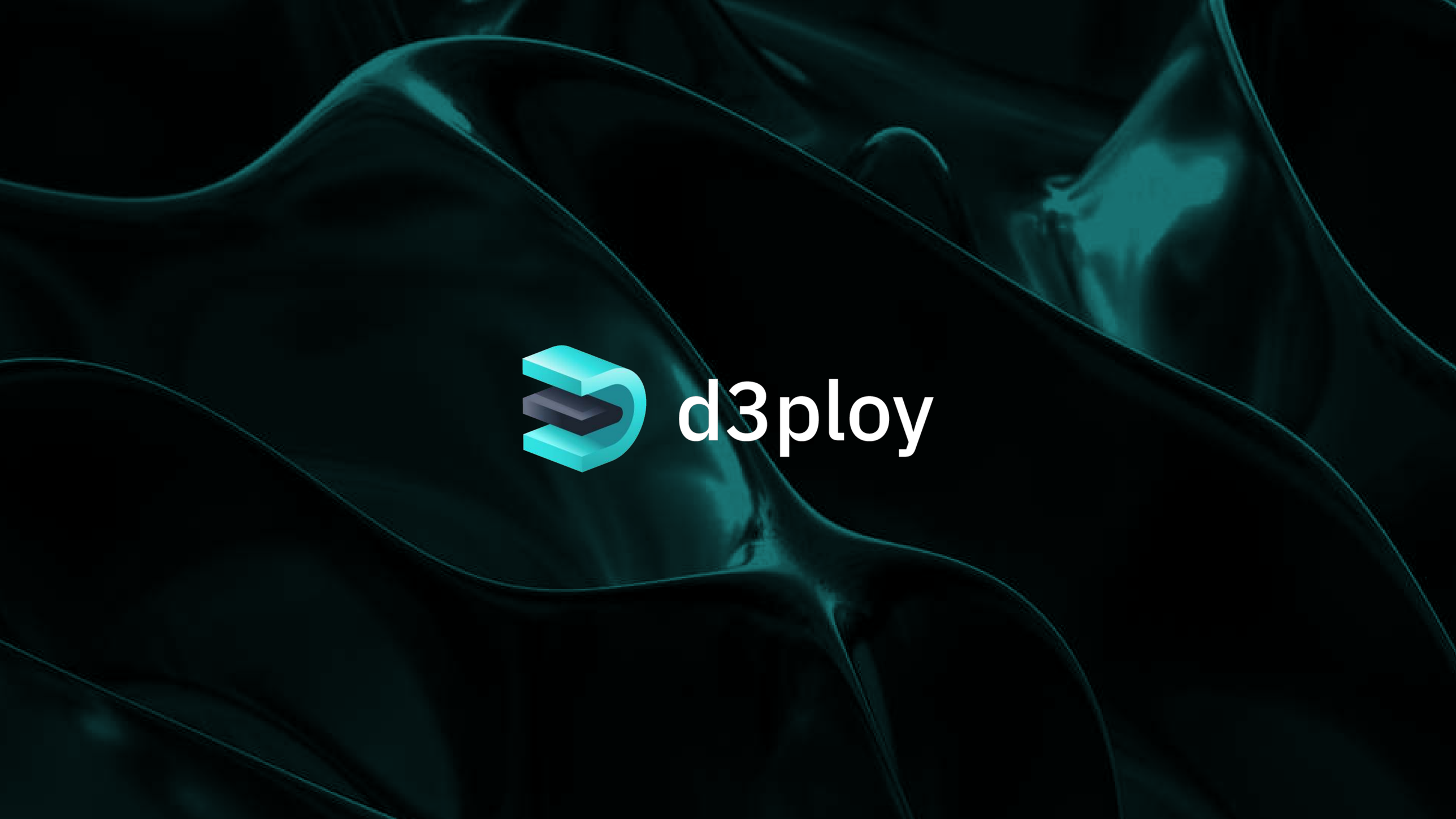
d3ploy

d3ploy

SECURITY ASSESSMENT

BSC News NFT

July 25th 2023

TABLE OF

Contents

01 Legal Disclaimer

02 D3ploy Intro

03 Project Summary

04 Audit Score

05 Audit Scope

06 Methodology

07 Key Findings

08 Vulnerabilities

09 Source Code

10 Appendix

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

BSC News is the leading media platform covering Decentralized Finance (DeFi) on BNB Chain covering a wide range of blockchain news revolving mainly around the DeFi sector of the crypto markets. BSC News aims to inform, educate and share information with the global investment community through our website, social media, newsletters, podcasts, research, and live ask me anything (AMA) sessions with top industry minds.

BSC News NFT - 10,000 premium access memberships that grant holders access to exclusive perks all around BNB Chain.

Project Name BSC News NFT

Contract Name -

Contract Address -

Contract Chain Not yet deployed on mainnet

Contract Type Smart Contract

Platform EVM

Language Solidity

Network BNB Chain (BEP20)

Codebase Private GitHub Repository

Total NFT Supply 10,000

INFO

Social

https://www.bsc.news/

https://twitter.com/BSCNews

https://t.me/BSCNewsAnn

https://discord.gg/bscnewsnft

-

-

info@bsc.news

AUDIT

Score

Issues	16
Critical	0
Major	3
Medium	0
Minor	2
Informational	2
Discussion	9

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

AUDIT Scope

CODEBASE FILES	LOCATION
BSCNewsNFTStaking.sol	✦ Raw Solidity File

REVIEW Methodology

TECHNIQUES

This report has been prepared for BSC News NFT to discover issues and vulnerabilities in the source code of the BSC News NFT 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 producedby 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 2023/07/12

Descrption Layout project

Architecture / Manual review / Static & dynamic security testing

Summary

Version v1.1

Date 2023/07/25

Descrption Re-audit addressed issues

Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
INSUFFICIENT CHECKS IN CLAIMREWARDS() FUNCTION	✦ Informational	Fixed
TOKENIDTOINDEX OF TOKEN IS NOT UPDATED IN WITHDRAW() FUNCTION	✦ Major	Fixed
CALLING CLAIMREWARDS() IS MORE PROFITABLE THAN CALLING WITHDRAWALL()	✦ Major	Fixed
GREATER REWARDS BY STAKING BEFORE REWARD PERIOD	✦ Major	Fixed
ARRAY LENGTH CACHING	✦ Gas	Fixed
CHEAPER CONDITIONAL OPERATORS	✦ Gas	Fixed
CHEAPER INEQUALITIES IN IF()	✦ Gas	Fixed
CHEAPER INEQUALITIES IN REQUIRE()	✦ Gas	Acknowledged
DEFINE CONSTRUCTOR AS PAYABLE	✦ Gas	Fixed

KEY Finding

TITLE	SEVERITY	STATUS
USE OF FLOATING PRAGMA	✦ Low	Fixed
GAS OPTIMIZATION FOR STATE VARIABLES	✦ Gas	Fixed
GAS OPTIMIZATION IN INCREMENTS	✦ Gas	Fixed
LONG REQUIRE/REVERT STRINGS	✦ Gas	Fixed
OUTDATED COMPILER VERSION	✦ Low	Fixed
STORAGE VARIABLE CACHING IN MEMORY	✦ Gas	Fixed
VARIABLES SHOULD BE IMMUTABLE	✦ Informational	Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

The claimRewards() function in the provided code does not perform any checks on the available rewards or the current balance of the contract. It directly transfers the reward tokens to the message sender without verifying if the contract has enough balance to cover the reward amount. Additionally, it does not validate if the reward amount stored in the rewards mapping for the sender is greater than zero before transferring the tokens.
Without the necessary checks, the function allows the sender to claim rewards even if the contract does not have sufficient balance to cover the reward amount. This can result in failed transactions or the loss of reward tokens if the contract runs out of balance during the claim process.

LOCATION

- BSCNewsNFTStaking.sol L90–99

Issue : INSUFFICIENT CHECKS IN CLAIMREWARDS() FUNCTION

Level : Informational

Remediation : To remediate this bug, it is important to add checks for both, the available rewards and the contract balance, before transferring the reward tokens to the sender.

Alleviation / Retest : Validation have been added.

IN - DEPTH Vulnerabilities

DESCRIPTION

In the withdraw() function, the index of the token which is being withdrawn is not set to zero or not deleted which will further create redundancy where tokenIdToIndex mapping will return the same index for two tokenIds. Suppose one token is withdrawn which has the last index let's say 5 and you can see tokenIdToIndex[tokenIds[I]] is not updated here. After this, suppose the same user stake another token which will be again indexed as 5.
When a token is withdrawn, its index in the tokenIdToIndex mapping should be set to zero or deleted. However, with this bug, the index is not updated, resulting in multiple tokens having the same index. This redundancy can cause confusion and potentially lead to incorrect data retrieval or processing.

LOCATION

- BSCNewsNFTStaking.sol L62-87

Issue : TOKENIDTOINDEX OF TOKEN IS NOT UPDATED IN WITHDRAW() FUNCTION

Level : Major

Remediation : Set the value of the mapping for the withdrawn token to zero or delete the entry altogether. You can delete the index of withdrawn tokens by using delete tokenIdToIndex[tokenIds[I]]; inside the withdraw function.

Alleviation / Retest : Fixed - using delete tokenIdToIndex[tokenId];

Vulnerabilities

DESCRIPTION

Calling the claimRewards() function separately may result in higher profitability compared to calling the withdrawAll() function. The reason for this is that the withdrawAll() function updates the lastUpdateTime variable to recent timeStamp, potentially leading to a decrease in the value of (lastTimeRewardApplicable() - lastUpdateTime) used in the rewardPerToken() function. By analyzing the transactions user can call the claimRewards() function when there is no transaction. Then lastUpdateTime will be storing old (small value) which will result in increasing (lastTimeRewardApplicable() - lastUpdateTime), which will eventually increase the reward.
The rewards per token may be underestimated, leading to a lower reward payout for users.

LOCATION

- BSCNewsNFTStaking.sol L90-99

Issue : CALLING CLAIMREWARDS() IS MORE PROFITABLE THAN CALLING WITHDRAWALL()

Level : Major

Remediation : To address this bug and ensure consistent and accurate reward calculations, it is recommended that Instead of updating the lastUpdateTime variable for every invocation of withdrawAll(), it should only be updated when a direct withdrawal (withdraw()) is made. This ensures that the rewards calculation is not affected by multiple updates of lastUpdateTime within the same transaction.

Alleviation / Retest : Withdrawall logic has been updated.

Vulnerabilities

DESCRIPTION

This issue allows users to exploit the timing of their staking activities to maximize their rewards. Staking tokens just before the reward period begins, and claiming rewards after the reward period starts, can result in higher rewards due to the way the rewardPerToken() and calculateRewards() functions are implemented. Suppose the user stake token just before the reward period starts (lastTimeRewardApplicable() - lastUpdateTime) will result in zero inside rewardPerToken() returned value of rewardPerToken() will be stored in userRewardPerTokenPaid[account] which will be less as compared to the normal value stored when the user tries to claim reward subtraction of rewardPerToken() - userRewardPerTokenPaid[_user] will result in a big number which will result in more reward calculation.

LOCATION

- BSCNewsNFTStaking.sol L44-58

Issue : GREATER REWARDS BY STAKING BEFORE REWARD PERIOD

Level : Major

Remediation : The bug allows users who stake their tokens just before the reward period begins to receive a disproportionately higher amount of rewards compared to other users who stake their tokens during the reward period.

Alleviation / Retest : Staking period status is now updated with OPEN and CLOSED during staking cycle.

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.
The following array was detected to be used inside loop without caching it's value in memory: userTokens.

LOCATION

- BSCNewsNFTStaking.sol L48-54; L66-83

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 : Fixed

Vulnerabilities

DESCRIPTION

During compilation, x != 0 is cheaper than x > 0 for unsigned integers in solidity inside conditional statements.

LOCATION

- BSCNewsNFTStaking.sol L92; L142; L143

Issue : CHEAPER CONDITIONAL OPERATORS

Level : Gas

Remediation : Consider using x != 0 in place of x > 0 in uint wherever possible.

Alleviation / Retest : X > 0 has been updated as X != 0 for uints.

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 (>, <).

LOCATION

- BSCNewsNFTStaking.sol L92

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 ~3 gas as long as the logic of the code is not affected.

Alleviation / Retest : Fixed

Vulnerabilities

DESCRIPTION

The contract was found to be performing comparisons using inequalities inside the require statement. When inside the require statements, non-strict inequalities (>=, <=) are usually costlier than strict equalities (>, <).

LOCATION

- BSCNewsNFTStaking.sol L156

Issue : CHEAPER INEQUALITIES IN REQUIRE()

Level : Gas

Remediation : It is recommended to go through the code logic, and, if possible, modify the non-strict inequalities with the strict ones to save ~3 gas as long as the logic of the code is not affected.

Alleviation / Retest : Acknowledged, gas related only.

Vulnerabilities

DESCRIPTION

Developers can save around 10 opcodes and some gas if the constructors are defined as payable.
However, it should be noted that it comes with risks because payable constructors can accept ETH during deployment.

LOCATION

- BSCNewsNFTStaking.sol L35-38

Issue : DEFINE CONSTRUCTOR AS PAYABLE

Level : Gas

Remediation : It is suggested to mark the constructors as payable to save some gas. Make sure it does not lead to any adverse effects in case an upgrade pattern is involved.

Alleviation / Retest : Constructor is now payable to save some gas

DESCRIPTION

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.
The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.The following affected files were found to be using floating pragma:
['BSCNewsNFTStaking.sol'] - ^0.8.0

LOCATION

- BSCNewsNFTStaking.sol L04

Issue : USE OF FLOATING PRAGMA

Level : Low

Remediation : It is recommended to use a fixed pragma version, as future compiler versions may handle certain language constructions in a way the developer did not foresee.Using a floating pragma may introduce several vulnerabilities if compiled with an older version.
The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.Instead of ^0.8.0 use pragma solidity 0.8.18, which is a stable and recommended version right now.

Alleviation / Retest : Pragma is strict.

Vulnerabilities

DESCRIPTION

Plus equals (+=) costs more gas than addition operator. The same thing happens with minus equals (-=). Therefore, x +=y costs more gas than x = x + y.

LOCATION

- BSCNewsNFTStaking.sol L55; L84

Issue : GAS OPTIMIZATION FOR STATE VARIABLES

Level : Gas

Remediation : Consider

- addition operator over plus equals
- subtraction operator over minus equals
- division operator over divide equals
- multiplication operator over multiply equals

Alleviation / Retest : x += y has been updated as x = x+y

Vulnerabilities

DESCRIPTION

++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.

LOCATION

- BSCNewsNFTStaking.sol L48; L66

Issue : GAS OPTIMIZATION IN INCREMENTS

Level : Gas

Remediation : Consider changing the post-increments (i++) to pre-increments (++i) as long as the value is not used in any calculations or inside returns. Make sure that the logic of the code is not changed.

Alleviation / Retest : Loop counter has been updated to save gas.

Vulnerabilities

DESCRIPTION

The require() and revert() functions take an input string to show errors if the validation fails.
This 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.

LOCATION

- BSCNewsNFTStaking.sol L142; L143: L144-147; L156

Issue : LONG REQUIRE/REVERT STRINGS

Level : Gas

Remediation : It is recommended to short the strings passed inside require() and revert() 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 : require strings have been shortened.

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 following outdated versions were detected:
['BSCNewsNFTStaking.sol'] - ^0.8.0

LOCATION

- BSCNewsNFTStaking.sol L04

Issue : OUTDATED COMPILER VERSION

Level : Low

Remediation : It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version 0.8.18, which patches most solidity vulnerabilities.

Alleviation / Retest : Compiler version is updated.

Vulnerabilities

DESCRIPTION

The contract BSCNewsNFTStaking is using the state variables multiple times in functions. SLOADs are expensive (100 gas after the 1st one) compared to MLOAD/MSTORE (3 gas each).

LOCATION

- **BSCNewsNFTStaking.sol** tokensStaked / **function** stake L30
- **BSCNewsNFTStaking.sol** stakedAssets / **function** withdraw L29
- **BSCNewsNFTStaking.sol** tokenIdToIndex / **function** withdraw L31
- **BSCNewsNFTStaking.sol** rewards / **function** claimRewards L27
- **BSCNewsNFTStaking.sol** periodFinish / **function** startStakingPeriod L18
- **BSCNewsNFTStaking.sol** rewardRate / **function** startStakingPeriod L19
- **BSCNewsNFTStaking.sol** rewardsDuration / **function** startStakingPeriod L20
- **BSCNewsNFTStaking.sol** periodFinish / **function** lastTimeRewardApplicable L18
- **BSCNewsNFTStaking.sol** rewardPerTokenStored / **function** rewardPerToken L22

Issue : STORAGE VARIABLE CACHING IN MEMORY

Level : Gas

Remediation : Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 SLOAD) and then read from this cache to avoid multiple SLOADs.

Alleviation / Retest : Fixed.

Vulnerabilities

DESCRIPTION

Constants and Immutables should be used in their appropriate contexts. constant should only be used for literal values written into the code. immutable variables should be used for expressions, or values calculated in, or passed into the constructor.

LOCATION

- BSCNewsNFTStaking.sol L15; L16

Issue : VARIABLES SHOULD BE IMMUTABLE

Level : Informational

Remediation : It is recommended to use immutable instead of constant.

Alleviation / Retest : Fixed.

SOURCE Code

Raw Solidity File

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 BSC News NFT project using the above techniques to examine and discover vulnerabilities and safe coding practices in BSC News NFT'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