d3ploy

d3ploy

SECURITY ASSESSMENT

Zebec Protocol

June 26th 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

Zebec, a pioneer in streaming finance, enables real-time and continuous streams of payments and financial transactions for payroll, investments and more.

Zebec's vision extends beyond web3 applications. The company aims to create a future where money is able to move more freely; giving individuals, businesses, investors, and teams faster and easier access to funds and tokens. To pave the way for the mass adoption of real-time payroll, Zebec deploys its cutting-edge technology to the traditional payroll providers.

Project Name *Zebec Protocol*
Contract Name *ZBC Token*
Contract Address *0x37a56cdcD83Dce2868f721De58cB3830C44C6303*
Contract Chain *Mainnet*
Contract Type *Smart Contract*
Platform *EVM*
Language *Solidity*
Network *Solana (SOL) & BNB Chain (BEP20)*
Codebase *Private GitHub Repository*
Total Token Supply *10,000.000.000*

INFO

Social

https://zebec.io/

https://twitter.com/Zebec_HQ

https://t.me/zebececosystem

https://discord.com/jUwZ3cHauZ

-

https://github.com/Zebec-protocol

-

89

PASS

AUDIT

Score

Issues	20
Critical	2
Major	1
Medium	0
Minor	3
Informational	5
Discussion	9

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

AUDIT Scope

CODEBASE FILES	LOCATION
Zebec-protocol/bnb-zebec-contract	✦ Private Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for Zebec Protocol to discover issues and vulnerabilities in the source code of the Zebec Protocol 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:

- Testingthesmartcontractsagainstbothcommonanduncommonattackvectors.
- 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 2023/06/12

Descrption Layout project
Architecture / Manual review / Static & dynamic security testing
Summary

Version v1.1

Date 2023/06/26

Descrption Re-audit addressed issues
Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
MISSING MODIFIERS [ISWHITELISTEDTOKEN]	✦ Critical	Fixed
MISSING MODIFIERS [WHENNOTPAUSED]	✦ Critical	Fixed
VARIABLES DECLARED BUT NEVER USED	✦ Gas	Fixed
ARRAY LENGTH CACHING	✦ Gas	Fixed
CHEAPER INEQUALITIES IN IF()	✦ Gas	Fixed
EVENT BASED REENTRANCY	✦ Low	Fixed
USE OF FLOATING PRAGMA	✦ Low	Fixed
UNCHECKED ARRAY LENGTH	✦ Major	Fixed
GAS OPTIMIZATION IN INCREMENTS	✦ Gas	Fixed
INTERNAL FUNCTIONS NEVER USED	✦ Gas	Fixed

KEY Finding

TITLE	SEVERITY	STATUS
MISSING EVENTS	✦ Low	Fixed
MISSING INDEXED KEYWORDS IN EVENTS	✦ Informational	Fixed
MISSING STATE VARIABLE VISIBILITY	✦ Informational	Fixed
PUBLIC CONSTANTS CAN BE PRIVATE	✦ Gas	Fixed
REQUIRE WITH EMPTY MESSAGE	✦ Informational	Fixed
RETURN INSIDE LOOP	✦ Informational	Fixed
USE OF SAFEMATH LIBRARY	✦ Gas	Fixed
UNNECESSARY CHECKED ARITHMETIC IN LOOP	✦ Gas	Fixed
FUNCTION SHOULD BE EXTERNAL	✦ Gas	Fixed
UNUSED RECEIVE FALLBACK	✦ Informational	Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

Access control plays an important role in the segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens, and in some cases compromise of the smart contract.

The contract Core is using a modifier isWhitelistedToken to check if the tokens are whitelisted or not but the functions instantStream() and instantStreamTNS are missing the modifier.
This could allow users to create streams with any token that is not whitelisted.

LOCATION

- contracts/Core.sol L555-618

Issue : MISSING MODIFIERS [ISWHITELISTEDTOKEN]

Level : Critical

Remediation : It is recommended to add the isWhitelistedToken modifier to all the functions that are creating streams using address inputs obtained from end-users

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

Access control plays an important role in the segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens, and in some cases compromise of the smart contract.

The contract Core is using a modifier whenNotPaused to check if the contract is paused but some of the business-critical functions are missing this modifier.
This could allow users to use the contract's functions even when the contract is in a paused state.

LOCATION

- contracts/Core.sol L287-315; L796-831; L1209-1257

Issue : MISSING MODIFIERS [WHENNOTPAUSED]

Level : Critical

Remediation : It is recommended to add the whenNotPaused modifier to all the business-critical functions.

Alleviation / Retest : The team commented on the bug with sensible logic.

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract Staking has declared a variable coreContract but it is not used anywhere in the code. This represents dead code or missing logic.
Unused variables increase the contract's size and complexity, potentially leading to higher gas costs and a larger attack surface.

LOCATION

- /contracts/Staking.sol L50

Issue : VARIABLES DECLARED BUT NEVER USED

Level : Gas

Remediation : To remediate this vulnerability, developers should perform a code review and remove any variables that are declared but never used.

Alleviation / Retest : Fixed

IN - DEPTH 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.

LOCATION

- /contracts/Staking.sol L108-114; L226-231; L204-213; L1131-1133; L1144-1150; L1285-1301; L 1304-1317

Issue : ARRAY LENGTH CACHING

Level : Gas

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

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

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

LOCATION

- /contracts/libs/BulkTransferLibrary.sol L36-39; L90-93
- /contracts/BulkTransfer.sol L51; L87; L93
- /contracts/Staking.sol L135: L156
- /contracts/Core.sol L301; L410; L706; L721-722; L731; L914; L968; L984; L1033; L1035; L1057; L1059 L1099; L1169; L1232; L1237

Issue : CHEAPER INEQUALITIES IN IF()

Level : Gas

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

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

n a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.In the case of event-based Re-entrancy attacks, events are emitted after an external call leading to missing event calls.

LOCATION

- /contracts/Core.sol L555-618

Issue : EVENT BASED REENTRANCY

Level : Low

Remediation : It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing and event emits must happen before the call.

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

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

LOCATION

- /contracts/interface/IStaking.sol L02
- /contracts/interface/ICore.sol L02
- /contracts/interface/IBulkTransfer.sol L02
- /contracts/interface/IFundTransfer.sol L02
- /contracts/interface/IRegistry.sol L02
- /contracts/libs/CoreUtilsLibrary.sol L03
- /contracts/libs/BulkTransferLibrary.sol L03
- /contracts/BulkTransfer.sol L02
- /contracts/Staking.sol L02
- /contracts/Core.sol L03

Issue : USE OF FLOATING PRAGMA

Level : Low

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

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of Gas consumed in a block. If array.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed.

```
for (uint256 i = 0; i < array.length ; i++) { cosltyFunc(); }
```

This becomes a security issue if an external actor influences array.length.
E.g., if an array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.

LOCATION

- /contracts/Staking.sol L108

Issue : UNCHECKED ARRAY LENGTH

Level : Major

Remediation : Either explicitly or just due to normal operation, the number of iterations in a loop can grow beyond the block gas limit, which can cause the complete contract to be stalled at a certain point. Therefore, loops with a bigger or unknown number of steps should always be avoided.

Alleviation / Retest : Fixed. 600 limit is in place.

IN - DEPTH Vulnerabilities

DESCRIPTION

++i costs less gas compared to i++ or i += 1 for unsigned integers. In i++, the compiler has to create a temporary variable to store the initial value. This is not the case with ++i in which the value is directly incremented and returned, thus, making it a cheaper alternative.

LOCATION

- /contracts/Staking.sol L108; L226
- /contracts/Core.sol L204; L1131; L1144; L1285; L1287; L1304

Issue : GAS OPTIMIZATION IN INCREMENTS

Level : Gas

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

Alleviation / Retest : Fixed

DESCRIPTION

The contract declared internal functions but was not using them in any of the functions or contracts.
Since internal functions can only be called from inside the contracts, it makes no sense to have them if they are not used. This uses up gas and causes issues for auditors when understanding the contract logic.

LOCATION

- /contracts/Core.sol L1320-1322; L1353-1365

Issue : INTERNAL FUNCTIONS NEVER USED

Level : Gas

Remediation : Having dead code in the contracts uses up unnecessary gas and increases the complexity of the overall smart contract.
It is recommended to remove the internal functions from the contracts if they are never used.

Alleviation / Retest : Fixed. The team commented on the bug with sensible logic.

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 Staking / Core was found to be missing these events on the function grantWhitelisterRole which would make it difficult or impossible to track these transactions off-chain.

LOCATION

- /contracts/Staking.sol L63-72; L102-115; L179-219
- /contracts/Core.sol L182-186; L188-191; L1328-1330

Issue : MISSING EVENTS

Level : Low

Remediation : Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

Alleviation / Retest : The team commented on the bug with sensible logic.

IN - DEPTH Vulnerabilities

DESCRIPTION

Events are essential for tracking off-chain data and when the event paraemters are indexed they can be used as filter options which will help getting only the specific data instead of all the logs.

LOCATION

- /contracts/interface/IStaking.sol L06; L09
- /contracts/interface/ICore.sol L49-59

Issue : MISSING INDEXED KEYWORDS IN EVENTS

Level : Informational

Remediation : Consider adding indexed keyword to crucial event parameters that could be used in off-chain tracking. Do remember that the indexed keyword costs more gas.

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

Visibility modifiers determine the level of access to the variables in your smart contract. This defines the level of access for contracts and other external users. It makes it easier to understand who can access the variable.
The contract defined a state variable coreContract / tokenAddress / whitelistedTokens / staking / bulkTransfer / tnsRegistry that was missing a visibility modifier.

LOCATION

- /contracts/Staking.sol L50
- /contracts/Staking.sol L51
- /contracts/Core.sol L160
- /contracts/Core.sol L162
- /contracts/Core.sol L163
- /contracts/Core.sol L164

Issue : MISSING STATE VARIABLE VISIBILITY

Level : Informational

Remediation : Explicitly define visibility for all state variables. These variables can be specified as public, internal or private.

Alleviation / Retest : Fixed

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.
The following variable is affected: MAX_FEE / WHITELISTER_ROLE / WHITELIST_ROLE / FUND_WITHDRAW_ROLE / WITHDRAW_ROLE

LOCATION

- /contracts/Staking.sol L44
- /contracts/Staking.sol L52
- /contracts/Core.sol L37
- /contracts/Core.sol L38-39
- /contracts/Core.sol L40

Issue : PUBLIC CONSTANTS CAN BE PRIVATE

Level : Gas

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

Alleviation / Retest : Fixed

DESCRIPTION

A require statement was detected with an empty message. It takes two parameters and the message part is optional. This is shown to the user when and if the require statement evaluates to false. This message gives more information about the statement and why it gave a false response.

LOCATION

- /contracts/Core.sol L1372

Issue : REQUIRE WITH EMPTY MESSAGE

Level : Informational

Remediation : It is recommended to add a descriptive message, no longer than 32 bytes, inside the require statement to give more detail to the user about why the condition failed.

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

The function _getStakingStreamFee has defined a return keyword inside a ForStatement loop. This represents an error because the loop will simply return on it's first iteration.

LOCATION

- /contracts/Staking.sol L226-231

Issue : RETURN INSIDE LOOP

Level : Informational

Remediation : Instead of return, the contract should have used break to at least run the other iterations of the first loop.

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

SafeMath library is found to be used in the contract. This increases gas consumption than traditional methods and validations if done manually.
Also, Solidity 0.8.0 includes checked arithmetic operations by default, and this renders SafeMath unnecessary.

LOCATION

- /contracts/libs/CoreUtilsLibrary.sol L09
- /contracts/BulkTransfer.sol L19
- /contracts/Staking.sol L20
- /contracts/Core.sol L34

Issue : USE OF SAFEMATH LIBRARY

Level : Gas

Remediation : We do not recommend using 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 should be upgraded to Solidity version 0.8.0+ which automatically checks for overflows and underflows.

Alleviation / Retest : safemath is not used anymore. It is also recommended to remove it's import statement.

IN - DEPTH Vulnerabilities

DESCRIPTION

Increments inside a loop could never overflow due to the fact that the transaction will run out of gas before the variable reaches its limits. Therefore, it makes no sense to have checked arithmetic in such a place.

LOCATION

- /contracts/Staking.sol L108; L226
- /contracts/Core.sol L204; L1131; L1144; L1285; L1287; L1304

Issue : UNNECESSARY CHECKED ARITHMETIC IN LOOP

Level : Gas

Remediation : It is recommended to have the increment value inside the unchecked block to save some gas.

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

A function with public visibility modifier was detected that is not called internally. public and external differs in terms of gas usage. The former use more than the latter when used with large arrays of data. This is due to the fact that Solidity copies arguments to memory on a public function while external read from calldata which a cheaper than memory allocation.

LOCATION

- /contracts/Core.sol L431-456

Issue : FUNCTION SHOULD BE EXTERNAL

Level : Gas

Remediation : If you know the function you create only allows for external calls, use the external visibility modifier instead of public. It provides performance benefits and you will save on gas.

Alleviation / Retest : Fixed

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract was found to be defining an empty fallback / receive function.
It is not recommended to leave them empty unless there's a specific use case such as to receive Ether via an empty receive() function.

LOCATION

- /contracts/Staking.sol L241; L244
- /contracts/Core.sol L1324; L1326

Issue : UNUSED RECEIVE FALLBACK

Level : Informational

Remediation : It is recommended to go through the code to make sure these functions are properly implemented and are not missing any validations in the definition.

Alleviation / Retest : receive() has been removed. An empty fallback is kept. Fixed.

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