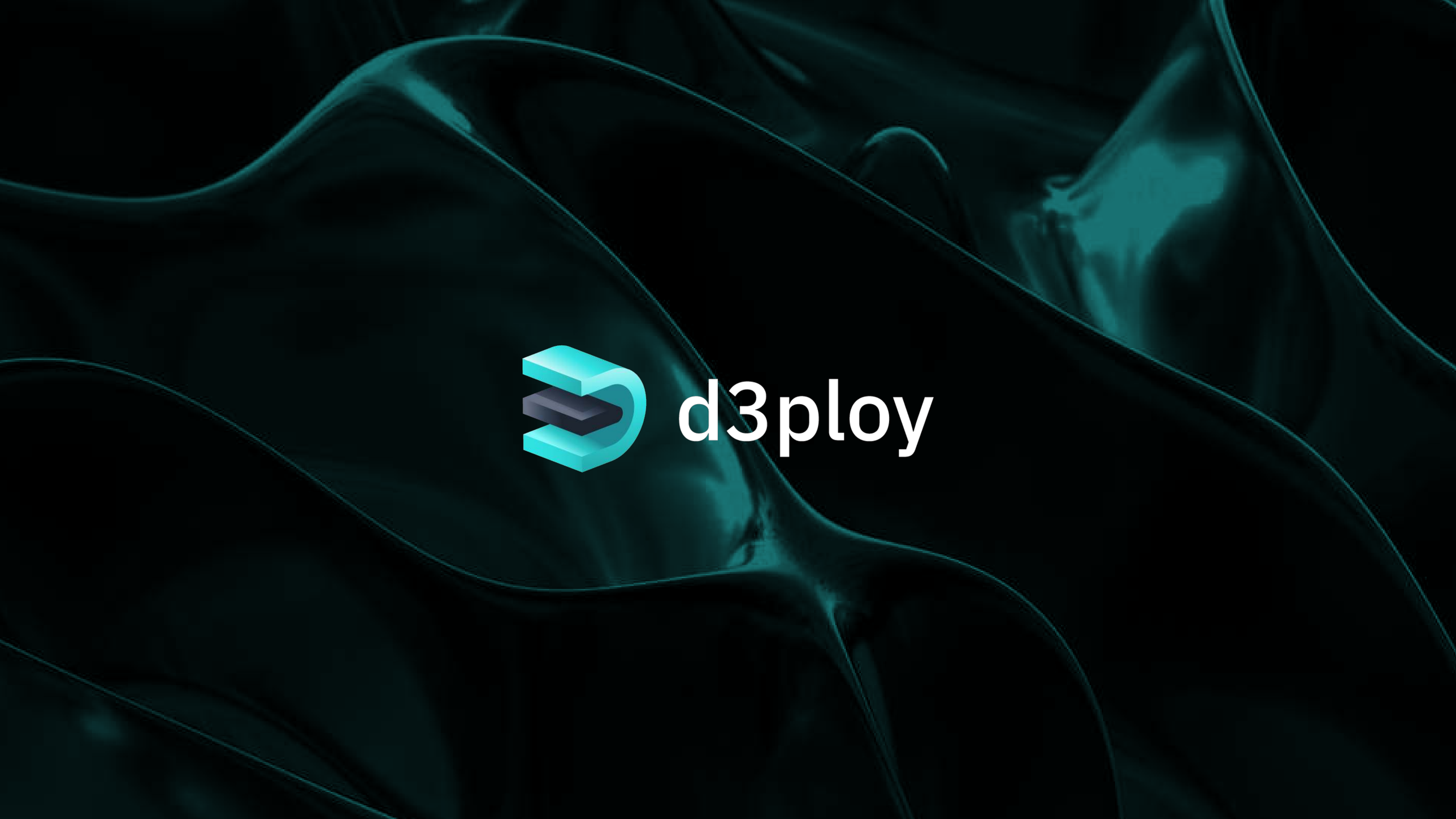
d3ploy

d3ploy

SECURITY ASSESSMENT

Magpie Protocol

February 05th 2024

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

96

PASS

AUDIT

Score

Issues	6
Critical	0
Major	1
Medium	1
Minor	1
Informational	1
Discussion	2

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 *2024/01/30*

Descrption *Layout project*

Architecture / Manual review / Static & dynamic security testing

Summary

Version *v1.1*

Date *2024/02/05*

Descrption *Re-audit applied fixes*

Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
Lack of Sender Validation in uniswapV3SwapCallback Function	✦ Major	Fixed
Lack of Consideration for Future Fee Implementation in wormhole's publishMessage	✦ Medium	Acknowledged
Outdated Pragma version	✦ Minor	Acknowledged
Missing NatSpec Comments	✦ Informational	Fixed
Cheaper Inequalities in if()	✦ Gas	Fixed
Cheaper Conditional Operators	✦ Gas	Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

In the provided contract, after executing a flash swap, the uniswapV3SwapCallback function is called, allowing the contract to return tokens to the initiator. However, there is no validation to ensure that the caller (msg.sender) is the Uniswap contract. This lack of validation can lead to a fund drain vulnerability where an attacker may impersonate the Uniswap contract and drain funds from the contract.

AFFECTED CODE

- MagpieRouterV2.sol #L31-33, L35-37, L39-L41

Issue : Lack of Sender Validation in uniswapV3SwapCallback Function

Level : Major

Remediation : To mitigate the fund drain vulnerability, it is recommended to add a validation check in the uniswapV3SwapCallback function to ensure that msg.sender is the legitimate Uniswap contract. This can be achieved by checking the address of msg.sender against the known Uniswap contract address.

Alleviation / Retest : There are no funds storing in the contract.

Vulnerabilities

DESCRIPTION

The dataTransfer function in the provided contract lacks consideration for potential future fees. Currently, the function publishMessage() of wormhole checks whether the msg.value matches the expected fee using require(msg.value == messageFee(), "invalid fee"). However, if the Wormhole protocol decides to introduce fees in the future, this condition would always fail, rendering the external calls within the protocol unable to perform their critical tasks.

Reference: wormhole#publishMessage():
https://github.com/wormhole-foundation/wormhole/blob/9bc408ca1912e7000c5c2085 215be9d44713028b/ethereum/contracts/Implementation.sol#L21

AFFECTED CODE

- LibWormhole.sol #L40-L44

Issue : Lack of Consideration for Future Fee Implementation in wormhole's publishMessage

Level : Medium

Remediation : To fix this issue it is recommended to pass msg.value while calling external call publishMessage() if any fee is applied in the future oi user can pay.

Alleviation / Retest : Magpie team commented that they don't consider this an issue 'Wormhole isn't charging fees for data transfer when you deliver it yourself. If Wormhole encounters issues in the future, they can simply disable it and use another protocol for data transfer, therefor it is easy to mitigate a potential issue.'

Vulnerabilities

DESCRIPTION

Using an outdated compiler version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version. The contracts found in the repository were allowing an old compiler version to be used, i.e., 0.8.17.

AFFECTED CODE

- All Contracts

Issue : Outdated Pragma version

Level : Minor

Remediation : Keep the compiler versions updated in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.22 pragma version which is stable and not too recent.
Reference: https://swcregistry.io/docs/SWC-103

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

Solidity contracts use a special form of comments to document code. This special form is named the Ethereum Natural Language Specification Format (NatSpec).
The document is divided into descriptions for developers and end-users along with the title and the author.
The contracts in the scope were missing these comments.

Issue : Missing NatSpec Comments

Level : Informational

Remediation : Developers should review their codebase and add Natspec comments to all relevant functions, variables, and events. Natspec comments should include a description of the function or event, its parameters, and its return values.

Alleviation / Retest : The comments has been added.

Vulnerabilities

DESCRIPTION

The contract was found to be doing comparisons using inequalities inside the “if” statement. When inside the “if” statements, non-strict inequalities (>=, <=) are usually cheaper than the strict equalities (>, <).

AFFECTED CODE

- MagpieRouterV2.sol #L26, L117

Issue : Cheaper Inequalities in if()

Level : Gas

Remediation : It is recommended to go through the code logic, and, if possible, modify the strict inequalities with the non-strict ones to save gas as long as the logic of the code is not affected.

Alleviation / Retest : The issue has been fixed.

Vulnerabilities

DESCRIPTION

Upon reviewing the code, it has been observed that the contract uses conditional statements involving comparisons with unsigned integer variables. Specifically, the contract employs the conditional operators x != 0 and x > 0 interchangeably. However, it's important to note that during compilation, x != 0 is generally more cost-effective than x > 0for unsigned integers within conditional statements.

AFFECTED CODE

- LibAssetV2.sol #L40

Issue : Cheaper Conditional Operators

Level : Gas

Remediation : Whenever possible, use the x != 0 conditional operator instead of x > 0 for unsigned integer variables in conditional statements.

Alleviation / Retest : The issue has been fixed.

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