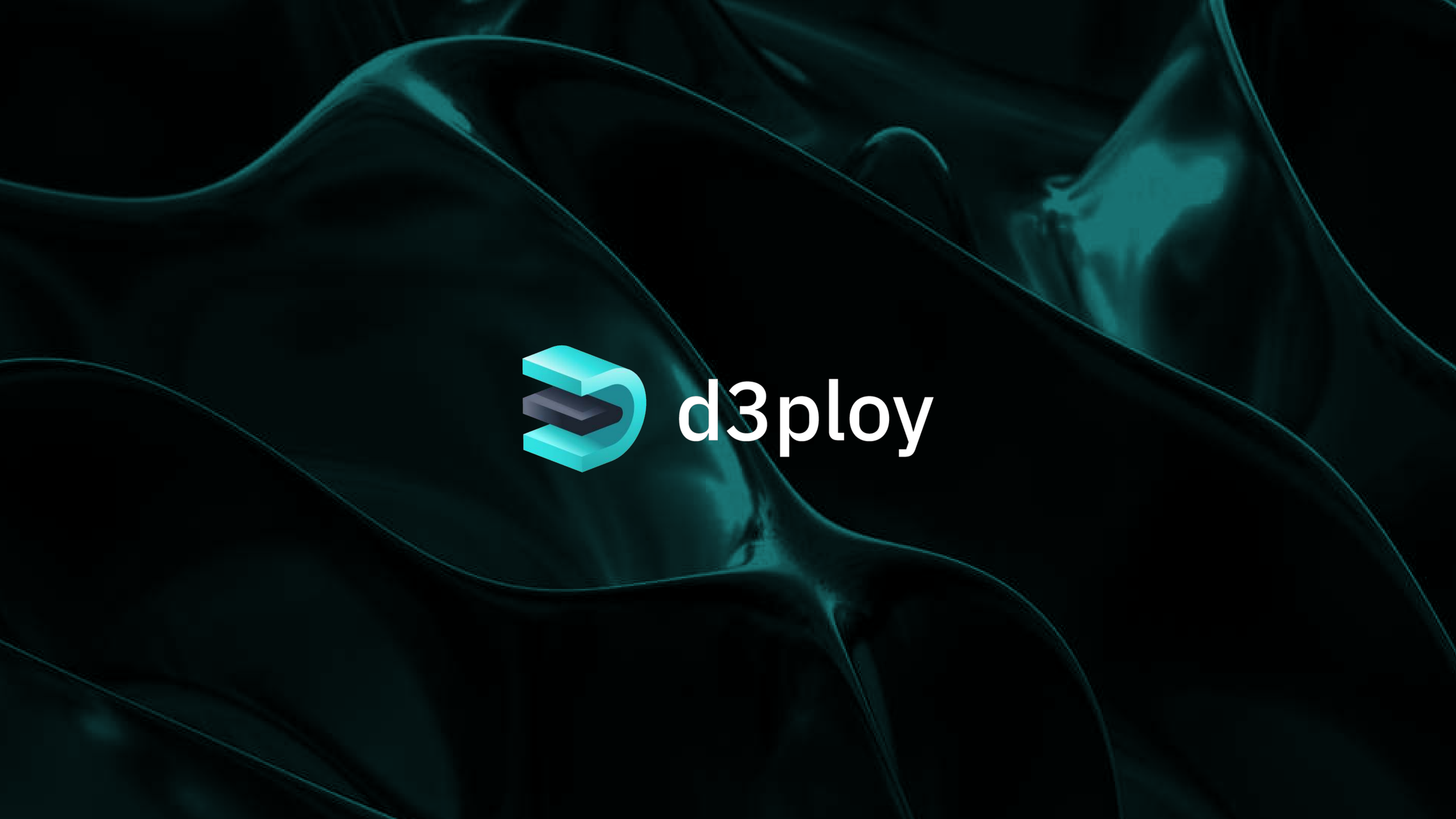
d3ploy

d3ploy

SECURITY ASSESSMENT

PRINT3R

September 01st 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

PRINT3R aims to shake-up the existing derivatives market by fully gamifying the trading experience, creating an immersive and addictive experience for traders at any level.

PRINT3R is a gamified decentralized derivatives platform that focuses on ease of use and encouraging user loyalty through engaging user interactions, and sharing revenue. Trade liquid crypto assets at up to 50x leverage without the fuss of KYC or sign-up. Just connect a wallet, and you're in. The platform guarantees no price impact on longs, shorts, and swaps, all at minimal fees.
Derivatives are a cornerstone in the crypto realm. PRINT3R's ambition is to elevate this model, merging key success elements from each aspect of the web3 landscape.

Project Name PRINT3R

Contract Name PRINT Token

Contract Address -

Contract Chain Not Yet Deployed on Mainnet

Contract Type Smart Contract

Platform EVM

Language Solidity

Network Base Chain (BASE-69)

Codebase Private GitHub Repository

Max Supply 8, 121, 212

INFO

Social

https://www.print3r.xyz/

https://twitter.com/PRINT3Rxyz

https://t.me/print3rXYZ

https://discord.gg/uPZf5eqaSZ

https://print3r.gitbook.io/print3r/

https://github.com/PRINT3Rxyz

-

96

PASS

AUDIT

Score

Issues	13
Critical	0
Major	1
Medium	1
Minor	3
Informational	2
Discussion	6

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

AUDIT Scope

GITHUB REPOSITORY

PRINT3Rxyz/print3r-contracts

LOCATION

- GitHub Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for Print3r.xyz to discover issues and vulnerabilities in the source code of the Print3r 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 2023/08/30

Descrption Layout project

Architecture / Manual review / Static & dynamic security testing

Summary

Version v1.1

Date 2023/09/01

Descrption Reaudit addressed vulnerabilities

Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
Using EXTCODESIZE To Check For Externally Owned Accounts	✦ Informational	Acknowledged
Superfluous Event Fields	✦ Gas	Acknowledged
Unused Imports	✦ Gas	Fixed
Cheaper Conditional Operators	✦ Gas	Fixed
Array Length Caching	✦ Gas	Fixed
Unnecessary Checked Arithmetic In Loop	✦ Gas	Fixed
Custom Errors instead of Revert	✦ Gas	Fixed
Require With Empty Message	✦ Informational	Fixed
Missing Zero Address Validations	✦ Minor	Acknowledged
Empty Catch Block	✦ Minor	Fixed

KEY Finding

TITLE	SEVERITY	STATUS
Missing Zero Address Validations	✦ Major	Fixed
Failure to Delete increasePositionRequestKeys after Execution	✦ Medium	Acknowledged
Missing Zero Value Validations	✦ Minor	Acknowledged

IN - DEPTH Vulnerabilities

DESCRIPTION

Upon reviewing the code, it has come to attention that the extcodesize opcode is used to determine whether an account is an externally owned account or another contract. While extcodesize typically returns 0 for externally owned accounts, there is an important consideration regarding its behavior during contract deployment or constructor execution. Specifically, when a contract is under construction or its constructor is running, extcodesize for the contract's address returns zero. This behavior can lead to inaccurate results when attempting to identify externally owned accounts during these specific circumstances.

AFFECTED CODE

- /src/core/PositionRouter.sol #L836

IMPACTS

During contract deployment or constructor execution, the extcodesize check may incorrectly identify the account as externally owned due to the opcode's behavior returning zero.

Issue : Using EXTCODESIZE To Check For Externally Owned Accounts

Level : Informational

Type : Misconfiguration

Remediation : To accurately identify externally owned accounts, consider using alternative methods or checks that are not affected by the behavior of extcodesize during contract deployment or constructor execution.

Alleviation / Retest : Print3r team acknowledged the issue.

IN - DEPTH Vulnerabilities

DESCRIPTION

Upon reviewing the code, it has been identified that certain events include fields for block.timestamp and block.number. These fields are automatically added to event information by default. However, manually adding them can lead to unnecessary gas consumption. To optimize gas usage and reduce transaction costs, it is recommended to avoid including these superfluous event fields.

AFFECTED CODE

- /src/core/PositionRouter.sol #L725-L740

IMPACTS

Including superfluous event fields, such as block.timestamp and block.number, in events can have implications on the contract's gas consumption and efficiency

Issue : Superfluous Event Fields

Level : Gas

Type : Gas Optimization

Remediation : Remove any manually added event fields that duplicate block.timestamp or block.number

Alleviation / Retest : Print3r team acknowledged the issue.

Vulnerabilities

DESCRIPTION

The contract PositionRouter.sol was importing contracts ITimelock.sol & IVault.sol which was not used anywhere in the code. This increases the gas cost and overall contract's complexity.

AFFECTED CODE

- /src/core/PositionRouter.sol #L07, L13

IMPACTS

Unused imports in smart contracts can lead to an increase in the size of the code, making it more difficult to verify and potentially slowing down its execution. Moreover, having unused code in a smart contract can also increase the attack surface by potentially introducing vulnerabilities that can be exploited by malicious actors. This can lead to security issues and compromise the integrity of the contract. Additionally, including unused imports in smart contracts can also increase deployment and gas costs, making it more expensive to deploy and run the contract on the Ethereum network.

Issue : Unused Imports

Level : Gas

Type : Gas Optimization

Remediation : It is recommended to remove the import statement if the external contracts or libraries are not used anywhere in the contract.

Alleviation / Retest : Fixed. Unused imports have been removed.

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

- /src/staking/BrrrXpAmplifier.sol #L209, L223
- /src/core/PositionRouter.sol #L336, L452, L562

IMPACTS

Employing x != 0 in conditional statements can result in reduced gas consumption compared to using x > 0. This optimization contributes to cost-effectiveness in contract interactions.

Issue : Cheaper Conditional Operators

Level : Gas

Type : Gas Optimization

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

Alleviation / Retest : Fixed. Conditional operators have been adjusted to != 0 to save gas.

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

- /src/staking/BrrrXpAmplifier.sol #L135, L269

IMPACTS

Reading the length of an array multiple times in a loop by calling .length costs more gas.

Issue : Array Length Caching

Level : Gas

Type : Gas Optimization

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. Array length is now cached before using.

Vulnerabilities

DESCRIPTION

Upon reviewing the code, it has been identified that the contract uses checked arithmetic operations inside loops where increments occur. However, it's important to note that increments inside loops are unlikely to cause overflow since the transaction will run out of gas before the variable reaches its limits. As a result, using checked arithmetic for increments within loops may be unnecessary and can lead to additional gas consumption.

AFFECTED CODE

- /main/src/staking/BrrrXpA mplifier.sol #L135, L269

IMPACTS

Unnecessary checked arithmetic operations can lead to higher gas consumption, as each arithmetic operation comes with its own gas cost. This can contribute to increased transaction fees and operational costs.

Issue : Unnecessary Checked Arithmetic In Loop

Level : Gas

Type : Gas Optimization

Remediation : Consider having the increment value inside the unchecked block to save some gas.

Alleviation / Retest : Fixed. Loops have been unchecked.

Vulnerabilities

DESCRIPTION

The contract was found to be using revert() statements in multiple places. Since Solidity v0.8.4, custom errors have been introduced which are a better alternative to the revert. This allows the developers to pass custom errors with dynamic data while reverting the transaction and also makes the whole implementation a bit cheaper than using revert.

VULNERABLE CODE

- /src/core/PositionRouter.sol #L659, L681

IMPACTS

Using revert() instead of error() costs more gas.

Issue : Custom Errors instead of Revert

Level : Gas

Type : Gas Optimization

Remediation : It is recommended to replace the instances of revert() statements with error() to save gas.

Alleviation / Retest : Fixed. Custom errors are implemented.

Vulnerabilities

DESCRIPTION

During code analysis, it has been observed that some require statements lack descriptive messages, which provide crucial information to users when conditions are not met. These messages, limited to 32 bytes, improve user understanding of why a transaction was reverted.

VULNERABLE CODE

- /src/staking/BrrrXpAmplifier.sol #L67

IMPACTS

Users may be left without clear context when a transaction is reverted due to unmet conditions, leading to confusion and frustration

Issue : Require With Empty Message

Level : Informational

Type : Gas Optimization

Remediation : Add concise, informative messages to require statements, explaining why the condition failed. Ensure messages are clear and within the 32-byte limit

Alleviation / Retest : Fixed

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

- /src/core/PositionRouter.sol #L195, L205-206, L317, L390

IMPACTS

If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Issue : Missing Zero Address Validations

Level : Minor

Type : Missing Input Validation

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

Alleviation / Retest : Print3r team acknowledged the issue.

Vulnerabilities

DESCRIPTION

The contract PositionRouter._callRequestCallback() is making a call to gmxPositionCallback() inside a try/catch block. However, the catch block is left empty and is not doing anything. This shows a missing error handling.

AFFECTED CODE

- /src/core/PositionRouter.sol #L857

IMPACTS

Missing error handling could go unnoticed and the user won't know if the function executed properly or not.

Issue : Empty Catch Block

Level : Minor

Type : Missing Error Handling

Remediation : It is recommended to handle all errors properly inside the catch statement and emit an event for the failed reason as well if needed.

Alleviation / Retest : Fixed. The catch block now reverts.

IN - DEPTH Vulnerabilities

DESCRIPTION

Upon a comprehensive assessment of the contract's code, a potential scenario leading to the loss of rewards has been identified within the claimPendingRewards() function. The function checks if the available balance of a specific token is equal to the user's claimable token rewards, it does not include the mechanism to revert if the available balance is less than userTokenRewards.

VULNERABLE CODE

- /src/staking/BrrrXpAmplifier.sol #L149-L161

IMPACTS

Users will lose their rewards if userTokenRewards is not available in the contract while calling claimPendingRewards()

Issue : Missing Zero Address Validations

Level : Major

Type : Missing Validation

Remediation : Apply mechanism to revert if contract balance is less than userTokenRewards.

Alleviation / Retest : Fixed. Validation has been added.

Vulnerabilities

DESCRIPTION

The function PositionRouter.executeIncreasePositions() executes all the positions in batch inside a loop and inside every execution it deletes the index value for increasePositionRequestKeys. However, the deletion part is missing inside the function PositionRouter.executeIncreasePosition(). This will leave residual values inside the mapping if the user tries to execute single positions. This also affects the decreasePositionRequestKeys mapping.

AFFECTED CODE

- /src/staking/BrrrXpAmplifier.sol #L149-L161

IMPACTS

Failure to delete the mapping after execution could leave residual results after executing the mapping. This would get executed again if the user tries to execute them in batch creating inconsistencies and incorrect calculations.

Issue : Failure to Delete increasePositionRequestKeys after Execution

Level : Medium

Type : Business Logic

Remediation : It is recommended to delete the mappings in both the batch and single execution functions.

Alleviation / Retest : Print3r team acknowledged the issue.

Vulnerabilities

DESCRIPTION

Upon a thorough examination of the contract's code, it has come to attention that the createIncreasePosition() function lacks validation to ensure that specific input values, such as _minOut, are greater than zero. This absence of validation can potentially lead to unintended behavior and errors during the function's execution.

AFFECTED CODE

- /main/src/core/PositionRouter.sol #L317-L353

IMPACTS

Input values that are not greater than zero can result in unexpected and inaccurate behavior during the execution of the function. This can result in a Sandwich Attack

Issue : Missing Zero Value Validations

Level : Minor

Type : Missing Input Validation

Remediation : Apply mechanism to check whether is _minOut greater than zero.

Alleviation / Retest : Print3r team acknowledged the issue.

SOURCE Code

GitHub Repository

PRINT3Rxyz/print3r-contracts

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 Print3r project using the above techniques to examine and discover vulnerabilities and safe coding practices in Print3r'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