d3ploy

d3ploy

SECURITY ASSESSMENT

WarpGate

Launchpad

March 19th 2024

TABLE OF

Contents

LEGAL

Disclaimer

D3ploy audits are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts d3ploy to perform a security review. D3ploy does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

D3ploy audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort. The report is provided only for the contract(s) mentioned in the report and does not include any other potential additions and/or contracts deployed by Owner. The report does not provide a review for contract(s), applications and/or operations, that are out of this report scope.

D3ploy’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

D3ploy represents an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. D3ploy’s position is that each company and individual are responsible for their own due diligence and continuous security. The security audit is not meant to replace functional testing done before a software release. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits and a public bug bounty program to ensure the security of the smart contracts.

D3PLOY

Introduction

D3ploy is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Secure your project with d3ploy

We offer field-proven audits with in-depth reporting and a range of suggestions to improve and avoid contract vulnerabilities. Industry-leading comprehensive and transparent smart contract auditing on all public and private blockchains.

Vunerability checking

A crucial manual inspection carried out to eliminate any code flaws and security loopholes. This is vital to avoid vulnerabilities and exposures incurring costly errors at a later stage.

Contract verification

A thorough and comprehensive review in order to verify the safety of a smart contract and ensure it is ready for launch and built to protect the end-user

Risk assessment

Analyse the architecture of the blockchain system to evaluate, assess and eliminate probable security breaches. This includes a full assessment of risk and a list of expert suggestions.

In-depth reporting

A truly custom exhaustive report that is transparent and depicts details of any identified threats and vulnerabilities and classifies those by severity.

Fast turnaround

We know that your time is valuable and therefore provide you with the fastest turnaround times in the industry to ensure that both your project and community are at ease.

Best-of-class blockchain engineers

Our engineers combine both experience and knowledge stemming from a large pool of developers at our disposal. We work with some of the brightest minds that have audited countless smart contracts over the last 4 years.

PROJECT

Introduction

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

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

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

INFO

Social

- https://warpgate.pro/
- https://twitter.com/WarpGateX
- https://t.me/WarpGateCommunity
- https://discord.gg/warpgate
- https://medium.com/@warpgate2024
- https://github.com/WarpGate-Labs/
- -

92

PASS

AUDIT

Score

✦ Issues	15
✦ Critical	0
✦ Major	3
✦ Medium	2
✦ Minor	4
✦ Informational	2
✦ Discussion	4

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

AUDIT Scope

CODEBASE FILES	LOCATION
WarpGate-Labs/warp-gate-launchpad	✦ Private Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for WarpGate to discover issues and vulnerabilities in the source code of the WarpGate project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic, Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective in the comments below.

TIMESTAMP

Version	*v1.0*
Date	*2024/02/28*
Descrption	*Layout project*
	Architecture / Manual review / Static & dynamic security testing
	Summary
Version	*v1.1*
Date	*2023/03/19*
Descrption	*Re-audit applied fixes*
	Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
Double Claiming Vulnerability in userClaim() Function	✦ Major	Fixed
Rounding Error Allows Buyer Fee Bypass in calculateBuyerFee() Function	✦ Major	Fixed
Missing Handling of Fees on Transfer in ERC20 Token Transfers	✦ Major	Fixed
DOSDuetoLackofHandling Fees on Token Transfer in transferAndCheck()	✦ Medium	Fixed
Lack of Excess Ether Refund Vulnerability in createV2() Function	✦ Medium	Fixed
Use Ownable2Step	✦ Minor	Acknowledged
Missing Events in Important Functions	✦ Minor	Fixed
Floating and Outdated Pragma	✦ Minor	Acknowledged
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	✦ Minor	Fixed

KEY Finding

TITLE	SEVERITY	STATUS
Use Call instead of Transfer	✦ Informational	Fixed
Functions should be declared External	✦ Informational	Fixed
Public Constants can be Private	✦ Gas	Fixed
Custom Errors instead of Revert	✦ Gas	Fixed
Gas Optimization in Require/Revert Statements	✦ Gas	Fixed
Dead Code	✦ Gas	Acknowledged

IN - DEPTH Vulnerabilities

DESCRIPTION

If the pool type is Instant, the user can swap the tokens immediately. Whereas the userClaim() function allows users to claim swapped tokens from a pool if the type is not instant. However, when the pool type is instant, users can claim the swapped amount twice. This occurs because the function does not enforce a check to prevent users from claiming tokens if they have already received them through _swap().

AFFECTED CODE

- FixedSwap.sol #L386-L392

Issue : Double Claiming Vulnerability in userClaim() Function

Level : Major

Remediation : It is recommended to implement a check in the userClaim() function to verify whether the user has already claimed tokens from a _swap(). If so, prevent them from claiming tokens again through the userClaim function.

Alleviation / Retest : This is fixed by setting an already claimed variable to true preventing double claims.

IN - DEPTH Vulnerabilities

DESCRIPTION

The calculateBuyerFee() function is intended to calculate the buyer fee for swapping tokens from a pool. However, due to a rounding error caused by the use of integer division, the fee calculation can be bypassed if the totalAmount input is sufficiently low. This occurs because the division operation truncates any fractional remainder, resulting in an inaccurate fee calculation.

AFFECTED CODE

- FixedSwap.sol #L306

Issue : Rounding Error Allows Buyer Fee Bypass in calculateBuyerFee() Function

Level : Major

Remediation : It is recommended that while calculating the fees apply validations to avoid rounding error.

Alleviation / Retest : This is fixed. Now the function reverts if fee is 0.

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract contains a vulnerability related to transferring ERC20 tokens without considering the possibility of fees charged on transfer. Some ERC20 tokens implement a fee mechanism, where a certain percentage of tokens is deducted as a fee during each transfer. However, the contract does not account for this possibility when transferring tokens using the safeTransferFrom function.

AFFECTED CODE

- FixedSwap.sol #L273-L276

Issue : Missing Handling of Fees on Transfer in ERC20 Token Transfers

Level : Major

Remediation : To address this vulnerability it is recommended to add a mechanism to calculate the fees on every transfer while accounting.

Alleviation / Retest : The contract is now checking if the final and initial balance such that it should increase by the transferred amount. This is fixed.

IN - DEPTH Vulnerabilities

DESCRIPTION

The transferAndCheck() function in the given code transfers tokens from a specified address(from) to the contract address and then checks if the transferred amount is exactly equal to the specified amount. However, some tokens may charge fees on token transfers, leading to a discrepancy between the transferred amount and the expected amount. As a result, the function will always revert when dealing with tokens that charge fees on transfer, potentially causing a denial of service (DoS) scenario.

AFFECTED CODE

- Base.sol #L226-L238

Issue : DOSDuetoLackofHandling Fees on Token Transfer in transferAndCheck()

Level : Medium

Remediation : To address this vulnerability it is recommended to add a mechanism to calculate the fees on every transfer while accounting.

Alleviation / Retest : The issue has been fixed.

IN - DEPTH Vulnerabilities

DESCRIPTION

The createV2() function allows users to create a pool by paying a fee. However, it fails to refund any excess ether sent by the user beyond the required fee. This oversight results in the loss of any additional ether sent by the user.

AFFECTED CODE

- FixedSwap.sol #L107

Issue : Lack of Excess Ether Refund Vulnerability in createV2() Function

Level : Medium

Remediation : It is recommended to implement a mechanism to refund any excess ether sent by users when creating a pool by adding logic to check if the sent value exceeds the required fee and returning the excess amount back to the sender before completing the pool creation process.

Alleviation / Retest : Excess fee is not returned to the user.

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

- Base.sol #L21

Issue : Use Ownable2Step

Level : Minor

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

Alleviation / Retest : The issue has been acknowledged.

IN - DEPTH Vulnerabilities

DESCRIPTION

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.

The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

AFFECTED CODE

- Base.sol #L176-L183, L185-L187, L189-L191, L193-L197, L304-L306

Issue : Missing Events in Important Functions

Level : Minor

Remediation : Consider emitting events for important functions to keep track of them.

Alleviation / Retest : Important functions are not emitting events

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., 0.8.20. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified.

AFFECTED CODE

- Base.sol #L03
- FixedSwap.sol #L03
- FixedSwapV2.sol #L01
- IFixedSwap.sol #L01

Issue : Floating and Outdated Pragma

Level : Minor

Remediation : Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.23 pragma version

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

Alleviation / Retest : The pragma should be hardcoded and updated to 0.8.23.

IN - DEPTH Vulnerabilities

DESCRIPTION

The transfer() and transferFrom() method is used instead of safeTransfer() and safeTransferFrom(), presumably to save gas however OpenZeppelin's documentation discourages the use of transferFrom(), use safeTransferFrom() whenever possible because safeTransferFrom auto-handles boolean return values whenever there's an error.

AFFECTED CODE

- FixedSwap.sol #L273-L283, L288, L296, L329, L342, L348, L369-L372, L388-L391, L427-L431, L438

Issue : Use safeTransfer/safeTransferFrom instead of transfer/ transferFrom

Level : Minor

Remediation : Consider using safeTransfer() and safeTransferFrom() instead of transfer() and transferFrom(). Also, add a nonReentrant modifier to prevent reentrancy attacks and unintentional results.

Alleviation / Retest : The contract is now using safeTransfer and safeTransferFrom for transferring tokens.

IN - DEPTH Vulnerabilities

DESCRIPTION

Using Solidity's transfer function has some notable shortcomings when the withdrawer is a smart contract, which can render ETH deposits impossible to withdraw. Specifically, the withdrawal will inevitably fail when:

- Thewithdrawer smart contract does not implement a payable fallback function.
- The withdrawer smart contract implements a payable fallback function which uses more than 2300 gas units.
- Thewithdrawer smart contract implements a payable fallback function which needs less than 2300 gas units but is called through a proxy that raises the call's gas usage above 2300.

AFFECTED CODE

- FixedSwap.sol #L294, L327, L436

Issue : Use Call instead of Transfer

Level : Informational

Remediation : It is recommended to transfer ETH using the call() function, handle the return value using require statement, and use the nonreentrant modifier wherever necessary to prevent reentrancy.

Ref: *https://solidity-by-example.org/sending-ether/*

Alleviation / Retest : The contract is now using .call instead of transfer for sending ETH.

IN - DEPTH 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

- FixedSwapV2.sol #L06-L08

Issue : Functions should be declared External

Level : Informational

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

Alleviation / Retest : The function has been set to external.

IN - DEPTH Vulnerabilities

DESCRIPTION

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

AFFECTED CODE

- Base.sol #L29

Issue : Public Constants can be Private

Level : Gas

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

Alleviation / Retest : This is fixed. The variable is no longer public.

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract was found to be using a revert() statement. 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.

AFFECTED CODE

- FixedSwap.sol #L108, L270, L339

Issue : Custom Errors instead of Revert

Level : Gas

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

Alleviation / Retest : Custom errors are now being used.

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.

AFFECTED CODE

- Base.sol #L129-L132, L142-L145, L148-L150

Issue : Gas Optimization in Require/Revert Statements

Level : Gas

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 : The issue has been fixed.

IN - DEPTH Vulnerabilities

DESCRIPTION

It is recommended to keep the production repository clean to prevent confusion and the introduction of vulnerabilities. The functions and parameters, contracts, and interfaces that are never used or called externally or from inside the contracts should be removed when the contract is deployed on the mainnet.
The contracts were found to be defining some variables that are not used anywhere in the code.

AFFECTED CODE

- Base.sol #L346-L349

Issue : Dead Code

Level : Gas

Remediation : If the variables are not supposed to be used anywhere, consider removing them from the contract.

Alleviation / Retest : The issue has been acknowledged.

SOURCE Code

Private GitHub Repository

REPORT Appendix

FINDING CATEGORIES

The assessment process will utilize a mixture of static analysis, dynamic analysis, in-depth manual review and/or other security techniques.

This report has been prepared for WarpGate project using the above techniques to examine and discover vulnerabilities and safe coding practices in WarpGate's smart contract including the libraries used by the contract that are not officially recognized.

A comprehensive static and dynamic analysis has been performed on the solidity code in order to find vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds.

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The testing methods find and flag issues related to gas optimizations that help in reducing the overall gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

AUDIT SCORES

D3ploy Audit Score is not a live dynamic score. It is a fixed value determined at the time of the report issuance date.

D3ploy Audit Score is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports and scores are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts d3ploy to perform a security review.

d3ploy

WEBSITE d3ploy.co

@d3ploy_ TWITTER