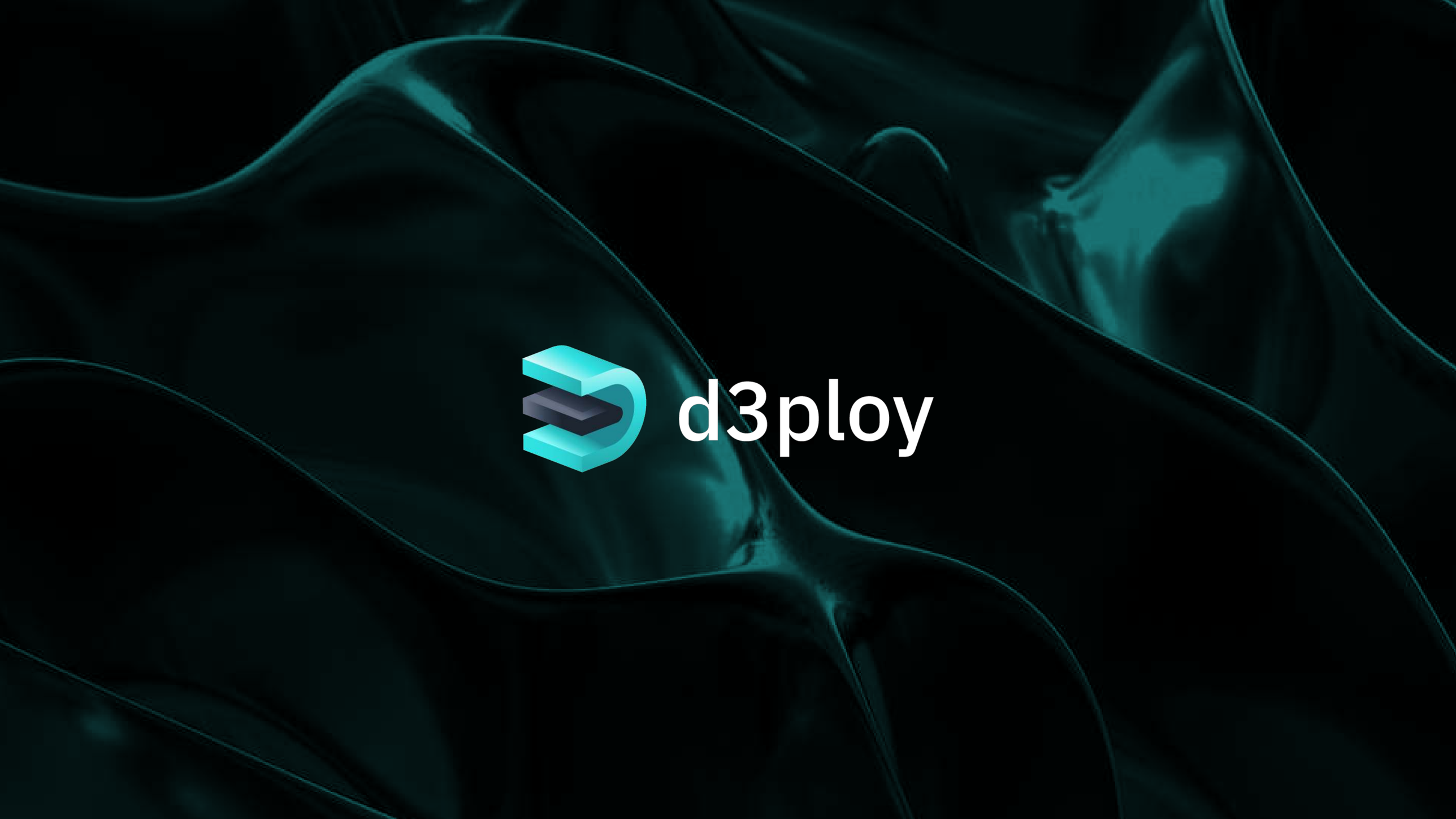
d3ploy

d3ploy

SECURITY ASSESSMENT

ASX

October 25th 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

Providing exposure to multiple asset classes including RWA's, Defi, and early stage venture capital investments through a single yield bearing token.

The ASX Ecosystem is being developed with a focus on asset diversification and ease of access for investors. The ecosystem will consist of a number of funds, beginning with the Flagship Fund, a multi-asset fund consisting of real world assets (RWAs), yield bearing defi positions, and investments in early stage venture capital deals. The Flagship Fund will work in tandem with the ecosystem token ($ASX), via both growth and yield distribution mechanics.

Project Name ASX

Contract Name ASX Token

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

Max Supply 10,000,000

INFO

Social

https://www.asx.capital/

https://twitter.com/asx_capital

https://t.me/ASXOfficial

-

https://docs.asx.capital/

-

-

AUDIT

Score

Issues	16
Critical	0
Major	1
Medium	2
Minor	1
Informational	3
Discussion	9

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

AUDIT Scope

TESTNET EXPLORER

goerli.etherscan.io/

LOCATION

- 0x543956d87CdB2566f498106eB7774a2B1eb58A91#code

REVIEW Methodology

TECHNIQUES

This report has been prepared for ASX to discover issues and vulnerabilities in the source code of the ASX 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 2023/10/19

Descrption Layout project

Architecture / Manual review / Static & dynamic security testing

Summary

Version v1.1

Date 2023/10/25

Descrption Reaudit addressed vulnerabilities

Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
Fee Bypass via Multi-Step Transaction	✦ Major	Acknowledged
Zero Slippage Value	✦ Medium	Acknowledged
Uniswap Swap Function with Insecure Deadline	✦ Medium	Acknowledged
Missing Zero Address Validations	✦ Minor	Fixed
Functions Should be Declared External	✦ Informational	Fixed
Use Ownable2Step	✦ Informational	Partially Fixed
Missing NatSpec Comments	✦ Informational	Acknowledged
Array Length Caching	✦ Gas	Fixed
Gas Optimization in Increments	✦ Gas	Fixed
Unnecessary Checked Arithmetic in Loops	✦ Gas	Fixed

KEY Finding

TITLE	SEVERITY	STATUS
Redundant Value Setting	✦ Gas	Fixed
Large Number Literals	✦ Gas	Fixed
Public Constants can be Private	✦ Gas	Fixed
Use of SafeMath	✦ Gas	Acknowledged
Variables should be Immutable	✦ Gas	Fixed
Unnecessary Default Value Initialization	✦ Gas	Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

The smart contract includes a fee system for token transfers. However, it can be bypassed by utilizing a multi-step transaction involving three users: A, B, and C. User A and User B are normal users and User C is listed in _isExcludedFromFees. Here's a scenario of how the fee can be skipped:

1. User A intends to send tokens to User B but wishes to avoid paying the associated fees.
2. User A first transfers the tokens to User C (an intermediary). No fee will be added since C is listed in _isExcludedFromFees.
3. User C then sends the tokens to User B, effectively bypassing the fee that would have been applied if User A directly transferred tokens to User B.

By following this sequence of transactions, User A can successfully skip paying the fees that are intended to be applied to token transfers. This scenario exploits the condition in the contract that excludes fees when either the sender or the recipient is on the list of accounts excluded from fees (_isExcludedFromFees).

AFFECTED CODE

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L475

Issue : Fee Bypass via Multi-Step Transaction

Level : Major

Type : Business Logic

Remediation : To address this issue, it is recommended to use only one between to or from while validating _isExcludedFromFees, or you can apply any other mechanisms that can prevent this Multi-step transfer.

Alleviation / Retest : According to the client this is intentional behavior as the user is trusted.

Vulnerabilities

DESCRIPTION

In some of the functions, a token swap operation and adding liquidity is performed with a hardcoded minAmt (slippage) value set to zero. Slippage refers to the maximum acceptable difference between the expected price of an asset and the actual executed price during a swap. A slippage of zero means that the code expects the swap to return a value near 0.

AFFECTED CODE

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L587, L615, L634, L654

Issue : Zero Slippage value

Level : Medium

Type : Slippage Risk

Remediation : To make the token swap function more robust and adaptable to market conditions, it is recommended to set a non-zero slippage tolerance (e.g., a small percentage) rather than a hardcoded zero value. This will allow the code to accommodate minor price fluctuations and ensure that the swap is more likely to succeed. Or take input from the user to set minAmt.

Alleviation / Retest : This won't be fixed as it will need external price oracles like Chainlink or Uniswap Twap which will result in large codebase updates. Nonetheless, there are management features that allow admin to turn swaps or liquidity features on or off so the situation can be controlled.

IN - DEPTH 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

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L615, L634, L654

Issue : Uniswap Swap Function with Insecure Deadline

Level : Medium

Type : Front-Running

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 : In the current contract, the swap is executed in ERC20 common transfer function that must have only standard parameters not including a deadline. User transfer transactions can't have any other parameters like deadline.

IN - DEPTH 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

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L264, L283, L344

Issue : Cheaper Conditional Operators

Level : Minor

Type : Missing Input Validation

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

Alleviation / Retest : Zero address validations have been implemented in the functions mentioned above.

Vulnerabilities

DESCRIPTION

Public functions that are never called by a contract should be declared external in order to conserve gas.
The following functions were declared as public but were not called anywhere in the contract, making public visibility useless

AFFECTED CODE

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L192, L212, L226, L292, L309

Issue : Functions should be declared External

Level : Informational

Type : Best Practices

Remediation : Use the “external” state visibility for functions that are never called from inside the contract.

Alleviation / Retest : The functions that are not called anywhere inside the contract have been made external.

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

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L08, L39, L72

Issue : Use Ownable2Step

Level : Informational

Type : Missing Best Practices

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

Alleviation / Retest : The AssetX.sol contract has implemented the 2step flow but others have not.

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

Type : Missing Best Practices

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 :

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

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L227, L350

Issue : Array Length Caching

Level : Discussion

Type : Gas Optimization

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

Alleviation / Retest : Array length is being cached to save gas during each iteration.

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

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L227, L350

Issue : Gas Optimization in Increments

Level : Discussion

Type : Gas Optimization

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 loops are now using ++i instead to save gas.

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

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L227, L350

Issue : Unnecessary Checked Arithmetic in Loops

Level : Discussion

Type : Gas Optimization

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 : Unchecked loops have been implemented to save gas.

Vulnerabilities

DESCRIPTION

The smart contract has functions named excludeMultipleAccountsFromFees & whitelistMultipleAccounts. This function is designed to exclude or include & whitelist multiple accounts from fees based on the provided input. However, the function does not verify whether accounts that are already excluded or whitelisted from fees are being added again. This can lead to inefficient use of gas and confusion within the contract's state.

AFFECTED CODE

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L226, L349

Issue : Redundant Value setting

Level : Discussion

Type : Gas & Missing Best Practices

Remediation : To address this issue, it's recommended to add checks to ensure that accounts are not redundantly excluded from or included in fees. Before setting an account as excluded or included, whitelist.

Alleviation / Retest : Proper validations have been added to check for redundant value updates.

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

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L70, L310

Issue : Large Number Literals

Level : Discussion

Type : Gas & Missing Best Practices

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 numbers are now using a more readable format with underscores.

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

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L48

Issue : Public Constants can be Private

Level : Discussion

Type : Gas Optimization

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

Alleviation / Retest : The variable has been updated to private visibility.

Vulnerabilities

DESCRIPTION

SafeMath library is found to be used in the contracts. 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, rendering SafeMath unnecessary

AFFECTED CODE

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L09, L40, L73

Issue : Use of SafeMath

Level : Discussion

Type : Gas Optimization

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.
The compiler above 0.8.0+ automatically checks for overflows and underflows.

Alleviation / Retest : Print3r team acknowledged the issue.

DESCRIPTION

Declaring state variables that are not updated following deployment as immutable can save gas costs in smart contract deployments and function executions. Immutable state variables are those that cannot be changed once they are initialized, and their values are set permanently.

By declaring state variables as immutable, the compiler can optimize their storage in a way that reduces gas costs. Specifically, the compiler can store the value directly in the bytecode of the contract, rather than in storage, which is a more expensive operation.

AFFECTED CODE

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L21, L26

Issue : Variables should be Immutable

Level : Discussion

Type : Gas Optimization

Remediation : An “immutable” attribute should be added in the parameters that are never updated outside of the constructor to save the gas

Alleviation / Retest : The variables have been made immutable.

Vulnerabilities

DESCRIPTION

In Solidity, data types are automatically assigned default values if they are not explicitly initialized. However, the initialization of default values can sometimes be unnecessary or inefficient, depending on the specific use case.

AFFECTED CODE

- 0x2FF52489A957C3d07381713723664F9d7454F3de#code#F1 #L65

Issue : Unnecessary Default Value Initialization

Level : Discussion

Type : Gas Optimization

Remediation : It's not recommended to initialize the data types to their default values unless there's a use-case because it's unnecessary and costs around ~3 gas.

Alleviation / Retest : The variable is not defining a default value now.

SOURCE Code

Testnet Explorer

https://goerli.etherscan.io/
address/0x543956d87CdB2566f498106eB7774a2B1eb58A91#code

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