QuillAudits

Audit Report
April, 2024

For

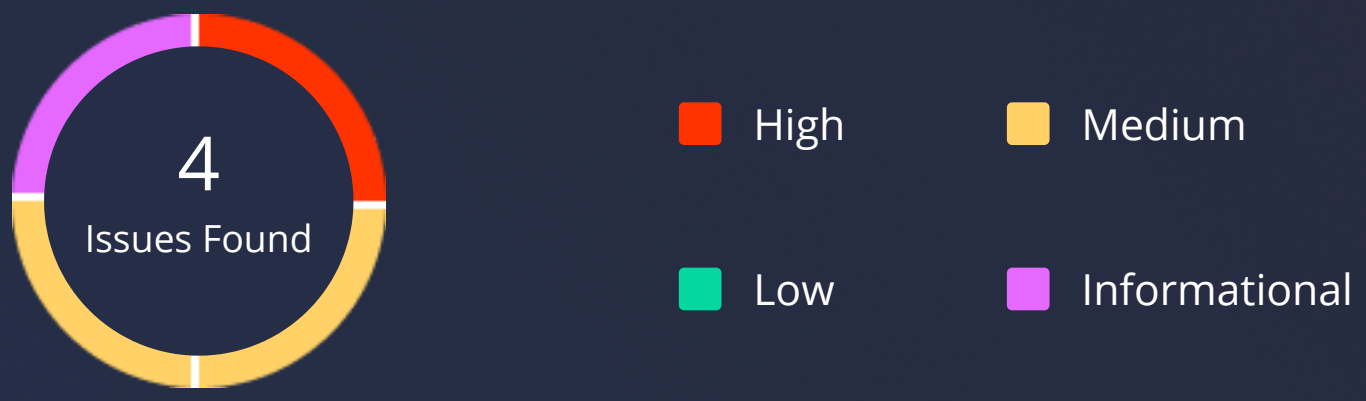
NITRO DOME

Table of Content

Executive Summary

Project Name	NitroDome
Project URL	*https://www.nitrodome.com/*
Overview	NitroDome stands at the forefront of gaming innovation, blending classic gaming experiences with the new era of digital ledger technologies. Beyond delivering pulse-racing games, NitroDome features a marketplace for digital collectibles, a platform for trading these collectibles, and unique, platform-exclusive titles.
Audit Scope	https://github.com/NitroDome/nitrodome_token https://github.com/NitroDome/nitrodome_vesting
Contracts in Scope	Branch: Main Contracts: -NitroDome.sol -Vesting.sol and interfaces
Commit Hash	b35fc13 dccf31a
Language	Solidity
Blockchain	Ethereum
Method	Manual Review, Automated Tools, Functional Testing
Review 1	10th April 2024 - 17th April 2024
Updated Code Received	27th April 2024
Review 2	29th April 2024
Fixed In	*https://github.com/NitroDome/nitrodome_vesting/commit/f526f1e5aeb4e596d39d0f3fb21c23d23c3c952f*

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	2	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	0	0	0

Checked Vulnerabilities

- Re-entrancy
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Unsafe type inference
- Style guide violation
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Solhint, Mythril, Slither, Solidity Static Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

A. Contract - Vesting.sol

High Severity Issues

A.1 User Vesting Values Are Overwritten When Address Repeats

Line **Function - addVestingScheduleForUser**

273-295

```
function addVestingScheduleForUser(
        address user,
        uint256 vestingScheduleIndex,
        uint256 totalAllocation
        ) public onlyOwner NotZeroAddress(user) {
        if (vestingScheduleIndex >= vestingScheduleCount)
        revert InvalidVestingScheduleIndex();
        if (totalAllocation == 0) revert AllocationIsZero();
        VestingSchedule storage vestingSchedule =
vestingSchedules[vestingScheduleIndex];
        uint256 initialClaimableAmount =
(vestingSchedule.initialClaimablePercent * totalAllocation) / 1e4;
        vestingSchedulesPerUser[user][vestingScheduleIndex] =
UserVestingSchedule({
        claimedAmount: 0,
        initialClaimableAmount: initialClaimableAmount,
        monthlyClaimableAmount: (vestingSchedule.monthlyVestingPercent *
totalAllocation) / 1e4,
        totalAllocation: totalAllocation
        });
        vestingSchedule.totalAllocation += totalAllocation;
        emit VestingScheduleAddedForUser(
        user,
        vestingScheduleIndex,
        totalAllocation
        );
        }
```

Description

In the **addVestingScheduleForUser** function, vesting schedules are assigned to users based on their address and a specific vesting schedule index. If multiple vesting schedules are assigned to the same user address with the same index, previous vesting data is overwritten.

This results in potential loss of data regarding earlier vesting schedules, where only the last entry is preserved. This can lead to incorrect token allocations and could potentially be exploited if the overwriting behavior is not anticipated by administrators, leading to unintended financial consequences.

Remediation

To remedy this issue, consider implementing one or more of the following changes:

- **Map Vesting by Unique IDs:** Instead of using just the user address and vesting schedule index, introduce a unique identifier for each vesting entry. This would prevent overwriting and allow multiple schedules per user.
- **Append Vesting Schedules:** Change the structure to allow an array of vesting schedules per user. This way, each new schedule is appended rather than replacing the existing one.
- **Validation Logic:** Before setting a new vesting schedule, check if one already exists and handle it appropriately, either by rejecting the new entry, merging the schedules, or by allowing the administrator to explicitly choose to overwrite.

Status

Resolved

B. Contract - NitroDome.sol

Medium Severity Issues

A.2 Dependency on Manual Token Transfer for Claim Functionality

Line	Function - NitroDome.sol
N/A	N/A

Description

The current implementation of the vesting contract requires that tokens be manually transferred to the contract by an administrator or other authorized entity before users can begin claiming their allocated tokens. This design introduces a dependency on human intervention and increases the risk of delays or errors in the token distribution process. Users are unable to claim any tokens until the contract itself holds enough tokens to cover these claims, potentially leading to dissatisfaction and a lack of trust if the tokens are not transferred in a timely manner or if the transfer is forgotten or mishandled.

Remediation

- **Automate Token Transfers:** If feasible, automate the transfer of tokens to the vesting contract through the use of smart contract functions that can be called to transfer tokens as soon as they are available or based on specific triggers.
- **Integration with Treasury Management:** Integrate the vesting contract with a treasury management system where tokens are automatically allocated and transferred based on preset rules and schedules.

Status

Acknowledged

A.3 Lack of Slippage Protection in Token Swap Function

Line	Function - _swapTokensForEth

370-380	function burn(uint256 tokenId) external {
	uint256 subcollectionIdx = getSubcollectionIdxByTokenId(tokenId);
	bool burnEnabled = subcollections[subcollectionIdx].burnEnabled;
	require(_isApprovedOrOwner(_msgSender(), tokenId), "Not token owner nor approved");
	require(burnEnabled, "Burn disabled");
	_burn(tokenId);
	delete _subIdByTokenId[tokenId];
	if (bytes(_tokenCIDs[tokenId]).length != 0) {
	delete _tokenCIDs[tokenId];
	}
	}

Description

In the provided function **_swapTokensForEth**, the contract swaps its own tokens for Ether using Uniswap V2's router interface. The code specifies a **minAmount** of 0 for the amount of ETH to be received in return. This setting does not protect against slippage, which is the difference between the expected price of a trade and the executed price that can occur during periods of high volatility or low liquidity.

Remediation

To mitigate the risks associated with high slippage, consider the following:

- **Implement Slippage Protection:** Modify the function to include a minimum amount parameter that is dynamically calculated based on the current market rate and an acceptable slippage percentage. For example, if the acceptable slippage is 1%, calculate the minimum amount as 99% of the expected ETH return.

Status

Acknowledged

Informational Issues

A.4 Zero Cliff Period in Public Sale Vesting Schedule

Line	Function - deploy.js - Vesting deployment script
18-23	{ name: "Public", unlockedAtTGEPercent: 1500, // 15% cliff: 0, // 1 month monthlyReleasePercent: 1417, // 14.17% (last one will be less due to rounding) },

Description

The deployment script specifies the vesting schedule for the Public Sale with a cliff period set to zero months.

Remediation

While a zero cliff period may be strategically chosen to attract certain investors or due to other business considerations, it is essential to assess and align this decision with the overall token economics and distribution strategy. Consider the following options:

- **Reevaluation of the Cliff Period:** If the aim is to ensure a more gradual distribution to support price stability, introducing a cliff period of 1-3 months might be beneficial.

Status

Acknowledged

Functional Tests

Some of the tests performed are mentioned below:

NitroDome-

- ✓ Should correctly initialize with valid parameters
- ✓ Should correctly set initial values
- ✓ Should revert with InvalidArrayLength if arrays lengths don't match
- ✓ Should revert if the token address is the zero address
- ✓ Should correctly set the owner
- ✓ Should revert with Inactive if contract is not active

Vesting-

- ✓ Should set the vesting schedules correctly from deployment
- ✓ Should add vesting schedules correctly
- ✓ Should add a new vesting schedule successfully and increment the count
- ✓ Should add a user vesting schedule successfully
- ✓ Should revert if a non-owner tries to add a user vesting schedule
- ✓ Should revert if an invalid vesting schedule index is provided
- ✓ Should update the vesting schedule's total allocation correctly
- ✓ Should successfully add multiple vesting schedules for multiple users
- ✓ Should revert if a non-owner attempts to add vesting schedules for users
- ✓ Should revert if the lengths of the input arrays don't match
- ✓ Should set the vesting schedule for a user
- ✓ Should allow users to claim tokens after cliff period
- ✓ Should not allow claiming before cliff period
- ✓ Should return an array of default vesting schedules for users with no vesting schedules
- ✓ Should return the correct vesting schedules for a user with assigned schedules
- ✓ Should correctly handle multiple vesting schedules for multiple users without mixups

Functional Tests

- Should return an array of zeros for a user with no vesting schedules
- Should return correct claimable amounts for a user with assigned vesting schedules
- Should return correct claimable amounts for multiple users with assigned vesting schedules
- Should allow owner to emergency withdraw tokensShould correctly initialize with valid parameters

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the NitroDome. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in NitroDome smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of NitroDome smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of NitroDome to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2024

For

NITRO DOME

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com