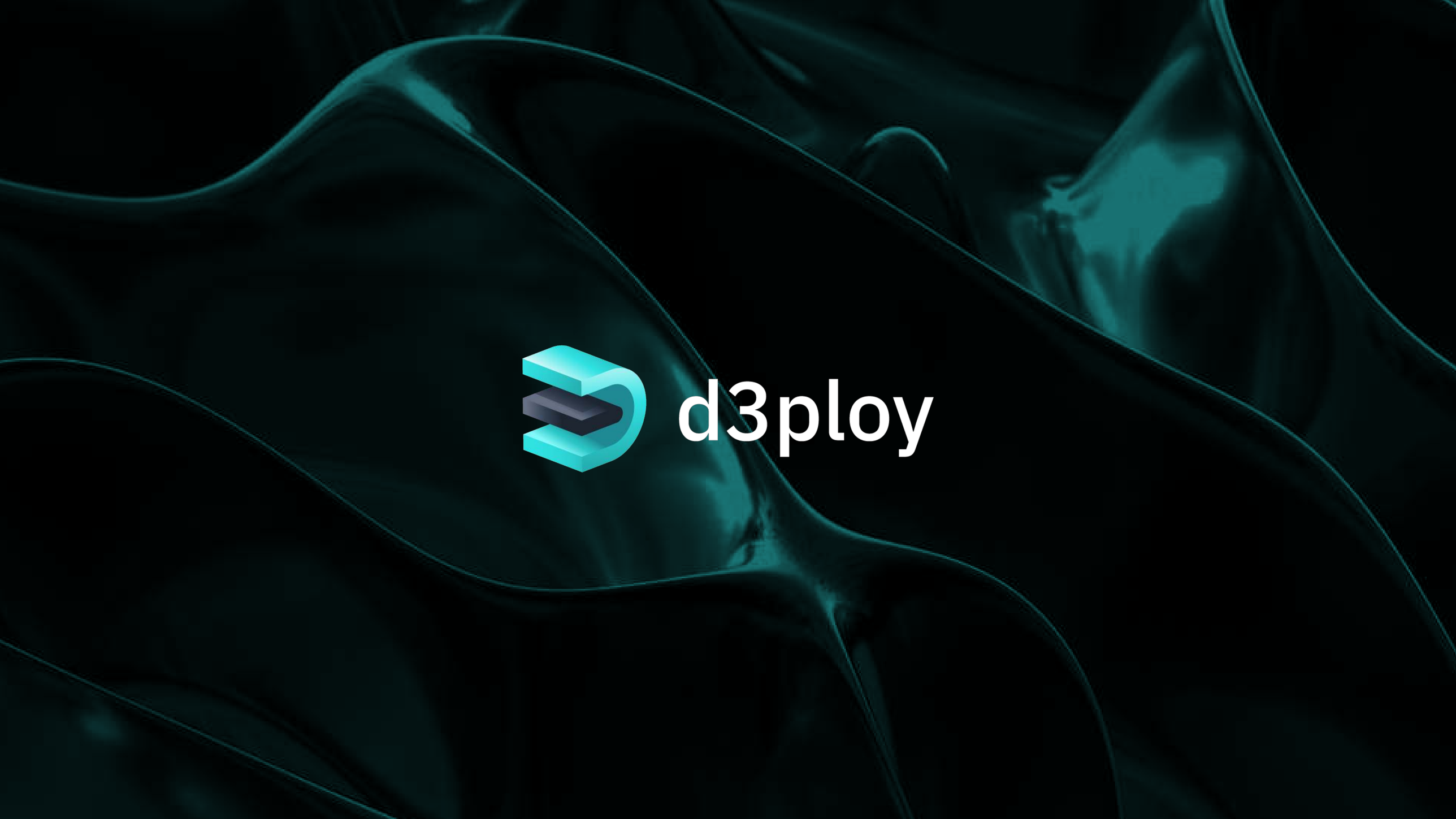
d3ploy

d3ploy

SECURITY ASSESSMENT

WarpGate

February 09th 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

WarpGate stands at the forefront of the Immutable zkEVM chain, a catalyst for the flourishing gaming ecosystem and decentralized economy. Their mission is to redefine the gaming experience, empower the community, and pave the way for a seamless fusion of gaming and decentralized finance.

A brief overview of WarpGate's product offering: Decentralized Exchange (DEX); Liquidity Pools; Launchpad with Decentralized Auctions; Initial Farm Offering (IFO); Yield Farming; Inter-Game Exchange (IGE).

Project Name WarpGate X
Contract Name WARP Token
Contract Address -
Contract Chain Not Yet Deployed on Mainnet
Contract Type Smart Contract
Platform EVM
Language Solidity
Network ImmutableX
Codebase Private GitHub Repository
Total Token Supply -

INFO

Social

https://warpgate.pro/

https://twitter.com/WarpGateX

https://t.me/WarpGateCommunity

https://discord.gg/warpgate

https://medium.com/@warpgate2024

https://github.com/WarpGate-Labs/

-

91

PASS

AUDIT

Score

Issues	13
Critical	0
Major	1
Medium	1
Minor	4
Informational	0
Discussion	7

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

AUDIT Scope

CODEBASE FILES	LOCATION
WarpGate-Labs/warpgate-contracts/projects/v3-core	✦ Private Repository
WarpGate-Labs/warpgate-contracts/projects/v3-lm-pool	✦ Private Repository
WarpGate-Labs/warpgate-contracts/projects/v3-periphery	✦ Private Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for WarpGate to discover issues and vulnerabilities in the source code of the WarpGate 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 *2024/02/07*

Descrption *Layout project*

Architecture / Manual review / Static & dynamic security testing

Summary

Version *v1.1*

Date *2023/02/09*

Descrption *Re-audit applied fixes*

Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
Initiating a front-running attack during pool initialization may result in depleting initial deposits made by liquidity providers	✦ Major	Acknowledged
Incomplete contract verification may lead to unnoticed transfer failures	✦ Medium	Acknowledged
Missing Two-Step Validation in Critical Address Change	✦ Minor	Acknowledged
Missing Zero Address Validations	✦ Minor	Acknowledged
Missing Same Address Validation in setOwner Function	✦ Minor	Acknowledged
Floating and Outdated Pragma	✦ Minor	Acknowledged
Public Constants can be Private	✦ Gas	Acknowledged
Large Number Literals	✦ Gas	Acknowledged
Multiplication/Division by 2 should use Bit-Shifting	✦ Gas	Acknowledged

KEY Finding

TITLE	SEVERITY	STATUS
Code Optimization by using max and min	✦ Gas	Acknowledged
Gas Optimization in Increments	✦ Gas	Acknowledged
Gas Optimization in Require Statements	✦ Gas	Acknowledged
Use of SafeMath	✦ Gas	Acknowledged

IN - DEPTH Vulnerabilities

DESCRIPTION

PancakeV3Pool.initialize is vulnerable to front-running, allowing attackers to manipulate prices unfairly and drain assets from initial deposits.
The absence of access controls on this function allows any entity to call it on a deployed pool. Initialization with an incorrect price enables attackers to generate profits from the initial liquidity provider's deposits.

AFFECTED CODE

- PancakeV3Pool.sol #L278-L306

Issue : Initiating a front-running attack during pool initialization may result in depleting initial deposits made by liquidity providers

Level : Major

Remediation : Introduce access controls to the initialize function. Transfer price operations from initialize to the constructor.

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

Due to the absence of contract existence verification in the pool, failed transactions involving tokens that have been destroyed may be incorrectly perceived as successful. The TransferHelper.safeTransfer function executes transfers through a low-level call without validating the existence of the contract. Consequently, if tokens are either not deployed or have been destroyed, the safeTransfer function might falsely indicate success without actually executing any transfer.

In scenarios where the token is not deployed, liquidity addition becomes impossible. On the other hand, if the token has been destroyed, the pool will incorrectly assume assets were sent even though no actual transfer occurred.

AFFECTED CODE

- v3-core/contracts/libraries/TransferHelper.sol #L1 4-L23

Issue : Incomplete contract verification may lead to unnoticed transfer failures

Level : Medium

Remediation : Add acontract existence check before the low-level call in TransferHelper.safeTransfer. This prevents the pool from accepting the sale of a token that no longer exists without returning any tokens in exchange.

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

Changing critical addresses in contracts should be a two-step process where the first transaction (from the old/current address) registers the new address (i.e. grants ownership) and the second transaction (from the new address) replaces the old address with the new one (i.e. claims ownership). This gives an opportunity to recover from incorrect addresses mistakenly used in the first step. If not, contract functionality might become inaccessible. The contract did not implement two-step validation when changing the owner address in the function “transferOwnership()”

AFFECTED CODE

- v3-core/contracts/PancakeV3Factory.sol #L36-L60: constructor()
- v3-core/contracts/PancakeV3Factory.sol #L82-L85: setOwner()

Issue : Missing Two-Step Validation in Critical Address Change

Level : Minor

Remediation : Enable a two-step process for critical address changes.

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

AFFECTED CODE

- v3-core/contracts/PancakeV3Factory.sol #L36-L60: constructor()
- v3-core/contracts/PancakeV3Factory.sol #L82-L85: setOwner()

Issue : Missing Zero Address Validations

Level : Minor

Remediation : Add azero address validation to all the functions where addresses are being set.

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

A function named setOwner allows the contract owner to be changed. However, it lacks a validation check to verify whether the newly specified owner (_owner) is the same as the current owner.
This omission raises a potential security concern, as it does not prevent unnecessary emissions of the OwnerChanged event when the new owner is identical to the existing one.

AFFECTED CODE

- v3-core/contracts/PancakeV3Factory.sol #L82-L85: setOwner()

Issue : Missing Same Address Validation in setOwner Function

Level : Minor

Remediation : Add a zero address validation to all the functions where addresses are being set.

Alleviation / Retest : The issue has been acknowledged.

DESCRIPTION

Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.
The contract allowed floating or unlocked pragma to be used.
This allows the contracts to be compiled with all the solidity compiler versions above the limit specified.

AFFECTED CODE

- All the contract files

Issue : Floating and Outdated Pragma

Level : Minor

Remediation : Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.22 pragma version. Since most of the code is developed for older versions, it is highly recommended to carefully consider this remediation because it may break the code or introduce other inconsistencies.

Reference: https://swcregistry.io/docs/SWC-103

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.

AFFECTED CODE

- v3-lm-pool/contracts/PancakeV3LmPool.sol #L22

Issue : Public Constants can be Private

Level : Gas

Remediation : If reading the values for the constants is not necessary, consider changing the public visibility to private.

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.

AFFECTED CODE

- v3-core/contracts/PancakeV3Factory.sol #L89
- v3-periphery/contracts/libraries/NFTDescriptor.sol #L264
- v3-periphery/contracts/NFTDescriptorEx.sol #L288

Issue : Large Number Literals

Level : Gas

Remediation : Scientific notation in the form of 2e10 is also supported, where the mantissa can be fractional, but the exponent has to be an integer. The literal MeE is equivalent to M * 10**E. Examples include 2e10, 2e10, 2e-10, 2.5e1, as suggested in official solidity documentation.
https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals
It is recommended to use numbers in the form “35 * 1e7 * 1e18” or “35 * 1e25”.
The numbers can also be represented by using underscores between them to make them more readable such as “35_00_00_000”

Alleviation / Retest : The issue has been acknowledged.

DESCRIPTION

In Solidity, the EVM (Ethereum Virtual Machine) executes operations in terms of gas consumption, where gas represents the computational cost of executing smart contract functions. Multiplication and division by two can be achieved using either traditional multiplication and division operations or bitwise left shift (<<) and right shift (>>) operations, respectively. However, using bit-shifting operations is more gas-efficient than using traditional multiplication and division operations.

- x * 2 can be replaced with x << 1.
- x / 2 can be replaced with x >> 1.

AFFECTED CODE

- v3-core/contracts/libraries/Oracle.sol #L164
- v3-periphery/contracts/libraries/NFTSVG.sol #L362

Issue : Multiplication/Division by 2 should use Bit-Shifting

Level : Gas

Remediation : It is recommended to use left and right shift instead of multiplying and dividing by 2 to save some gas.

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

In Solidity contract code, optimizing expressions involving powers of 2, such as 2**32, by using the built-in type(uint32).max and type(uint32).min constants can lead to improved code readability and gas efficiency. The original code utilizes 2**32 to calculate the maximum storage capacity of a uint32 data type, but this expression can be replaced with more expressive and gas-efficient alternatives.

AFFECTED CODE

- v3-core/contracts/libraries/Oracle.sol #L136-L137
- v3-core/contracts/libraries/SafeCast.sol #L25
- v3-periphery/contracts/NFTDescriptorEx.sol #L332
- v3-periphery/contracts/libraries/NFTDescriptor.sol #L308

Issue : Code Optimization by using max and min

Level : Gas

Remediation : To optimize code involving powers of 2, developers should replace expressions like 2**32 with type(uint32).max for maximum values and type(uint32).min for minimum values. It is essential to note that type(uint32).max is equivalent to 2**32- 1.

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

The contract uses two for loops, which 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

- v3-core/contracts/libraries/Oracle.sol #L118
- v3-core/contracts/libraries/Oracle.sol #L320
- v3-periphery/contracts/NonfungibleTokenPositio nDescriptor.sol #L46
- v3-periphery/contracts/NFTDescriptorEx.sol #L112
- v3-periphery/contracts/NFTDescriptorEx.sol #L243
- v3-periphery/contracts/lens/PancakeInterfaceMul ticall.sol #L30
- v3-periphery/contracts/lens/TickLens.sol #L23
- v3-periphery/contracts/lens/TickLens.sol #L30
- v3-periphery/contracts/base/Multicall.sol #L13
- v3-periphery/contracts/libraries/OracleLibrary.sol #L152
- v3-periphery/contracts/libraries/OracleLibrary.sol #L174
- v3-periphery/contracts/libraries/NFTDescriptor.sol #L88
- v3-periphery/contracts/libraries/NFTDescriptor.sol #L219

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 meanwhile saves some gas.

Alleviation / Retest : The issue has been acknowledged.

Vulnerabilities

DESCRIPTION

The require() statement takes an input string to show errors if the validation fails. The strings inside these functions that are longer than 32 bytes require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas. Once such example is given below:

AFFECTED CODE

- v3-periphery/contracts/NonfungiblePositionMana ger.sol #L391
- v3-periphery/contracts/base/ERC721Permit.sol #L74

Issue : Gas Optimization in Require Statements

Level : Gas

Remediation : It is recommended to go through all the require() statements present in the contract and shorten the strings passed inside them to fit under 32 bytes. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Alleviation / Retest : The issue has been acknowledged.

DESCRIPTION

SafeMath library is found to be used in the contract. This increases gas consumption more than traditional methods and validations if done manually.
Also, Solidity 0.8.0 and above includes checked arithmetic operations by default, and this renders SafeMath unnecessary.

AFFECTED CODE

- v3-periphery/contracts/NFTDescriptorEx.sol #L10
- v3-periphery/contracts/libraries/NFTDescriptor.sol #L10

Issue : Use of SafeMath

Level : Gas

Remediation : We do not recommend using the SafeMath library for all arithmetic operations. It is good practice to use explicit checks where it is really needed and to avoid extra checks where overflow/underflow is impossible.
It is recommended to upgrade to the latest compiler because versions above 0.8.0+ automatically check for overflows and underflows

Alleviation / Retest : The issue has been acknowledged.

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 WarpGate project using the above techniques to examine and discover vulnerabilities and safe coding practices in WarpGate'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