d3ploy

d3ploy

SECURITY ASSESSMENT

TEN Finance

November 23rd 2022

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

TEN, the Token Enrichment Network, is the next generation of Decentralized Finance, SIMPLIFIED. TEN leverages innovations brought to market by first generation DeFi projects and integrates emerging technologies to enhance yield optimization in a simple yet efficient manner.

TEN simplifies staking with the most liquid BNB Chain Liquidity Pools available and provides a robust yield earning environment on the market while adapting to daily liquidity needs of different pools. TEN is Decentralized Finance, Simplified!

Project Name TEN Finance
Contract Name TENFI Token
Contract Address 0xd15c444f1199ae72795eba15e8c1db44e47abf62
Contract Chain Mainnet
Contract Type Smart Contract
Platform EVM
Language Solidity
Network BNB Chain (BEP20)
Codebase Private GitHub Repository
Total Token Supply 256.000.000

INFO

Social

https://www.ten.finance/

https://twitter.com/tenfinance

https://t.me/tenfinance

-

https://medium.com/tenfinance

https://github.com/tenfinance

contact@ten.finance

93

PASS

AUDIT

Score

Issues	4
Critical	0
Major	0
Medium	0
Low	3
Informational	1
Discussion	0

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

AUDIT Scope

CODEBASE FILES	LOCATION
tenfinance/BnbY/contracts/ BnbY.sol	✦ Private GitHub Repository
tenfinance/BnbY/contracts/ IBnbY.sol	✦ Private GitHub Repository
tenfinance/BnbY/contracts/ StakeManager.sol	✦ Private GitHub Repository
tenfinance/BnbY/contracts/ IStakeManager.sol	✦ Private GitHub Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for TEN Finance to discover issues and vulnerabilities in the source code of the TEN Finance 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	*2022/11/23*
Descrption	*Layout project*
	Architecture / Manual review / Static & dynamic security testing
	Summary

KEY Finding

TITLE	SEVERITY	STATUS
Floating Pragma	✦ Minor	Pending
Missing Input Validation in Minting Amount	✦ Minor	Pending
Gas Optimization in Require Statements	✦ Gas	Pending
Missing Two-Step Validation in Critical Address Change	✦ Minor	Pending

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 contracts found in the repository were allowing floating or unlocked pragma to be used, i.e., ^0.8.0.
This allows the contracts to be compiled with all the solidity compiler versions above 0.8.0. The following contracts were found to be affected -

LOCATION

- BnbY.sol
- IBnbY.sol
- StakeManager.sol
- IStakeManager.sol

IMPACTS

If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.
Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is really low therefore this is only informational.

Issue : Floating Pragma

Type : Floating Pragma (SWC-103)

Level : Low

Remediation : Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere.

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

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Input validation is a frequently-used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe for processing within the code, or when communicating with other components.
When the smart contract does not validate the inputs properly, it may introduce a range of vulnerabilities.
The contract did not implement any higher range when minting tokens inside the function mint()

LOCATION

- BnbY.sol - mint() - L35-L30

IMPACTS

If the contract allows unlimited minting, the tokens price could be manipulated by the staking manager.

Issue : Missing Input Validation in Minting Amount

Type : Input Validation

Level : Low

Remediation : Define minting limits and use a require() input validation in the minting amount.

Alleviation / Retest :

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.

LOCATION

- StakeManager.sol L92; L422-L425; L490

IMPACTS

Having longer require strings than 32 bytes costs a significant amount of gas.

Issue : Gas Optimization in Require Statements

Type : Gas Optimization

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 :

IN - DEPTH Vulnerabilities

DESCRIPTION

Changing critical addresses in contracts should be a two-step process where the first transaction (from the old/current address) registers the new address (i.e. grants ownership) and the second transaction (from the new address) replaces the old address with the new one (i.e. claims ownership). This gives an opportunity to recover from incorrect addresses mistakenly used in the first step. If not, contract functionality might become inaccessible.
The contract did not implement two-step validation when setting the stake manager's address in the function setStakeManager().

LOCATION

- BnbY.sol - setStakeManager() - L40-L51

IMPACTS

If the contract does not validate properly for critical address changes, any mistake from the owner when changing the address could prove fatal and the contract and its funds might be lost forever.

Issue : Missing Two-Step Validation in Critical Address Change

Type : Missing Best Practices

Level : Low

Remediation : Enable a two-step process for critical address changes.

Alleviation / Retest :

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