



d3ploy



SECURITY ASSESSMENT

Fluid

January 30th 2024

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.

WEBSITE

d3ploy.co



d3ploy

@d3ploy_

TWITTER

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.

- Vulnerability 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.

WEBSITE

d3ploy.co



d3ploy

@d3ploy_

TWITTER

PROJECT

Introduction

Fluid is a fintech innovator offering advanced crypto trading software that integrates with platforms like Telegram and Discord.

With a focus on scalability and a strong development pipeline, we're set for rapid growth. Our edge comes from industry expertise, strategic partnerships, and a commitment to transparency. We're not just following the market, we're leading it.

Imagine executing a market order in under 15 seconds within Telegram, bypassing the tedious steps of logging in, connecting wallets, and manually inputting order details. With Fluid, you can swiftly trade top cryptocurrencies like BTC, ETH, and AVAX with up to 50x leverage, view real-time PnL stats, bridge assets, and even swap tokens seamlessly.

Project Name **Fluid**

Contract Name **FLUID Token**

Contract Address **0x4E47951508Fd4A4126F8ff9CF5E6Fa3b7cC8E073**

Contract Chain Mainnet

Contract Type **Smart Contract**

Platform **EVM**

Language **Solidity**

Network **Ethereum (ERC20), Arbitrum**

Codebase **Private GitHub Repository**

Max Supply **10,000,000**

INFO

Social



<https://fluid.trade/>



<https://twitter.com/FluidToken>



<https://t.me/FluidTradingPortal>



<https://fluidtrade.gitbook.io/docs/>



<https://medium.com/@fluidtrade>



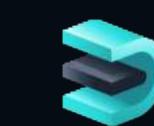
<https://github.com/FluidTrade>



support@fluid.trade

WEBSITE

d3ploy.co



d3ploy

@d3ploy_

TWITTER



A U D I T

Score

◆ Issues	13
◆ Critical	0
◆ Major	2
◆ Medium	3
◆ Minor	3
◆ Informational	1
◆ Discussion	4

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

AUDIT Scope

RAW SOLIDITY FILES

FluidOFTV1.sol

LiquidityManagerV3.sol

RevShareStakingV5.sol

LOCATION

◆ Solidity Files

◆ Solidity Files

◆ Solidity Files

WEBSITE

d3ploy.co



d3ploy

@d3ploy_

TWITTER

REVIEW Methodology

TECHNIQUES

This report has been prepared for Fluid to discover issues and vulnerabilities in the source code of the Fluid 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 2024/01/25

Description Layout project

Architecture / Manual review / Static & dynamic security testing

Summary

Version v1.1

Date 2024/01/30

Description Reaudit addressed vulnerabilities

Final Summary

WEBSITE

d3ploy.co



d3ploy

@d3ploy_

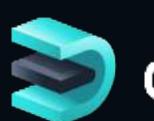
TWITTER

KEY Finding

TITLE	SEVERITY	STATUS
Front-Running Vulnerability in createThePool Function	◆ Major	Fixed
Rounding Error in depositETH Function	◆ Major	Fixed
Zero Slippage Value	◆ Medium	Acknowledged
Uniswap Swap Function with Insecure Deadline	◆ Medium	Acknowledged
Lack of Reversibility in removeLimits Function	◆ Medium	Acknowledged
Missing Return Value Validation	◆ Minor	Partially Fixed
Floating and Outdated Pragma	◆ Minor	Fixed
Use Ownable2Step	◆ Minor	Acknowledged
Missing NatSpec Comments	◆ Informational	Acknowledged

WEBSITE

d3ploy.co



d3ploy

@d3ploy_

TWITTER

KEY Finding

TITLE	SEVERITY	STATUS
Gas Optimization in Increments	◆ Gas	Acknowledged
Custom Error to Save Gas	◆ Gas	Acknowledged
Gas Optimization in Require Statements	◆ Gas	Acknowledged
Large Number Literals	◆ Gas	Acknowledged

WEBSITE

d3ploy.co



d3ploy

@d3ploy_

TWITTER

IN - DEPTH Vulnerabilities

DESCRIPTION

The `createThePool` function in the provided smart contract creates a Uniswap pair for the token. However, it lacks protection against front-running attacks. An attacker could potentially front-run this transaction and create the Uniswap pair before the admin, by calling directly `createPair()` from Uniswap, causing the admin's transaction to revert when attempting to create the pair again. As a result, the `launched` state variable will remain false, and the owner won't be able to launch the token.

AFFECTED CODE

- FluidOFTV1.sol #`createThePool()`

Issue : Front-Running Vulnerability in `createThePool` Function

Level : Major

Type : Front-Running

Remediation : To mitigate the front-running vulnerability, it is recommended to check whether the Uniswap pair already exists before attempting to create it. If the pair already exists, the function should handle the situation gracefully without reverting the transaction.

Alleviation / Retest : The team deployed the following fixes to mitigate this issue:

- Made `createThePool` a private function
- Removed `theLaunchedbool` (not needed in this case)
- Added `createThePool()` to the constructor

IN - DEPTH Vulnerabilities

DESCRIPTION

The depositETH function in the provided smart contract performs a calculation to determine sharePerTokenAtDeposit using the formula `(msg.value * 1e18) / totalShareWithBonus`. However, if `msg.value` is less than `totalShareWithBonus`, due to the rounding behavior in Solidity, the result of this calculation could be zero. Consequently, `sharePerTokenAtDeposit` will be zero, and the subsequent addition to `sharePerTokenCurrent` will not have any effect.

AFFECTED CODE

- RevShareStakingV5.sol #depositETH()

Issue : Rounding Error in depositETH Function

Level : Major

Type : Rounding Error

Remediation : To address the rounding issue and ensure accurate calculations, it is recommended to Check whether `msg.value` is greater than `totalShareWithBonus` before doing this calculation.

Alleviation / Retest : A validation is now added that checks “`sharePerTokenAtDeposit > 0`” when depositing ETH.

IN - DEPTH Vulnerabilities

DESCRIPTION

In swapBack() function, 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

AFFECTED CODE

- FluidOFTV1.sol #swapBack()

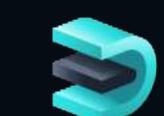
Issue : Zero Slippage Value

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 : Comments fromtheteam:“Slippage risk isn't a concern since the transaction happens along with a users swap and is typically a very small amount anyway. Adding slippage to this function could result in the original users swap to fail and cost them gas.”



IN - DEPTH Vulnerabilities

DESCRIPTION

The provided Contracts FluidOFTV1.sol#swapBack() 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

- FluidOFTV1.sol #swapBack()

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.

For example: Uniswap sets it to 10 minutes on the Ethereum mainnet and to 5 minutes on L2 networks

<https://github.com/Uniswap/interface/blob/main/src/constants/misc.ts#L7-L8>

Alleviation / Retest : Comments from the team: "Deadline is set to the same time as when the function is being executed, this is set to assume the tx will go through at the time of users tx that calls the swapBack function. Adding a future swap deadline could result in the tx failing or receiving 0 ETH due to the Zero Slippage value."

IN - DEPTH Vulnerabilities

DESCRIPTION

The removeLimits function in the smart contract lacks a mechanism for reversibility, as it directly sets the limitsInEffect global variable to false without providing any means to revert or reinstate its original value (true). This irreversible action can have significant implications for the functioning of the smart contract.

AFFECTED CODE

- FluidOFTV1.sol L300

Issue : Lack of Reversibility in removeLimits Function

Level : Medium

Type : Irreversible Action

Remediation : To address this issue and introduce reversibility, consider implementing a more comprehensive approach in the removeLimits function. Instead of directly setting limitsInEffect to false, implement a solution that allows the owner to toggle the limits on and off as needed. For example:

```
function toggleLimits(bool _limitsInEffect) external onlyOwner {  
    limitsInEffect = _limitsInEffect;  
}
```

Alleviation / Retest : Fluid team commented that this is a feature, not a bug.



IN - DEPTH Vulnerabilities

DESCRIPTION

In Solidity, `call()` and `send()` functions are used to interact with other contracts or send Ether. They return a boolean value indicating success (true) or failure (false). If the return value isn't explicitly checked, execution continues even if the call fails. This can lead to unexpected behavior and potential vulnerabilities.

AFFECTED CODE

- FluidOFTV1.sol L425-L427

```
(success, ) = address(teamWallet).call{value: ethForTeam}("");
(success, ) = address(treasuryWallet).call{value: ethForTreasury}("");
(success, ) = address(revWallet).call{value: ethForRev}("");
```

Issue : Missing Return Value Validation

Level : Minor

Type : Missing Validation

Remediation : It is recommended to have input validation on the return values of the call functions mentioned above to make sure invalid data is not returned and that the calls are successful.

Alleviation / Retest : Comments from the team: "If the transfer fails then ETH will be left within the token contract, and the tx will still be executed. The withdrawStuckETH function was added to the contract in case such an issue arose, at which point the ETH can be withdrawn manually."

IN - DEPTH Vulnerabilities

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, i.e., $\geq 0.8.19$. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

AFFECTED CODE

- LiquidityManagerV3.sol [L02](#)

Issue : Floating and Outdated Pragma

Level : Minor

Type : Floating Pragma (SWC-103)

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.

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

Alleviation / Retest : The pragma has been fixed and updated to 0.8.21.

IN - DEPTH 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

- LiquidityManagerV3.sol [L16](#)
- FluidOFTV1.sol [L26](#)

Issue : Use Ownable2Step

Level : Minor

Type : Missing Best Practices

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

Alleviation / Retest : Fluid team acknowledged the issue.



IN - DEPTH 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.

AFFECTED CODE

- FluidOFTV1.sol
- LiquidityManagerV3.sol
- RevShareStakingV5.sol

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 : Fluid team acknowledged the issue.

IN - DEPTH 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.

VULNERABLE CODE

- FluidOFTV1.sol L328

Issue : Gas Optimization in Increments

Level : Gas

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 : Fluid team acknowledged the issue.

IN - DEPTH Vulnerabilities

DESCRIPTION

During code analysis, it was observed that the smart contract is using the `revert()` statements for error handling. However, since Solidity version 0.8.4, custom errors have been introduced, providing a better alternative to the traditional `revert()`. Custom errors allow developers to pass dynamic data along with the revert, making error handling more informative and efficient. Furthermore, using custom errors can result in lower gas costs compared to the `revert()` statements.

AFFECTED CODE

- FluidOFTV1.sol L234

Issue : Custom Error to Save Gas

Level : Gas

Type : Gas Optimization

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

Alleviation / Retest : Fluid team acknowledged the issue.

IN - DEPTH 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.

VULNERABLE CODE

- FluidOFTV1.sol L18

Issue : Gas Optimization in Require Statements

Level : Gas

Type : Gas Optimization

Remediation : It is recommended to shorten the strings passed inside require() statements 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 : Fluid team acknowledged the issue.

IN - DEPTH 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

- FluidOFTV1.sol L40, L359

Issue : Large Number Literals

Level : Gas

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 : Fluid team acknowledged the issue.

SOURCE Code

Raw Solidity Files

- FluidOFTV1.sol
- LiquidityManagerV3.sol
- RevShareStakingV5.sol

WEBSITE

d3ploy.co



d3ploy

@d3ploy_

TWITTER

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

WEBSITE

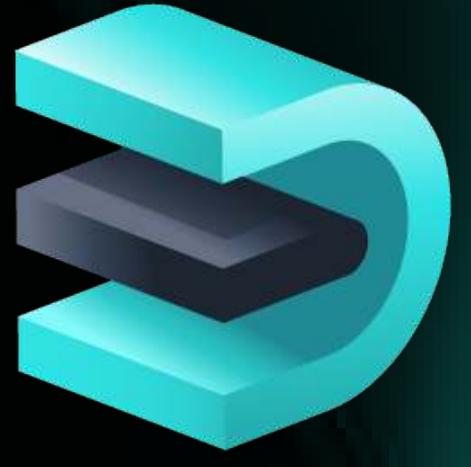
d3ploy.co



d3ploy

@d3ploy_

TWITTER



d3ploy

W E B S I T E

d3ploy.co

@d3ploy_

T W I T T E R