *v1.1*

Date *2023/10/20*

Descrption *Re-audit applied fixes*

Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
Unreturned Excess Ether in Deposit Function	✦ Medium	Fixed
Inadequate Handling of Tokens with Fees On Transfer in Uniswap V2 Swap	✦ Medium	Fixed
Inadequate Approval Handling for Curve LP Token Swaps	✦ Medium	Fixed
Uniswap Swap Function with Insecure Deadline	✦ Medium	Fixed
Use Ownable2Step	✦ Minor	Fixed
Use of transfer and send Functions in Smart Contract	✦ Informational	Fixed
Missing State Variable Visibility	✦ Informational	Acknowledged
Public Function Visibility in LibRouter Library	✦ Gas	Fixed
Internal Function Never Used	✦ Gas	Fixed
Gas Optimization in Increments	✦ Gas	Acknowledged

KEY Finding

TITLE	SEVERITY	STATUS
Array Length Caching	✦ Informational	Fixed
Unnecessary Checked Arithmetic in Loops	✦ Gas	Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

In the provided contracts deposit() function, the contract accepts Ether from the user and converts it to Wrapped Ether (WETH) if it's native Ether. However, there's an issue with handling excess Ether sent by the user. If the user sends more Ether than the specified amount, the contract does not return the excess Ether to the user, thereby, causing a loss of funds.

AFFECTED CODE

- LibAsset.sol#deposit()

Issue : Unreturned Excess Ether in Deposit Function

Level : Medium

Remediation : To address this issue, consider implementing a mechanism to refund excess Ether to the user if they send more than the specified amount.

Alleviation / Retest : This is a feature and not a bug because the excess amount is paid to the bridge or the data transfer protocol as a fee.

IN - DEPTH Vulnerabilities

DESCRIPTION

The provided smart contract swapUniswapV2 function uses the swapExactTokensForTokens function from Uniswap V2 for token swaps. However, this function doesn't handle tokens that charge fees on transfer correctly. When swapping tokens with fees on transfer, the contract should use swapExactTokensForTokensSupportingFeeOnTransferTokens instead.

Tokens that charge fees on transfer, often referred to as "reflect" or "RFI" tokens, are designed so that a portion of the transferred amount is retained by the token contract. Using the standard swapExactTokensForTokens function can result in unexpected behavior, such as a smaller amount received after the swap due to the retained fees.

AFFECTED CODE

- LibUniswapV2.sol#swapUniswapV2()

Issue : Inadequate Handling of Tokens with Fees On Transfer in Uniswap V2 Swap

Level : Medium

Remediation : To handle tokens with fees on transfer correctly, it's recommended to use the swapExactTokensForTokensSupportingFeeOnTransferTokens() function provided by Uniswap V2.

Alleviation / Retest : This is fixed by adding another function swapUniswapV2Withfee() that handles fees on transfer.

Vulnerabilities

DESCRIPTION

In swapCurveLp() function, it performs multi-step token swaps through the Curve protocol. To facilitate these swaps, the contract approves tokens for each iteration. However, the issue lies in the approval logic for the next swap. For each swap iteration, you approve h.amountIn, which remains constant for all iterations. Ideally, you should be approving the amount that was received from the previous swap, which becomes the amountIn for the next one.

This means that the contract doesn't take into account the changing amountIn after each swap. In other words, the amount approved might not match the actual tokens available for the next swap. This could result in unexpected behavior and potential issues during the multi-step swap process.

AFFECTED CODE

- LibCurveLp.sol#swapCurveLp()

Issue : Inadequate Approval Handling for Curve LP Token Swaps

Level : Medium

Remediation : To address this issue and ensure the correct amount is approved for each swap . amountIn is being updated after every swap. It should use amountIn to approve tokens.

Alleviation / Retest : This is fixed by using the updated amount after each swap.

Vulnerabilities

DESCRIPTION

The provided Contracts swapUniswapV3V1 function contains a vulnerability related to the use of the block.timestamp as the deadline for Uniswap swaps. This allows a malicious miner or sequencer to manipulate the execution of the swap, potentially profiting from front-running or arbitrage opportunities.

In Ethereum, the block.timestamp represents the current block's timestamp, and it can be manipulated to a certain extent by miners or sequencers. This means that an attacker with control over when to mine or include transactions in a block can delay or reorder transactions to their advantage.

By setting the deadline to block.timestamp, the function makes it possible for a miner or sequencer to control when the swap transaction is actually executed, potentially gaining an advantage in price movements.

AFFECTED CODE

- LibUniswapV3.sol#swapUniswapV3V1()
- LibUniswapV2.sol#swapUniswapV2()

Issue : Uniswap Swap Function with Insecure Deadline

Level : Medium

Remediation : To mitigate this front-running risk, it's recommended to use a more secure and deterministic deadline in your Uniswap swaps. One common approach is to set the deadline to a fixed point in the future, allowing for a reasonable execution window.

Alleviation / Retest : The contracts already have a higher level of validation for the deadline. Any attempt to exploit will fail the transaction due to that check.

Vulnerabilities

DESCRIPTION

The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

AFFECTED CODE

- MagpieCelerBridge.sol
- MagpieStargateBridge.sol
- MagpieStargateBridgeV2.sol

Issue : Use Ownable2Step

Level : Minor

Remediation : It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Alleviation / Retest : The contracts are now using Ownable2Step for additional security.

Vulnerabilities

DESCRIPTION

The contract uses the transfer and send functions to transfer funds to other addresses. This can be considered a potential security issue, primarily due to the following reason:

Gas Limitation: The transfer and send functions have a fixed gas stipend, which might not be adequate for the recipient's contract execution, especially if it contains complex logic or needs to perform multiple operations. This could result in a failed transfer.

AFFECTED CODE

- MagpieCelerBridge.sol#deposit()
- LibLayerZero.sol#dataTransfer()

Issue : Use of transfer and send Functions in Smart Contract

Level : Informational

Remediation : It's advisable to replace the transfer and send functions with the call method, which provides better control over gas and error handling. When using call, you can specify the gas amount to send with the call and check its return value to handle errors gracefully

Alleviation / Retest : This will create additional issues in the contract regarding gas estimation leading to loss of funds if not addressed properly. Since this is not an issue and is only “Informational”,

IN - DEPTH Vulnerabilities

DESCRIPTION

In Solidity, the visibility of state variables is important as it determines how those variables can be accessed and modified by other contracts or functions.
The contract defined state variables that were missing a visibility modifier.

AFFECTED CODE

- MagpieCelerBridge.sol#refundAddresses[mapping]

Issue : Missing State Variable Visibility

Level : Informational

Remediation : Explicitly define visibility for all state variables. These variables can be specified as public, internal, or private.

Alleviation / Retest : Variable visibility is now explicitly defined.

DESCRIPTION

The functions within the LibRouter library are currently defined as public. This means that these functions can be called from outside the library, including from other contracts and externally. Making functions public can be useful in certain scenarios. Public functions introduce additional gas costs due to the compiler-generated checks for msg.value and input data size. In cases where these checks are unnecessary, this can lead to suboptimal gas consumption.

AFFECTED CODE

- LibRouter.sol#getHopParams()

Issue : Public Function Visibility in LibRouter Library

Level : Gas

Remediation : It's recommended to review the functions within the LibRouter library and consider reducing their visibility to internal or private if they are not intended to be accessed from external contracts or users

Alleviation / Retest : The function has been made private in the library.

Vulnerabilities

DESCRIPTION

In the provided smart contract, there are internal functions declared but not used within the contract's functions or by any other contracts. These unused internal functions consume gas when deploying and executing the contract. Additionally, having unused functions can potentially confuse auditors and developers trying to understand the contract's logic.

AFFECTED CODE

- LibAsset.sol#getAllowance()
- LibAsset.sol#getBalanceOf()

Issue : Internal Function Never Used

Level : Gas

Remediation : To improve code quality and reduce unnecessary gas consumption, it's recommended to remove any internal functions that are not used within the contract or by any other contracts in the system

Alleviation / Retest : Unused functions are not removed from the code.

Vulnerabilities

DESCRIPTION

The contract uses for loops that use post increments for the variable “i”. The contract can save some gas by changing this to ++i.
++i costs less gas compared to i++ or i += 1 for unsigned integers. In i++, the compiler has to create a temporary variable to store the initial value. This is not the case with ++i in which the value is directly incremented and returned, thus, making it a cheaper alternative.

AFFECTED CODE

- LibMulticall.sol#multicall()

Issue : Gas Optimization in Increments

Level : Gas

Remediation : It is recommended to switch to ++i and change the code accordingly so the function logic remains the same and saves some gas.

Alleviation / Retest : The loop is still using i++.

Vulnerabilities

DESCRIPTION

During each iteration of the loop, reading the length of the array uses more gas than is necessary. In the most favorable scenario, in which the length is read from a memory variable, storing the array length in the stack can save about 3 gas per iteration. In the least favorable scenario, in which external calls are made during each iteration, the amount of gas wasted can be significant.

AFFECTED CODE

- LibRouter.sol#swap()
- LibMulticall.sol#multicall()

Issue : Array Length Caching

Level : Gas

Remediation : Consider storing the array length of the variable before the loop and use the stored length instead of fetching it in each iteration.

Alleviation / Retest : Array lengths are now cached before using them in loops.

DESCRIPTION

Loops are in most cases bounded by definition (the bounding is represented by the exit condition). Therefore in the vast majority of cases, checking for overflows is really not needed, and can get very gas-expensive.

AFFECTED CODE

- LibMulticall.sol#multicall()

Issue : Unnecessary Checked Arithmetic in Loops

Level : Gas

Remediation : It is recommended to implement unchecked blocks in for loops wherever possible since they are already bounded by an upper length and there's a very rare chance that it might overflow.

Alleviation / Retest : The loop has been made unchecked to save gas

SOURCE Code

Private GitHub Repository

REPORT Appendix

FINDING CATEGORIES

The assessment process will utilize a mixture of static analysis, dynamic analysis, in-depth manual review and/or other security techniques.

This report has been prepared for Magpie Protocol project using the above techniques to examine and discover vulnerabilities and safe coding practices in Magpie Protocol's smart contract including the libraries used by the contract that are not officially recognized.

A comprehensive static and dynamic analysis has been performed on the solidity code in order to find vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds.

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The testing methods find and flag issues related to gas optimizations that help in reducing the overall gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

AUDIT SCORES

D3ploy Audit Score is not a live dynamic score. It is a fixed value determined at the time of the report issuance date.

D3ploy Audit Score is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports and scores are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts d3ploy to perform a security review.

d3ploy

WEBSITE d3ploy.co

@d3ploy_ TWITTER