AUDITS

MH SWIFTSCAN REVIEW

UPONLY

JUNE 23 RD 2022

AUDITS

TABLE OF CONTENTS

LEGAL DISCLAIMER

MH 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 MH Audits to perform a security review.

MH Audits 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.

MH 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.

MH Audits’ 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.

MH Audits 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. MH Audits’ 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.

INTRODUCTION

MH Audits 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 MH Audits
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 SUMMARY

PROJECT INTRODUCTION

UpOnly – A Premier Play-to-Earn Data and Prediction Platform.

UpOnly is building the industry's first play-to-earn data directory. The platform will compile comprehensive data on play-to-earn games and establish itself as the go-to-resource for blockchain gamers, similar to what CoinMarketCap and Coingecko accomplished for the broader crypto economy. The UpOnly data directory will be underpinned by a centralized database architecture and will utilize query solutions such as TheGraph to retrieve real-time data from listed blockchain games.

Project Name *UpOnly*

Contract Name *UPO Token*

Contract Address *0x9dBfc1cbf7a1E711503a29B4b5F9130ebeCcaC96*

Contract Chain *Mainnet*

Contract Type *Smart Contract*

Platform *EVM*

Language *Solidity*

Codebase *https://polygonscan.com/address/0x9dBfc1cbf7a1E711503a29B4b5F9130ebeCcaC96#code*

INFO & SOCIALS

Network *Polygon (MATIC)*

Max Token Supply *160.000.000*

Website *https://uponly.com/*

Twitter *https://twitter.com/UpOnlyOfficial*

Telegram Chat *https://t.me/UpOnlyOfficial*

Telegram Ann *https://t.me/UpOnlyNews*

Substack *https://uponlyofficial.substack.com/*

PolygonScan *https://polygonscan.com/token/0x9dBfc1cbf7a1E711503a29B4b5F9130ebeCcaC96*

AUDIT SCORES

75*

PASS

Issues	10
◆ Critical	0
◆ Major	0
◆ Medium	5
◆ Minor	5
◆ Informational	0
◆ Discussion	0

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

* Note that if no manual in-depth expert review has been performed a score multiplier of .9 will apply to the final result.

AUDIT SCOPE

FILE	LOCATION
Token.sol	*Polygon Deployment: /address/0x9dbfc1cbf7a1e711503a29b4b5f9130ebeccac96#code*

REVIEW
METHODOLOGY

TECHNIQUES

This report has been prepared for UpOnly to discover issues and vulnerabilities in the source code of the UpOnly project as well as any contract dependencies that were not part of an officially recognized library. An examination has been performed, utilizing Static Analysis and MH SwiftScan 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.

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	2022/06/23
Description	Layout project Automated / Static security testing Summary

KEY FINDINGS

TITLE	SEVERITY	STATUS
State Modifying Functions	◆ Medium	*Pending*
Solidity Modifier Side Effects	◆ Medium	*Pending*
Unprotected Ether Withdrawal	◆ Medium	*Pending*
Account Existence Check For Low Level Calls	◆ Medium	*Pending*
Using extcodesize To Check For Externally Owned Accounts	◆ Medium	*Pending*
Custom Errors To Save Gas	◆ Minor	*Pending*
Outdated Complier Version	◆ Minor	*Pending*
Assert Require State Changes	◆ Minor	*Pending*
Cheaper Inequalities In Require()	◆ Minor	*Pending*
Cheaper Inequalities In If()	◆ Minor	*Pending*

IN-DEPTH VULNERABILITIES

Description:

If you are migrating your smart contract from versions <0.5.0 *and you used a constant function that is now marked as a pure or view, special care should be taken if these functions modify the state as it may trap a contract in solidity versions* >=0.5.0

Location: *contracts/Token.sol L649-L660; L803-L812*

Issue: *State Modifying Functions*

Level: *Medium*

Recommendation: *Do not modify the state of the contract inside functions that are marked as* pure, view, or constant*.*

Alleviation:

IN-DEPTH VULNERABILITIES

Description:

Solidity functions should always use the Checks-Effects-Interactions *pattern which states that the initial stage will contain only checks and validations which resides in the modifiers. Due to this reason, modifiers should only implement checks and validations inside of it and should not make state changes and external calls. A contract was found to be violating this pattern and the modifier was making sensitive state changes and modifications.*

Location: *contracts/Token.sol L1699-L1702*

Issue: *Solidity Modifier Side Effects*

Level: *Medium*

Recommendation: *Only use modifiers for implementing checks and validations. Do not make external calls or state changing actions inside modifiers.*

Alleviation:

IN-DEPTH
VULNERABILITIES

Description:

Ether and tokens are the basis of smart contracts on which the contract runs and executes transactions. Therefore, it is absolutely necessary to have input and access control validations on the functions executing funds withdrawal within the contract. The following unprotected public and external functions were found which were accepting addresses controlled by external users.

Location: *contracts/Token.sol L1367-L1373*

Issue: *Unprotected Ether Withdrawal*

Level: *Medium*

Recommendation: *It is recommended to go through the functions and make sure that the ether withdrawal implements an access control, input validation, and/or that the funds of the user is depreciated after they withdraws the amount.*

Alleviation:

IN-DEPTH VULNERABILITIES

Description:

The low-level calls such as the delegatecall*,* call*, or* callcode*, do not validate prior to the call if the destination account exists or not. They will always return true even if the account is non-existent, therefore, giving invalid output.*

Location: *contracts/Token.sol L677-L688*

Issue: *Account Existence Check For Low Level Calls*

Level: *Medium*

Recommendation: *It is recommended to have an account existence check before making these low-level calls to confirm the presence of an external account with some valid code. Eg: using* extcodesize*.*

Alleviation:

IN-DEPTH

VULNERABILITIES

Description:

extcodesize is used to check if a contract is an externally owned account or another contract. extcodesize returns 0 for externally owned accounts but there's a specific condition here that when an extcodesize check is made to a contract that is still under construction or when the contract's constructor is running, extcodesize for its address returns zero. This may give erroneous outputs for checking externally owned contracts.

Location: *contracts/Token.sol L656*

Issue: *Using extcodesize To Check For Externally Owned Accounts*

Level: *Medium*

Recommendation: *It is recommended to manually check and validate at compile-time that the contract/account address being checked inside extcodesize does not return improper values due to the external contract's construction.*

Alleviation:

IN-DEPTH VULNERABILITIES

Description:

The contract was found to be using revert() *statements. 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 making the whole implementation a bit cheaper than using reverts.*

Location: *contracts/Token.sol L872*

Issue: *Custom Errors To Save Gas*

Level: *Minor*

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

Alleviation:

IN-DEPTH VULNERABILITIES

Description:

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Location: *contracts/Token.sol L09; L100; L126; L150; L545; L588; L626; L880; Global*

Issue: *Outdated Complier Version*

Level: *Minor*

Recommendation: *It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version* 0.8.7, *which patches most solidity vulnerabilities.*

Alleviation:

IN-DEPTH VULNERABILITIES

Description:

Statements inside require *and* assert *should not change state through any function call or keyword. The contract was found to be making state changes inside the* require *or* assert *statements.*

Location: *contracts/Token.sol L1145-L1148; L1217-L1220;*

Issue: *Assert Require State Changes*

Level: *Minor*

Recommendation: *It is recommended to not make any state changes inside* assert *or* require *statements and to always follow the pattern of check-effects-interactions. Assert should only be used to check invariants and should be replaced with* require *for user input and return values.*

Alleviation:

Description:

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

Location: *contracts/Token.sol L323; L379; L415; L466; L576; L617; L679; L767; L1920*

Issue: *Cheaper Inequalities In* Require()

Level: *Minor*

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

Alleviation:

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/Token.sol L864*

Issue: *Cheaper Inequalities In* If()

Level: *Minor*

Recommendation: *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:

SOURCE CODE

https://polygonscan.com/address/0x9dbfc1cbf7a1e711503a29b4b5f9130ebeccac96#contract

REPORT APPENDIX

FINDING CATEGORIES

The assessment process will utilize a mixture of static analysis, swift scan and other security techniques.

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

The scan runs a comprehensive static analysis on the solidity code and finds vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds. The coverage scope pays attention to all the informational and critical vulnerabilities with over (110+) modules. The scanning and auditing process covers the following areas:

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The scanner modules 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

MH Audits AuditScores is not a live dynamic score. It is a fixed value determined at the time of the report issuance date.

*****Note that if no manual in-depth expert review has been performed a score multiplier of .9 will apply to the final result.

MH Audits AuditScores are 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 MH Audits to perform a security review.

AUDITS

WEBSITE
MHAUDITS.IO

TWITTER
@MHAUDITS