d3ploy

d3ploy

SECURITY ASSESSMENT

BattleVerse

November 17th 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

BattleVerse.io is a P2E online game powered by DeFi x NFT and blockchain technology. Binance Smart Chain GameFi Hackathon prize winner.

Their main goal is to give the opportunity to earn through fun gameplay, even if one is not an experienced player in the world of Play-to-Earn (P2E) games. A player's every action in the huge game world, be it a battle, resource extraction, or completing missions and quests, will generate income.

Project Name	*BattleVerse*
Contract Name	*BVC Token*
Contract Address	*0x9bee0c15676a65ef3c8cdb38cb3dd31c675bbd12*
Contract Chain	*Mainnet*
Contract Type	*Smart Contract*
Platform	*EVM*
Language	*Solidity*
Network	*BNB Chain (BEP20)*
Codebase	*Private GitHub Repository*
Total Token Supply	*1,000.000.000*

INFO

Social

https://battleverse.io/

https://twitter.com/BattleVerse_io

https://t.me/battleverse_io

https://discord.com/HFVAnBS9qA

https://battleverse.medium.com/

https://github.com/tenfinance

marketing@battleverse.io

96

PASS

AUDIT

Score

Issues	7
Critical	0
Major	0
Medium	0
Low	4
Informational	3
Discussion	0

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

AUDIT Scope

CODEBASE FILES	LOCATION
BattleVerseIo/Marketplace-contracts/contracts/ Marketplace.sol	✦ Private GitHub Repository
BattleVerseIo/Marketplace-contracts/contracts/ Migrations.sol	✦ Private GitHub Repository
BattleVerseIo/Marketplace-contracts/contracts/ SimpleToken.sol	✦ Private GitHub Repository
BattleVerseIo/Marketplace-contracts/contracts/ TestNFT.sol	✦ Private GitHub Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for BattleVerse to discover issues and vulnerabilities in the source code of the BattleVerse 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	2022/11/07
Descrption	Layout project
	Architecture / Manual review / Static & dynamic security testing
	Summary
Version	v1.1
Date	2022/11/17
Descrption	Reaudit addressed issues
	Final Summary

KEY Finding

TITLE	SEVERITY	STATUS
Pragma Version Too Recent	✦ Low	Resolved
Floating Pragma	✦ Low	Resolved
Missing Events in Important Functions	✦ Low	Resolved
Gas Optimization in Require Statements	✦ Gas	Acknowledged
Unindexed Event Parameters	✦ Informational	Acknowledged
Cheaper Inequalities In Require()	✦ Gas	Acknowledged
Missing Zero Address Validations	✦ Low	Resolved

IN - DEPTH Vulnerabilities

DESCRIPTION

Solidity constantly releases new compiler versions and it is not recommended to stay on the latest and recent versions as there may be unidentified bugs, inconsistencies, and exploits present in the newer versions.

LOCATION

- Marketplace.sol
- SimpleToken.sol

IMPACTS

Recent compiler versions are not time-tested and may be susceptible to unknown vulnerabilities and exploits.

Issue : Pragma Version Too Recent

Type : Missing Best Practices

Level : Low

Remediation : It is suggested to use a compiler version that is neither too recent nor too old. A stable compiler version should be used that is time-tested by the community, which fixed vulnerabilities introduced in older compiler versions. The code should be kept updated according to the compiler release cycle. It should be tested before going on the Mainnet to reduce the chances of new vulnerabilities being introduced.

It is recommended to use version 0.8.7 which is the most stable at this time. .

Alleviation / Retest : The BattleVerse team opted to consider our recommendation and applied the suggested 0.8.7 compiler version.

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., ^0.8.15.
This allows the contracts to be compiled with all the solidity compiler versions above 0.8.15. The following contracts were found to be affected -

LOCATION

- Marketplace.sol
- SimpleToken.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 : The BattleVerse team opted to consider our references and applied the recommendation.

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.

LOCATION

The following functions were affected

- Marketplace.setFee()
- Marketplace.setBeneficiary()

IMPACTS

Events are used to track the transactions off-chain and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Issue : Missing Events in Important Functions

Type : Missing Best Practices

Level : Low

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

Alleviation / Retest : The BattleVerse team heeded our references and applied the suggested recommendation.

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

- Marketplace.sol L29; L125

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 : BattleVerse team will leave the checks 'as-is' for better readability. Agreed as not an exploitable issue.

IN - DEPTH Vulnerabilities

DESCRIPTION

Events in solidity contain two kinds of parameters - indexed and non-indexed. These indexes are also known as "topics" and are the searchable parameters used in events.
In the Ethereum system, events must be easily searched for, so that applications can filter and display historical events without undue overhead.
It was noticed that the following event parameters were not indexed making the search for past events cumbersome.

LOCATION

- Marketplace.sol L14-L17

IMPACTS

This does not impact the security aspect of the Smart contract but affects the ease of use when searching for past events.

Issue : Unindexed Event Parameters

Type : Missing Best Practices

Level : Informational

Remediation : It should be noted that indexed event parameters take up more gas than non-indexed ones. Keeping that in mind, the contract should add indexed keywords to the searchable parameters to make searching efficient using an event filter.

Alleviation / Retest : BattleVerse team commented that their app does not require searching for historical data, and they index that data off-chain. Will keep parameters unindexed. Agreed as not an exploitable issue.

IN - DEPTH Vulnerabilities

DESCRIPTION

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

LOCATION

- Marketplace.sol L30; L131

IMPACTS

Using non-strict inequalities inside require statements cost more gas.

Issue : Cheaper Inequalities In Require()

Type : Gas & Missing Best Practices

Level : Gas

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

Alleviation / Retest : BattleVerse team will leave the checks 'as-is' for better readability. Agreed as not an exploitable issue.

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract Marketplace.sol was found to be setting or using 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 burnt 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.

LOCATION

- constructor() - IERC20 _BVC L28

IMPACTS

If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Issue : Missing Zero Address Validations

Type : Missing Input Validation

Level : Low

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

Alleviation / Retest : The BattleVerse team heeded our references and applied the suggested recommendation.

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