Nitro League

Smart Contract Audit Report

October 28, 2022

Introduction

1. About Nitro League

Nitro League, an immersive racing world is the ultimate combination of GameFi, DeFi, and NFT. Nitro League is a dynamic hub for collectors, investors, and artists built on a robust virtual economy.

Nitro facilitates ownership, customization, collection, and trade for unique NFT cars and parts. Players may hone their racing skills and earn rewards for their performance. Anyone can fulfill their dream of becoming a racer in the Nitro League.

Nitro League has devised a fully functional virtual economy that enables players to buy land and buildings and build cars. Nitro League facilitates players to expand their guilds and empires.

Visit https://www.nitroleague.com/ to know more about it.

2. About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

Documentation Details

The Nitro League team has provided the following doc for audit:

1. https://nitroblockchain.notion.site/nitroblockchain/Standard-Contract-Nitro-League-9efedf36fcec419fac5bca5acaa3ee8f
2. https://docs.google.com/document/d/1bDHuZuAevdwHT42_lNMfA9Yn1D7NSRZ4MtJBZfU6SoQ/edit

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project, starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors, including -

1. Structural analysis of the smart contract is checked and verified.
2. An extensive automated testing of all the contracts under scope is conducted.
3. Line-by-line Manual Code review is conducted to evaluate, analyze and identify the potential security risks in the contract.
4. Evaluation of the contract's intended behavior and the documentation shared is an imperative step to verify the contract behaves as expected.
5. For complex and heavy contracts, adequate integration testing is conducted to ensure that contracts perform acceptably while interacting.
6. Storage layout verifications in the upgradeable contract are a must.
7. An important step in the audit procedure is highlighting and recommending better gas optimization techniques in the contract.

Audit Details

- Project Name: Nitro League
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github link: https://github.com/sl2-studio/nitro-league-blockchain
- Branch: feedback_candidate, audit_candidate
- Commit hash: 0079ab70e81f4000ef6d6a9828289d0255696ab4(audit_candidate)
- Commit hash (Final): 9d3885c6da27e67fd9e597bb4d59e4662db13c7c(feedback_candidate)
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The audit's focus was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level Reference

Every issue in this report were assigned a severity level from the following:

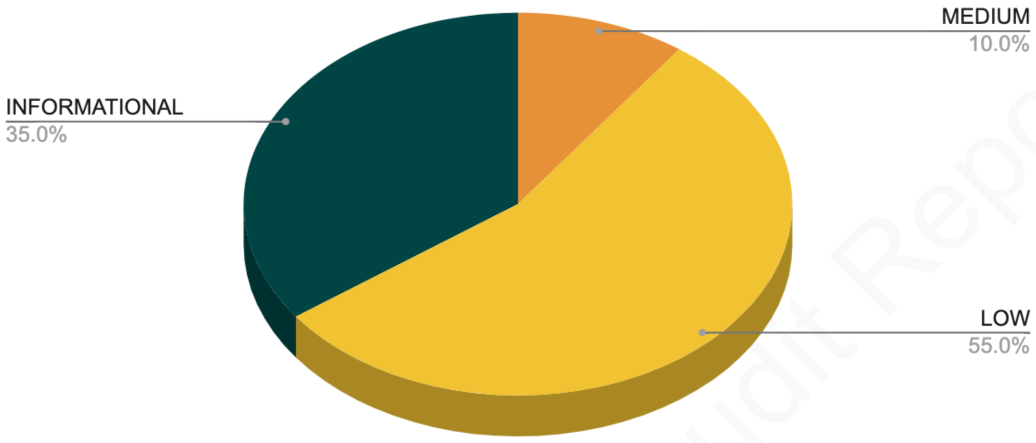
High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

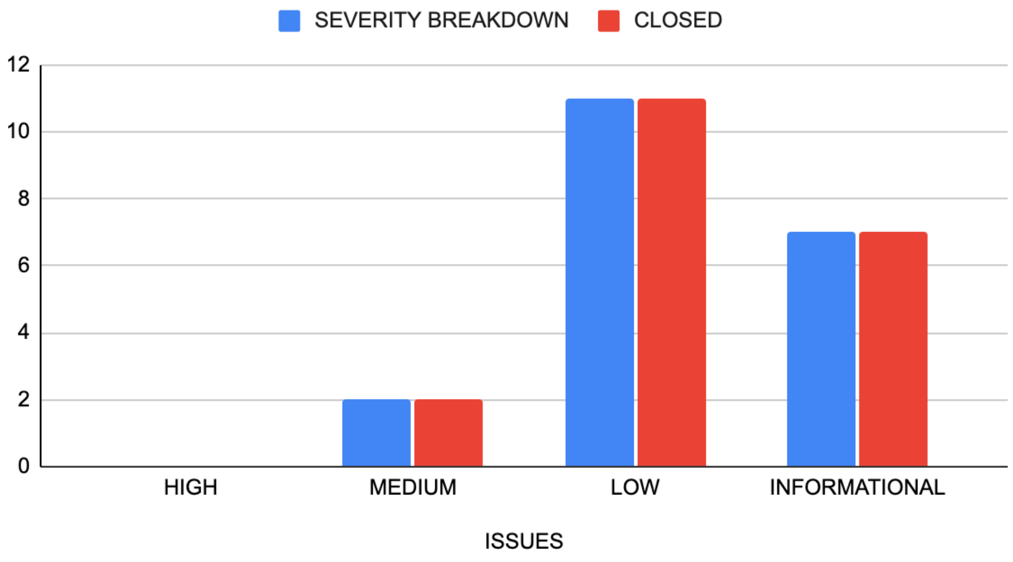
Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	-	-
Closed	-	2	10
Acknowledged	-	-	1

SEVERITY



SEVERITY BREAKDOWN vs CLOSED



Finding Overview

S.no	Findings	Risk	Status
01	minPlayers can be greater than maxPlayers if wrongly initialized. Can lead to unwanted behavior	Medium	Closed
02	Wrong function logic leads to invalid emission of events	Medium	Closed
03	Costly loop operations could be avoided	Low	Closed
04	Coding Style Issues in the Contract	Informational	Closed
05	Commented codes must be wiped out before deployment	Informational	Closed
06	Unlocked Pragma statements found in the contracts	Informational	Closed
07	claimRewards() function should only be executable in AWARDED state.	Low	Closed
08	No input validations to avoid array length mismatch	Low	Closed
09	Costly Loop operations found	Low	Closed
10	Unlocked Pragma statements found in the contracts	Informational	Closed
11	No Events emitted after imperative State Variable modification	Low	Closed
12	Absence of input validations after important state modifications	Low	Closed
13	External Visibility should be preferred	Low	Closed
14	Coding Style Issues in the Contract	Informational	Closed
15	Unlocked Pragma statements found in the contracts	Informational	Closed
16	Absence of Zero Address validation	Low	Closed
17	External Visibility should be preferred	Low	Closed
18	Unlocked Pragma statements found in the contracts	Informational	Closed
19	The reset of mintCount by the owner does not follow the intended procedure	Low	Acknowledged
20	setDailyMintLimit() function doesn't validate the arguments.	Low	Closed

Contract Name: Race

High Severity Issues

No issues were found.

Medium Severity Issues

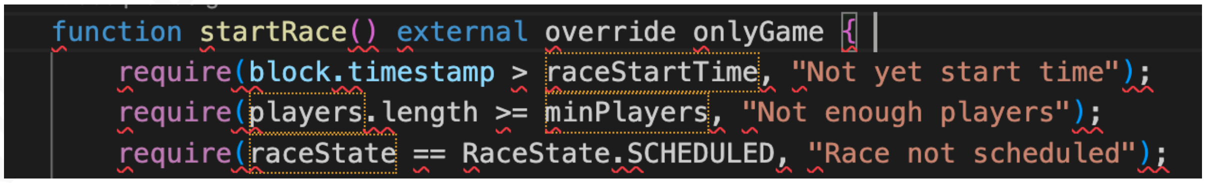
1. **minPlayers can be greater than maxPlayers if wrongly initialized. Can lead to unwanted behavior Line no - 173, 174**

   Description:
   During the manual review, it was found that neither the Race contract nor the RaceFactory contract includes adequate input validations on the arguments passed for the constructor.

   As per the current design, there is no input validation statement to verify if the minPlayers is actually smaller than the maxPlayers state variable. Keeping in mind the immutable nature of smart contracts, it's extremely imperative to include proper validations that ensure imperative state variables are initialized with only the right values.

   Wrong initialization (if minPlayers is greater than maxPlayers) of the above-mention state variables could lead to the following unwanted scenarios:

   a. startRace() function won't be executed unless the max number of players is stored instead of checking the min number of players.



```
function startRace() external override onlyGame {
    require(block.timestamp > raceStartTime, "Not yet start time");
    require(players.length >= minPlayers, "Not enough players");
    require(raceState == RaceState.SCHEDULED, "Race not scheduled");
```

   b. Functions like addPlayers() or joinRace() will also behave unexpectedly since they, too rely on maxPlayers state variable.

   Recommendation:

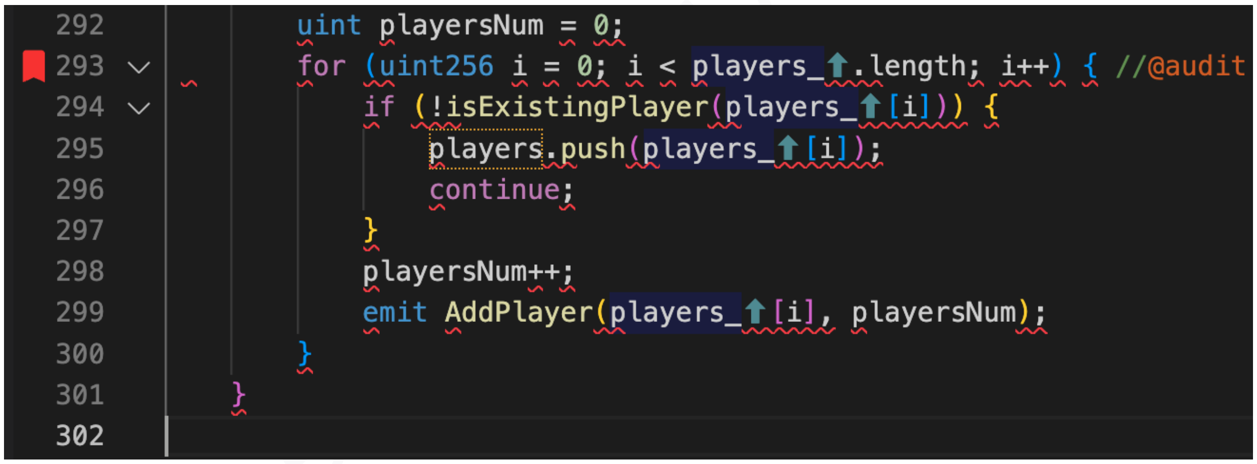
It is highly recommended to provide adequate input validations to functions/constructors to ensure only the right values are passed for updating the imperative state variables.

Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

2. **Wrong function logic leads to invalid emission of events**
   Line no: 294-299

Description:
The addPlayers() function in the contract includes a wrong implementation of the for loop.

As per the current design of the function, the loop first checks if the player address passed as an argument is already existing in the protocol. And only when the player address is unique, it executes further and adds a new player address in the players array.



```
292         uint playersNum = 0;
293         for (uint256 i = 0; i < players_.length; i++) { //@audit
294             if (!isExistingPlayer(players_[i])) {
295                 players.push(players_[i]);
296                 continue;
297             }
298             playersNum++;
299             emit AddPlayer(players_[i], playersNum);
300         }
301     }
302
```

However, once it updates the players array, it executes the continue keyword, which makes the execution flow skip the remaining block of code and start the next iteration of the loop immediately.

This leads to an undesirable scenario where lines 299 and 300 are never executed for every new player added, and this lead to the wrong emission of events with wrong values.

This could also lead to a major concern if the events are being tracked and stored off-chain since the data will not be accurate and reliable.

Recommendation:

The function must be adequately redesigned to emit valid events and ensure the right flow of execution.

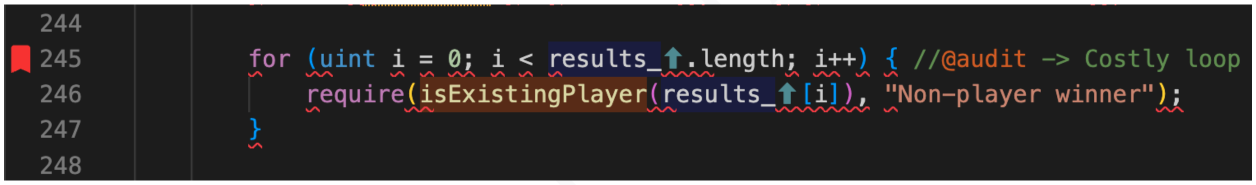
Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Low Severity Issues

1. **Costly loop operations could be avoided**
   Line no - 245-247, 293-297

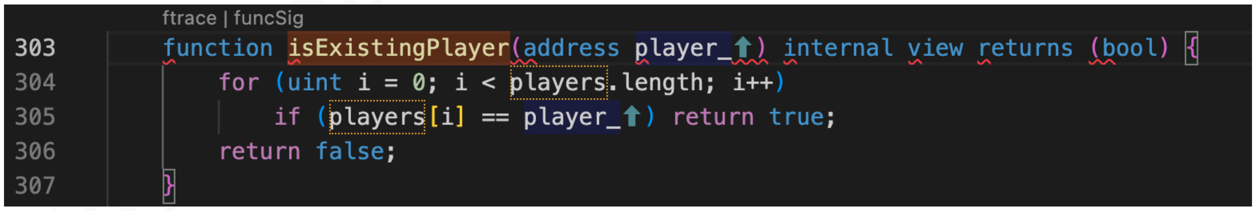
Description:
During the manual review of the contract, it was found that some functions include extremely costly loop operations.

For instance, the endRace and addPlayers() function calls the isExistingPlayer() function inside a loop to verify if a given player address exists in the protocol.



```
244
245        for (uint i = 0; i < results_.length; i++) { //@audit -> Costly loop
246            require(isExistingPlayer(results_[i]), "Non-player winner");
247        }
248
```

However, the isExistingPlayer() function includes a loop within its own body to provide this information. This makes function execution quite expensive in a scenario where there is a comparatively higher number of players in the contract.



```
ftrace | funcSig
function isExistingPlayer(address player_) internal view returns (bool) {
    for (uint i = 0; i < players.length; i++)
        if (players[i] == player_) return true;
    return false;
}
```

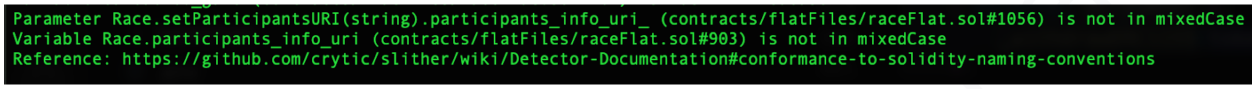
Recommendation:
The use of mappings could be introduced to identify existing or non-existing player addresses in the contract instead of looping over the entire player address list.

Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Recommendation / Informational

1. Coding Style Issues in the Contract

Description:
Code readability of a Smart Contract is largely influenced by the Coding Style issues and, in some specific scenarios, may lead to bugs in the future.



```
Parameter Race.setParticipantsURI(string).participants_info_uri_ (contracts/flatFiles/raceFlat.sol#1056) is not in mixedCase
Variable Race.participants_info_uri (contracts/flatFiles/raceFlat.sol#903) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

During the automated testing, it was found that the Race contract had quite a few code-style issues. Please follow this link to find details on naming conventions in solidity code.

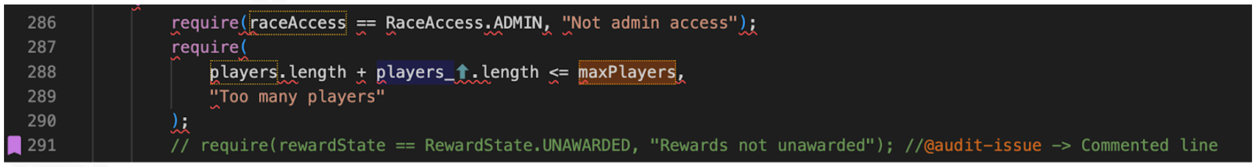
Recommendation:
Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

2. Commented codes must be wiped-out before deployment

Description:
The Race contract includes a few commented lines in the addPlayers() function.

This badly affects the readability of the code.



```
        require(raceAccess == RaceAccess.ADMIN, "Not admin access");
        require(
            players.length + players_.length <= maxPlayers,
            "Too many players"
        );
        // require(rewardState == RewardState.UNAWARDED, "Rewards not unawarded"); //@audit-issue -> Commented line
```

Recommendation:
If these instances of code are not required in the current version of the contract, then the commented codes must be removed before deployment.

Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

3. **Unlocked Pragma statements found in the contracts**
   Line no: 2

   Description:
   During the code review, it was found that the contracts included unlocked pragma solidity version statements.

   It’s not considered a better practice in Smart contract development to do so, as it might lead to accidental deployment to a version with unfixed bugs.

   Recommendation:
   It’s always recommended to lock pragma statements to a specific version while writing contracts.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Contract Name: RewardManager

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **claimRewards() function should only be executable in AWARDED state.**

   Description:
   The current design of the RewardManager indicates that the claim of rewards should only be possible after the positions have been accurately updated by the owner.

   In other words, the claimRewards() function should be executable after the rewardSate of the contract is updated to AWARDED state.

   However, during the review, it was found that no such checkpoint is included in the claimReward function.

   Although the function has been assigned an onlyOwner modifier, it's strongly recommended to ensure that an external function is executable only in the right conditions.

   Recommendation:
   Valid require statements must be included to ensure accurate function execution.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

2. **No input validations to avoid array length mismatch**
   Line no: 101 to 157

   Description:
   The depositRewards() function allows 6 different array arguments for the function where each of them goes through a for loop iteration.

   However, the function doesn't include any checkpoints to ensure that there is no array length mismatch while calling this function.

   The absence of such validations might lead to unwanted function execution and is, therefore not an adequate function design.

   Recommendation:
   Functions should include adequate require statements to validate the arguments being passed.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

3. **Costly Loop operations found**
   Line no: 160-214

   Description:
   The RewardManager contract includes a for loop in its claimRewards() function that uses a state variable like .length of a non-memory array, i.e, positionRewards, in the condition of the for loops.

   As a result, these state variables consume a lot more extra gas for every iteration of the for loop. The following function includes such loops at the mentioned lines:

   - **claimRewards() at Line 335**

   Recommendation:
   It's quite effective to use a local variable instead of a state variable like .length in a loop. This will be a significant step in optimizing gas usage for this function.

For instance,

```
local_variable = positionRewards.length;

 for (uint256 i = 0; i < local_variable; i++) {
       ….
      // for loop body
 }
```

Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Recommendation / Informational

1. **Unlocked Pragma statements found in the contracts**
   Line no: 2

   Description:
   During the code review, it was found that the contracts included unlocked pragma solidity version statements.

   It's not considered a better practice in Smart contract development to do so as it might lead to accidental deployment to a version with unfixed bugs.

   Recommendation:
   It's always recommended to lock pragma statements to a specific version while writing contracts.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Contract Name: RaceEvent

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **No Events emitted after imperative State Variable modification**
   Line no: 269-275

   Description:
   Functions that update an imperative arithmetic state variable contract should emit an event after the state modification.
   The setWinningPositions function modifies crucial arithmetic state variables, i.e, winningPositions RaceEvent contract but doesn't emit any event after that:

   Since there is no event emitted on updating these variables, it might be difficult to track it off-chain.

   Recommendation:
   An event should be fired after changing crucial arithmetic state variables.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

2. **Absence of input validations after important state modifications**
   Line no: 123-146, 240-257

   Description:
   During the review, it was found that some important functions do not include any input validations before updating the crucial state variables of the contract.

Some of the important ones are

- constructor():
    - The constructor lacks zero address validations on imperative addresses like nitroleague, rewardfactory etc.

- createRace():
    - The createRace() function lacks zero address validation as well as some crucial uint256 validations passed in the int_settings array argument.

Recommendation:
An event should be fired after changing crucial arithmetic state variables.

Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

3. **External Visibility should be preferred**

Explanation:
Functions that are never called throughout the contract should be marked as external visibility instead of public visibility.
This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as external within the contract:

- **getRewardState()**

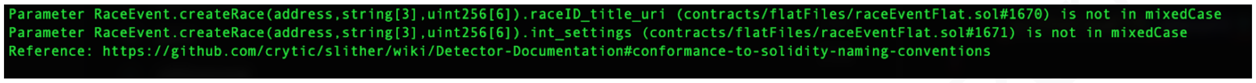
Recommendation:
If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Recommendation / Informational

1. **Coding Style Issues in the Contract**

   Description:
   Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.



```
Parameter RaceEvent.createRace(address,string[3],uint256[6]).raceID_title_uri (contracts/flatFiles/raceEventFlat.sol#1670) is not in mixedCase
Parameter RaceEvent.createRace(address,string[3],uint256[6]).int_settings (contracts/flatFiles/raceEventFlat.sol#1671) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

   During the automated testing, it was found that the RaceEvent contract had quite a few code-style issues. Please follow this link to find details on naming conventions in solidity code.

   Recommendation:
   Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

2. **Unlocked Pragma statements found in the contracts**
   Line no: 2

   Description:
   During the code review, it was found that the contracts included unlocked pragma solidity version statements.

   It's not considered a better practice in Smart contract development to do so, as it might lead to accidental deployment to a version with unfixed bugs.

   Recommendation:
   It's always recommended to lock pragma statements to a specific version while writing contracts.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Contract Name: NitroLeague

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **Absence of Zero Address validation**

   Description:
   The NitroLeague contract includes a few functions that update some imperative addresses in the contract.

   However, during the automated testing of the contact, it was found that no Zero Address Validation was performed on the following functions before updating the state variables of the contract:

   - setRewardFactory()
   - setTreasuryWallet()

   Recommendation:
   A require statement should be included in such functions to ensure no zero address is passed in the arguments.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

2. **External Visibility should be preferred**

   Explanation:
   Functions that are never called throughout the contract should be marked as external visibility instead of public visibility.
   This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as external within the contract:

- getGame()

Recommendation:
If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Recommendation / Informational

1. **Unlocked Pragma statements found in the contracts**
   Line no: 2

   Description:
   During the code review, it was found that the contracts included unlocked pragma solidity version statements.

   It's not considered a better practice in Smart contract development to do so, as it might lead to accidental deployment to a version with unfixed bugs.

   Recommendation:
   It's always recommended to lock pragma statements to a specific version while writing contracts.

   Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Final Audit

Low Severity Issues

2 New issues were found in Mintable contract

1. **The reset of mintCount by the owner does not follow the intended procedure.**
   Line no - 189-192

   Description:
   As per the current design of the contract, the reset of mintcount should only happen if the daily mint is exceeded and the lastUpdateTime greater than 24 hours.

   However, the **resetMintCount()** function in the contract allows the owner to bypass these checkpoints and directly reset the mint count for any token contract.

   This symbolizes excessive power in the hands of owners as the owner address can choose to stop minting at any time by resetting the mint count. This could also lead to a high severity issue if the owner keys are in the hands of the wrong actor in case the owner address is an EOA instead of a multi-sig.

   Recommendation:
   Imperative functions that influence the flow of tokens or minting of tokens should follow one specific procedure, even if the caller is an owner.

   If the above-mentioned design is intentional, it must be made clear to the community members that the owner has such control over the minting procedure of the tokens.

   Acknowledged (October 28th, 2022): The team has acknowledged the issue, and it is intended behavior.

2. **setDailyMintLimit() function doesn't validate the arguments.**
   Line no- 176-180

   Description:
   The setDailyMintLimit() function is responsible for updating the maximum daily mint limits of tokens.

   However, the function doesn't include any input validations on the uint256 arguments passed to it.

This could lead to an unwanted scenario where any invalid input passed to this function might stop new tokens from being as well.

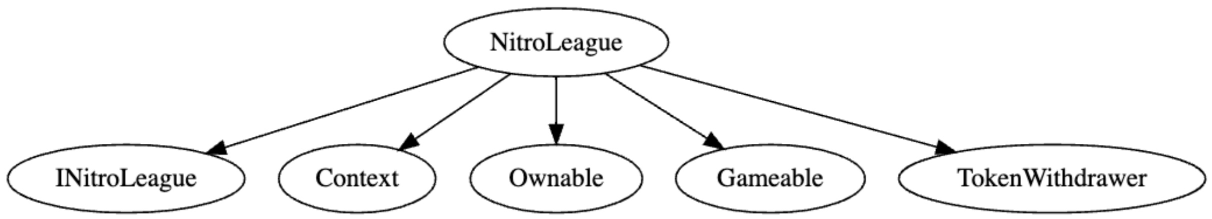
Recommendation:
Although the function is marked with an **onlyOwner** modifier, it's recommended to include adequate input validations for the function to ensure only the right arguments are passed to it.

Alternatively, functions that update any important arithmetic state variables can also include a maximum or minimum threshold check within its body to ensure the state variable is only updated with a value that falls in the pre-defined range of the contract.

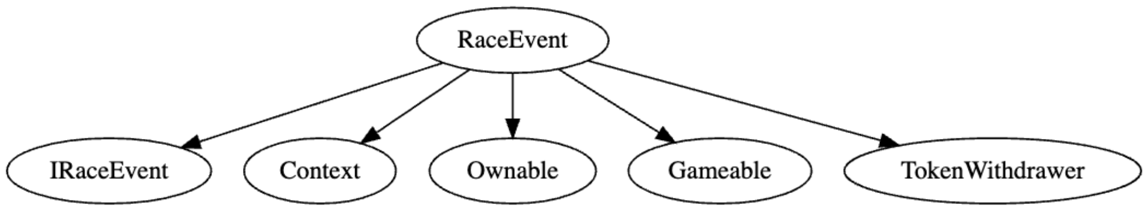
Amended (October 28th, 2022): The team has fixed the issue, and it is no longer present in commit 9d3885c6da27e67fd9e597bb4d59e4662db13c7c.

Additional Details - Inheritance Graph

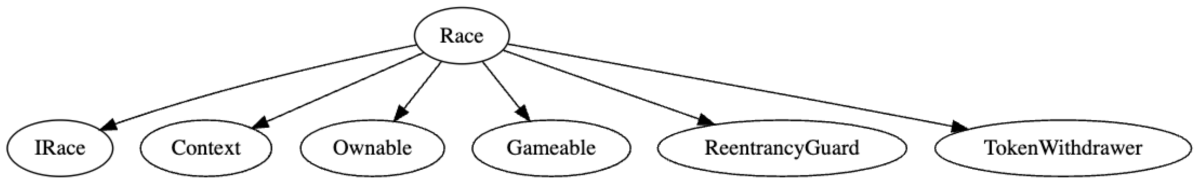
1. NitroLeague



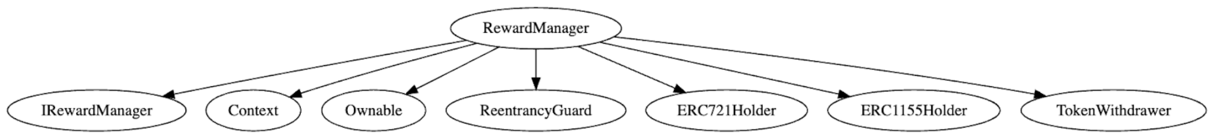
2. RaceEvent



3. Race

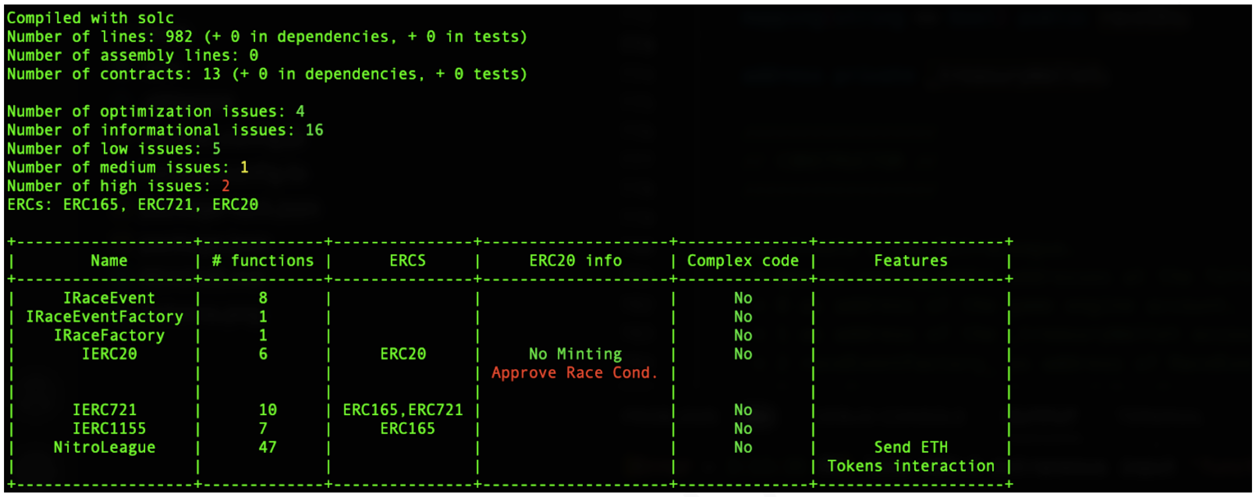


4. RewardManager



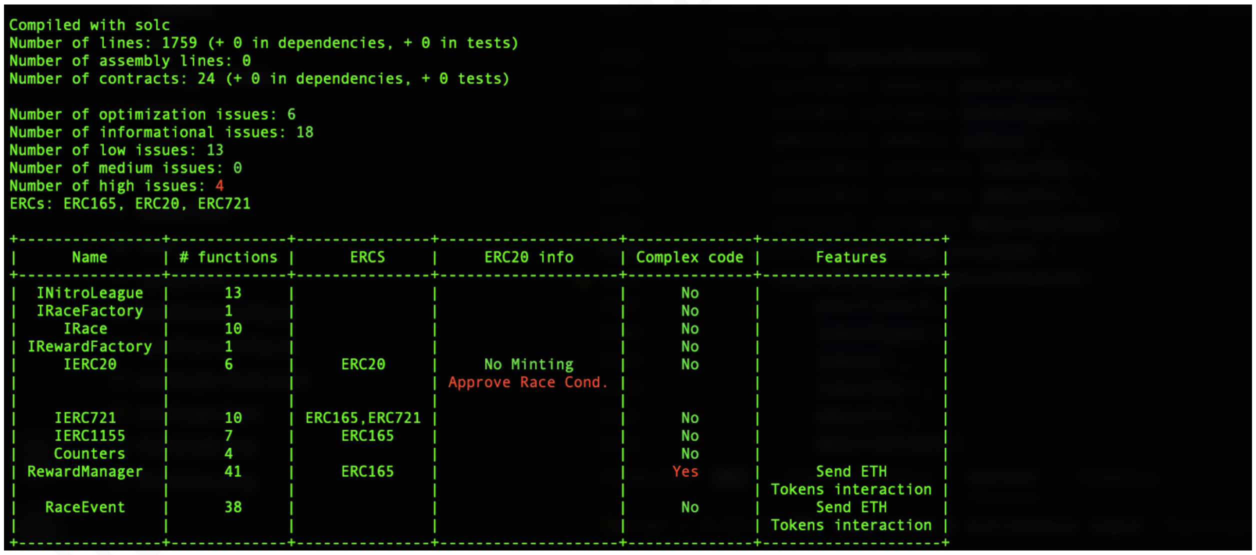
Automated Audit Result

1. NitroLeague



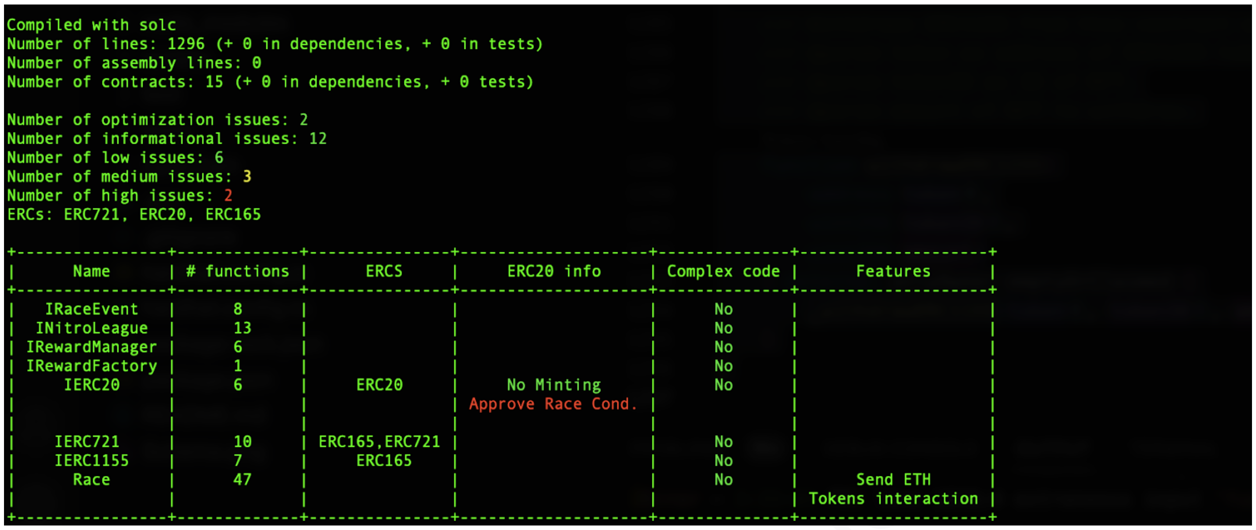
```
Compiled with solc
Number of lines: 982 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 13 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 4
Number of informational issues: 16
Number of low issues: 5
Number of medium issues: 1
Number of high issues: 2
ERCs: ERC165, ERC721, ERC20

+-------------------+-------------+---------------+--------------------+--------------+--------------------+
|        Name       | # functions |      ERCS     |     ERC20 info     | Complex code |      Features      |
+-------------------+-------------+---------------+--------------------+--------------+--------------------+
|     IRaceEvent    |      8      |               |                    |      No      |                    |
| IRaceEventFactory |      1      |               |                    |      No      |                    |
|    IRaceFactory   |      1      |               |                    |      No      |                    |
|       IERC20      |      6      |     ERC20     |     No Minting     |      No      |                    |
|                   |             |               | Approve Race Cond. |              |                    |
|                   |             |               |                    |              |                    |
|      IERC721      |      10     | ERC165,ERC721 |                    |      No      |                    |
|      IERC1155     |      7      |     ERC165    |                    |      No      |                    |
|    NitroLeague    |      47     |               |                    |      No      |      Send ETH      |
|                   |             |               |                    |              | Tokens interaction |
+-------------------+-------------+---------------+--------------------+--------------+--------------------+
```

2. RaceEvent



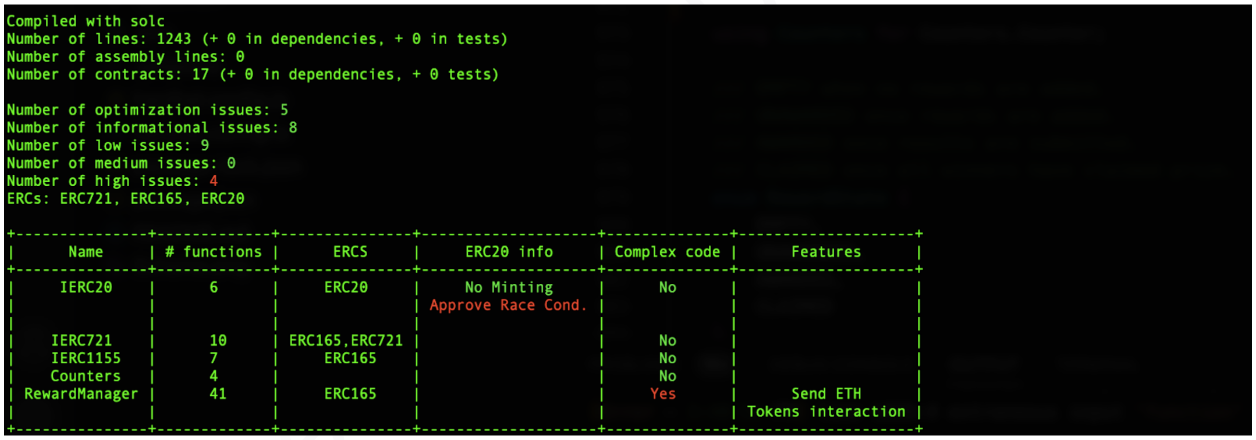
```
Compiled with solc
Number of lines: 1759 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 24 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 6
Number of informational issues: 18
Number of low issues: 13
Number of medium issues: 0
Number of high issues: 4
ERCs: ERC165, ERC20, ERC721

+----------------+-------------+---------------+--------------------+--------------+--------------------+
|      Name      | # functions |      ERCS     |     ERC20 info     | Complex code |      Features      |
+----------------+-------------+---------------+--------------------+--------------+--------------------+
|  INitroLeague  |      13     |               |                    |      No      |                    |
|  IRaceFactory  |      1      |               |                    |      No      |                    |
|     IRace      |      10     |               |                    |      No      |                    |
| IRewardFactory |      1      |               |                    |      No      |                    |
|     IERC20     |      6      |     ERC20     |     No Minting     |      No      |                    |
|                |             |               | Approve Race Cond. |              |                    |
|                |             |               |                    |              |                    |
|    IERC721     |      10     | ERC165,ERC721 |                    |      No      |                    |
|    IERC1155    |      7      |     ERC165    |                    |      No      |                    |
|    Counters    |      4      |               |                    |      No      |                    |
| RewardManager  |      41     |     ERC165    |                    |     Yes      |      Send ETH      |
|                |             |               |                    |              | Tokens interaction |
|   RaceEvent    |      38     |               |                    |      No      |      Send ETH      |
|                |             |               |                    |              | Tokens interaction |
+----------------+-------------+---------------+--------------------+--------------+--------------------+
```

3. Race



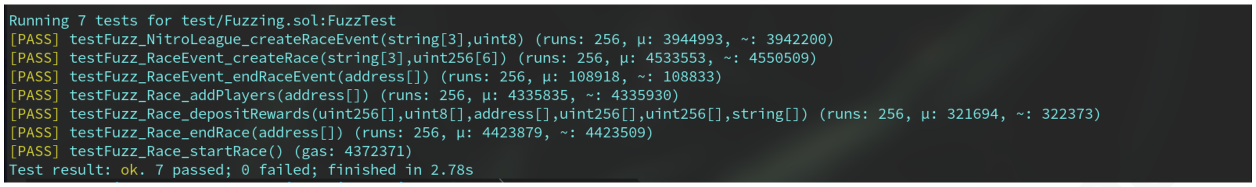
```
Compiled with solc
Number of lines: 1296 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 15 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 12
Number of low issues: 6
Number of medium issues: 3
Number of high issues: 2
ERCs: ERC721, ERC20, ERC165

+----------------+------------+--------------+--------------------+--------------+--------------------+
|      Name      | # functions |     ERCS     |     ERC20 info     | Complex code |      Features      |
+----------------+------------+--------------+--------------------+--------------+--------------------+
|   IRaceEvent   |      8      |              |                    |      No      |                    |
|  INitroLeague  |     13      |              |                    |      No      |                    |
| IRewardManager |      6      |              |                    |      No      |                    |
| IRewardFactory |      1      |              |                    |      No      |                    |
|     IERC20     |      6      |    ERC20     |     No Minting     |      No      |                    |
|                |             |              | Approve Race Cond. |              |                    |
|                |             |              |                    |              |                    |
|    IERC721     |     10      | ERC165,ERC721 |                    |      No      |                    |
|    IERC1155    |      7      |    ERC165    |                    |      No      |                    |
|      Race      |     47      |              |                    |      No      |      Send ETH      |
|                |             |              |                    |              | Tokens interaction |
+----------------+------------+--------------+--------------------+--------------+--------------------+
```

4. RewardManager

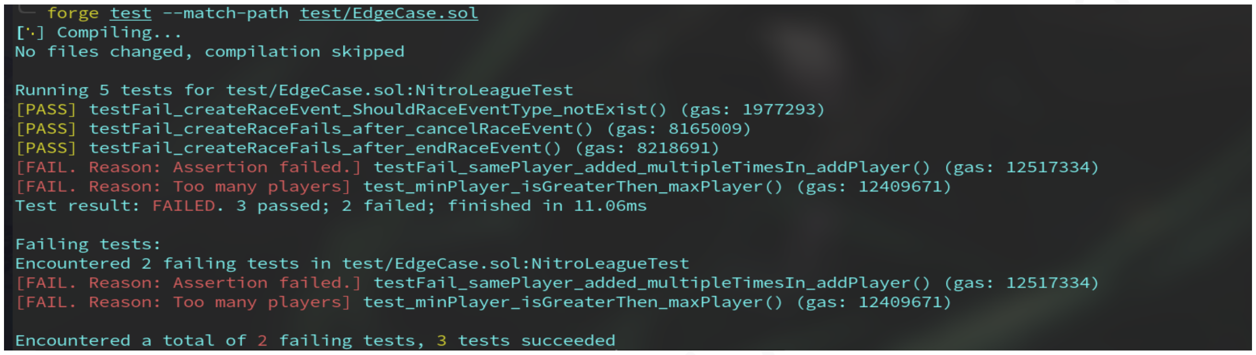


```
Compiled with solc
Number of lines: 1243 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 17 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 5
Number of informational issues: 8
Number of low issues: 9
Number of medium issues: 0
Number of high issues: 4
ERCs: ERC721, ERC165, ERC20

+---------------+-------------+---------------+--------------------+--------------+--------------------+
|      Name     | # functions |      ERCS     |     ERC20 info     | Complex code |      Features      |
+---------------+-------------+---------------+--------------------+--------------+--------------------+
|     IERC20    |      6      |     ERC20     |     No Minting     |      No      |                    |
|               |             |               | Approve Race Cond. |              |                    |
|               |             |               |                    |              |                    |
|    IERC721    |     10      | ERC165,ERC721 |                    |      No      |                    |
|    IERC1155   |      7      |     ERC165    |                    |      No      |                    |
|    Counters   |      4      |               |                    |      No      |                    |
| RewardManager |     41      |     ERC165    |                    |     Yes      |      Send ETH      |
|               |             |               |                    |              | Tokens interaction |
+---------------+-------------+---------------+--------------------+--------------+--------------------+
```

Fuzz Testing

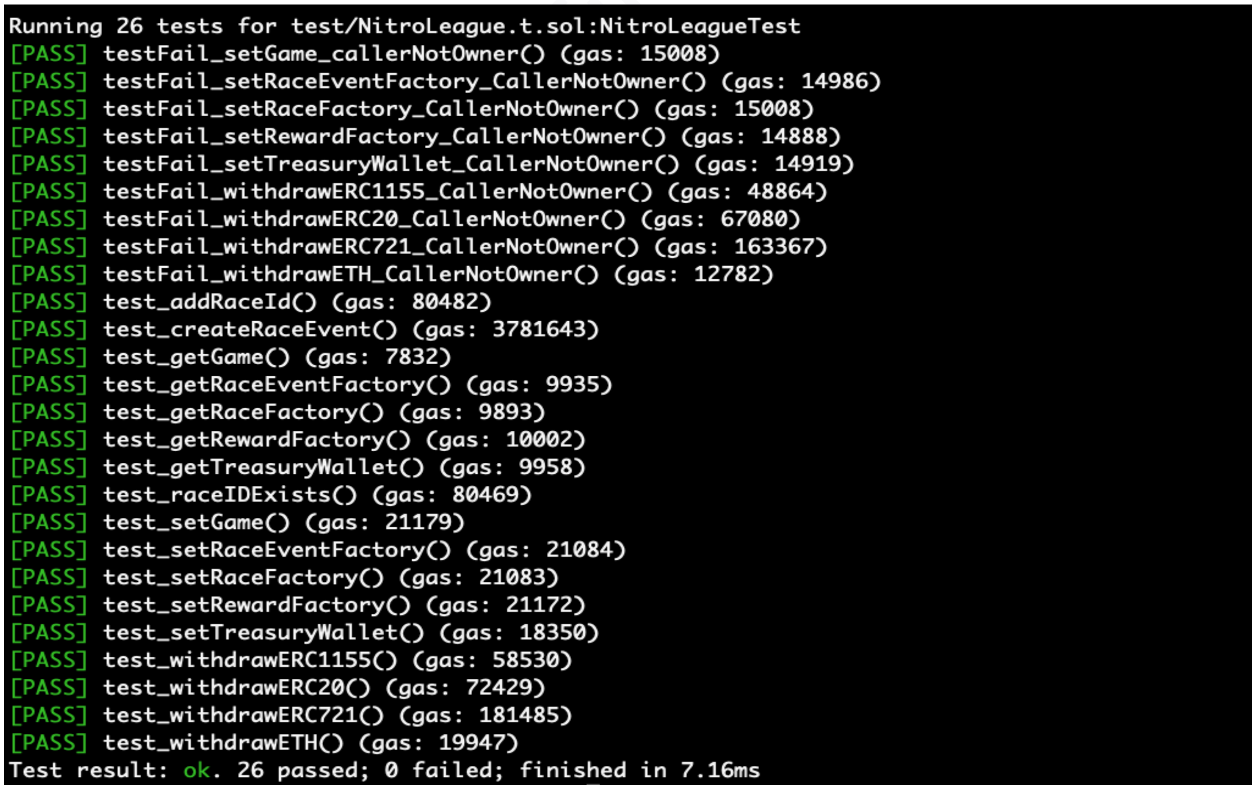


```
Running 7 tests for test/Fuzzing.sol:FuzzTest
[PASS] testFuzz_NitroLeague_createRaceEvent(string[3],uint8) (runs: 256, μ: 3944993, ~: 3942200)
[PASS] testFuzz_RaceEvent_createRace(string[3],uint256[6]) (runs: 256, μ: 4533553, ~: 4550509)
[PASS] testFuzz_RaceEvent_endRaceEvent(address[]) (runs: 256, μ: 108918, ~: 108833)
[PASS] testFuzz_Race_addPlayers(address[]) (runs: 256, μ: 4335835, ~: 4335930)
[PASS] testFuzz_Race_depositRewards(uint256[],uint8[],address[],uint256[],uint256[],string[]) (runs: 256, μ: 321694, ~: 322373)
[PASS] testFuzz_Race_endRace(address[]) (runs: 256, μ: 4423879, ~: 4423509)
[PASS] testFuzz_Race_startRace() (gas: 4372371)
Test result: ok. 7 passed; 0 failed; finished in 2.78s
```

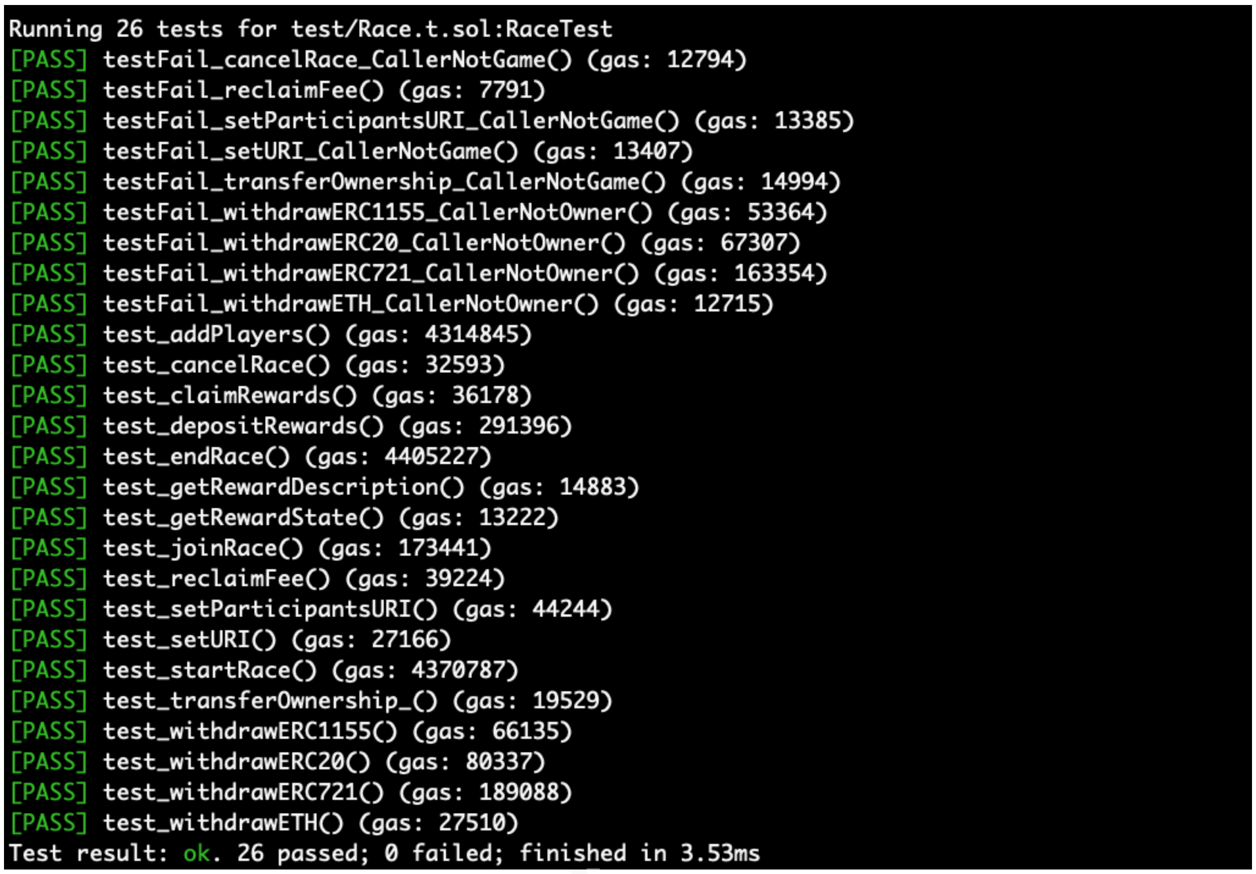


```
  forge test --match-path test/EdgeCase.sol
[⠊] Compiling...
No files changed, compilation skipped

Running 5 tests for test/EdgeCase.sol:NitroLeagueTest
[PASS] testFail_createRaceEvent_ShouldRaceEventType_notExist() (gas: 1977293)
[PASS] testFail_createRaceFails_after_cancelRaceEvent() (gas: 8165009)
[PASS] testFail_createRaceFails_after_endRaceEvent() (gas: 8218691)
[FAIL. Reason: Assertion failed.] testFail_samePlayer_added_multipleTimesIn_addPlayer() (gas: 12517334)
[FAIL. Reason: Too many players] test_minPlayer_isGreaterThen_maxPlayer() (gas: 12409671)
Test result: FAILED. 3 passed; 2 failed; finished in 11.06ms

Failing tests:
Encountered 2 failing tests in test/EdgeCase.sol:NitroLeagueTest
[FAIL. Reason: Assertion failed.] testFail_samePlayer_added_multipleTimesIn_addPlayer() (gas: 12517334)
[FAIL. Reason: Too many players] test_minPlayer_isGreaterThen_maxPlayer() (gas: 12409671)

Encountered a total of 2 failing tests, 3 tests succeeded
```

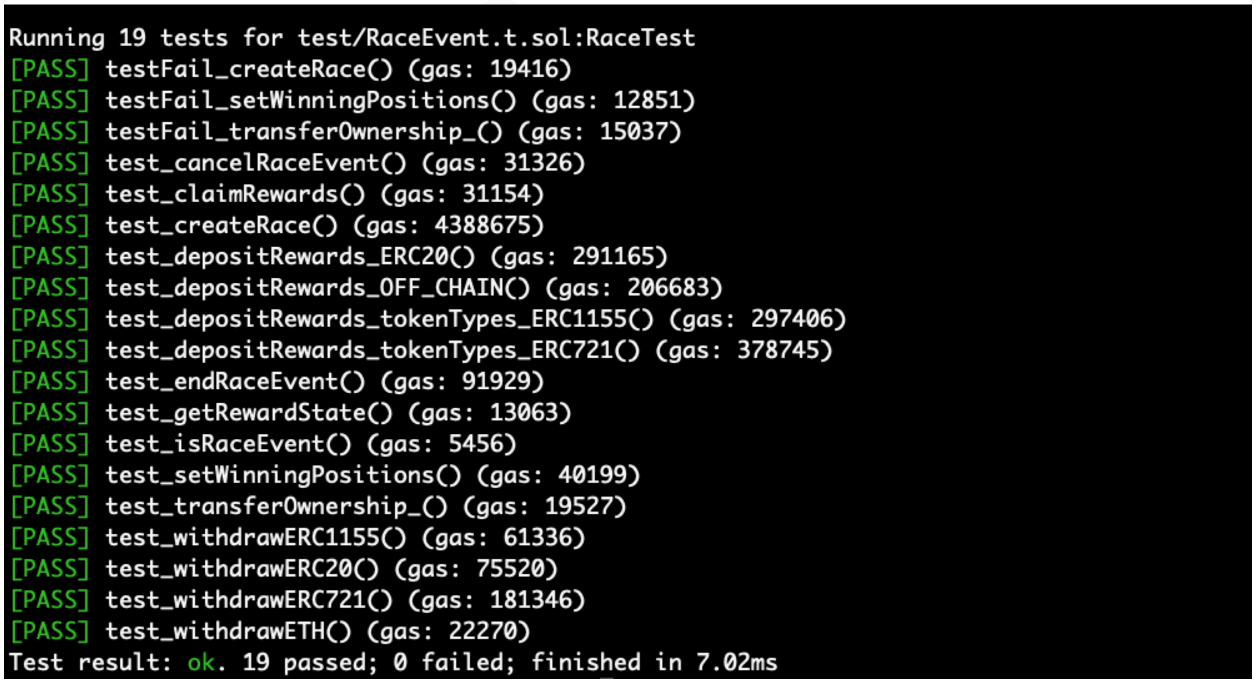
Auditor Test Cases



```
Running 26 tests for test/NitroLeague.t.sol:NitroLeagueTest
[PASS] testFail_setGame_callerNotOwner() (gas: 15008)
[PASS] testFail_setRaceEventFactory_CallerNotOwner() (gas: 14986)
[PASS] testFail_setRaceFactory_CallerNotOwner() (gas: 15008)
[PASS] testFail_setRewardFactory_CallerNotOwner() (gas: 14888)
[PASS] testFail_setTreasuryWallet_CallerNotOwner() (gas: 14919)
[PASS] testFail_withdrawERC1155_CallerNotOwner() (gas: 48864)
[PASS] testFail_withdrawERC20_CallerNotOwner() (gas: 67080)
[PASS] testFail_withdrawERC721_CallerNotOwner() (gas: 163367)
[PASS] testFail_withdrawETH_CallerNotOwner() (gas: 12782)
[PASS] test_addRaceId() (gas: 80482)
[PASS] test_createRaceEvent() (gas: 3781643)
[PASS] test_getGame() (gas: 7832)
[PASS] test_getRaceEventFactory() (gas: 9935)
[PASS] test_getRaceFactory() (gas: 9893)
[PASS] test_getRewardFactory() (gas: 10002)
[PASS] test_getTreasuryWallet() (gas: 9958)
[PASS] test_raceIDExists() (gas: 80469)
[PASS] test_setGame() (gas: 21179)
[PASS] test_setRaceEventFactory() (gas: 21084)
[PASS] test_setRaceFactory() (gas: 21083)
[PASS] test_setRewardFactory() (gas: 21172)
[PASS] test_setTreasuryWallet() (gas: 18350)
[PASS] test_withdrawERC1155() (gas: 58530)
[PASS] test_withdrawERC20() (gas: 72429)
[PASS] test_withdrawERC721() (gas: 181485)
[PASS] test_withdrawETH() (gas: 19947)
Test result: ok. 26 passed; 0 failed; finished in 7.16ms
```

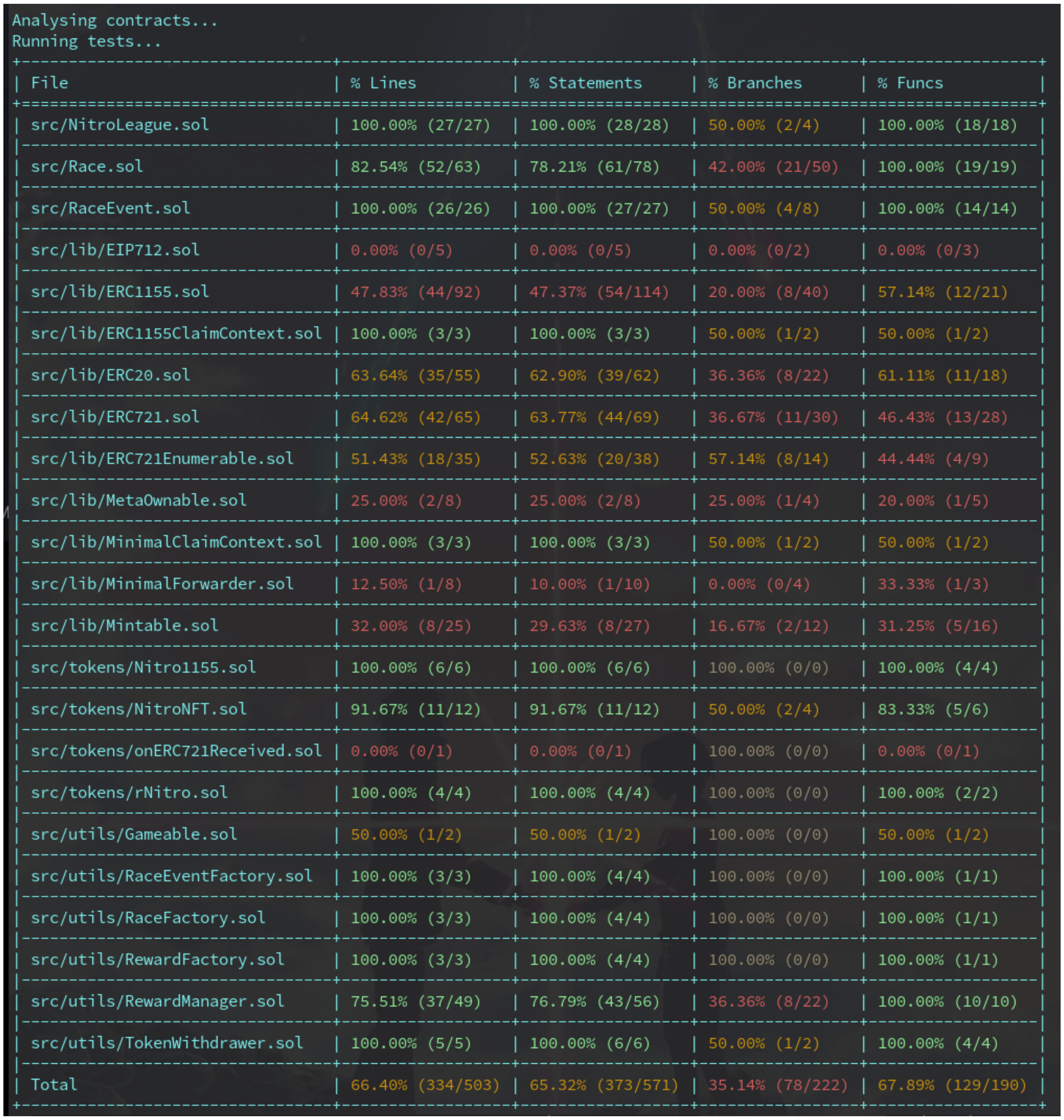


```
Running 26 tests for test/Race.t.sol:RaceTest
[PASS] testFail_cancelRace_CallerNotGame() (gas: 12794)
[PASS] testFail_reclaimFee() (gas: 7791)
[PASS] testFail_setParticipantsURI_CallerNotGame() (gas: 13385)
[PASS] testFail_setURI_CallerNotGame() (gas: 13407)
[PASS] testFail_transferOwnership_CallerNotGame() (gas: 14994)
[PASS] testFail_withdrawERC1155_CallerNotOwner() (gas: 53364)
[PASS] testFail_withdrawERC20_CallerNotOwner() (gas: 67307)
[PASS] testFail_withdrawERC721_CallerNotOwner() (gas: 163354)
[PASS] testFail_withdrawETH_CallerNotOwner() (gas: 12715)
[PASS] test_addPlayers() (gas: 4314845)
[PASS] test_cancelRace() (gas: 32593)
[PASS] test_claimRewards() (gas: 36178)
[PASS] test_depositRewards() (gas: 291396)
[PASS] test_endRace() (gas: 4405227)
[PASS] test_getRewardDescription() (gas: 14883)
[PASS] test_getRewardState() (gas: 13222)
[PASS] test_joinRace() (gas: 173441)
[PASS] test_reclaimFee() (gas: 39224)
[PASS] test_setParticipantsURI() (gas: 44244)
[PASS] test_setURI() (gas: 27166)
[PASS] test_startRace() (gas: 4370787)
[PASS] test_transferOwnership_() (gas: 19529)
[PASS] test_withdrawERC1155() (gas: 66135)
[PASS] test_withdrawERC20() (gas: 80337)
[PASS] test_withdrawERC721() (gas: 189088)
[PASS] test_withdrawETH() (gas: 27510)
Test result: ok. 26 passed; 0 failed; finished in 3.53ms
```



```
Running 19 tests for test/RaceEvent.t.sol:RaceTest
[PASS] testFail_createRace() (gas: 19416)
[PASS] testFail_setWinningPositions() (gas: 12851)
[PASS] testFail_transferOwnership_() (gas: 15037)
[PASS] test_cancelRaceEvent() (gas: 31326)
[PASS] test_claimRewards() (gas: 31154)
[PASS] test_createRace() (gas: 4388675)
[PASS] test_depositRewards_ERC20() (gas: 291165)
[PASS] test_depositRewards_OFF_CHAIN() (gas: 206683)
[PASS] test_depositRewards_tokenTypes_ERC1155() (gas: 297406)
[PASS] test_depositRewards_tokenTypes_ERC721() (gas: 378745)
[PASS] test_endRaceEvent() (gas: 91929)
[PASS] test_getRewardState() (gas: 13063)
[PASS] test_isRaceEvent() (gas: 5456)
[PASS] test_setWinningPositions() (gas: 40199)
[PASS] test_transferOwnership_() (gas: 19527)
[PASS] test_withdrawERC1155() (gas: 61336)
[PASS] test_withdrawERC20() (gas: 75520)
[PASS] test_withdrawERC721() (gas: 181346)
[PASS] test_withdrawETH() (gas: 22270)
Test result: ok. 19 passed; 0 failed; finished in 7.02ms
```

Test Coverage



```
Analysing contracts...
Running tests...
+---------------------------------+---------------------+---------------------+-------------------+-------------------+
| File                            | % Lines             | % Statements        | % Branches        | % Funcs           |
+=====================================================================================================================+
| src/NitroLeague.sol             | 100.00% (27/27)     | 100.00% (28/28)     | 50.00% (2/4)      | 100.00% (18/18)   |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/Race.sol                    | 82.54% (52/63)      | 78.21% (61/78)      | 42.00% (21/50)    | 100.00% (19/19)   |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/RaceEvent.sol               | 100.00% (26/26)     | 100.00% (27/27)     | 50.00% (4/8)      | 100.00% (14/14)   |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/EIP712.sol              | 0.00% (0/5)         | 0.00% (0/5)         | 0.00% (0/2)       | 0.00% (0/3)       |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/ERC1155.sol             | 47.83% (44/92)      | 47.37% (54/114)     | 20.00% (8/40)     | 57.14% (12/21)    |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/ERC1155ClaimContext.sol | 100.00% (3/3)       | 100.00% (3/3)       | 50.00% (1/2)      | 50.00% (1/2)      |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/ERC20.sol               | 63.64% (35/55)      | 62.90% (39/62)      | 36.36% (8/22)     | 61.11% (11/18)    |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/ERC721.sol              | 64.62% (42/65)      | 63.77% (44/69)      | 36.67% (11/30)    | 46.43% (13/28)    |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/ERC721Enumerable.sol    | 51.43% (18/35)      | 52.63% (20/38)      | 57.14% (8/14)     | 44.44% (4/9)      |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/MetaOwnable.sol         | 25.00% (2/8)        | 25.00% (2/8)        | 25.00% (1/4)      | 20.00% (1/5)      |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/MinimalClaimContext.sol | 100.00% (3/3)       | 100.00% (3/3)       | 50.00% (1/2)      | 50.00% (1/2)      |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/MinimalForwarder.sol    | 12.50% (1/8)        | 10.00% (1/10)       | 0.00% (0/4)       | 33.33% (1/3)      |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/lib/Mintable.sol            | 32.00% (8/25)       | 29.63% (8/27)       | 16.67% (2/12)     | 31.25% (5/16)     |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/tokens/Nitro1155.sol        | 100.00% (6/6)       | 100.00% (6/6)       | 100.00% (0/0)     | 100.00% (4/4)     |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/tokens/NitroNFT.sol         | 91.67% (11/12)      | 91.67% (11/12)      | 50.00% (2/4)      | 83.33% (5/6)      |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/tokens/onERC721Received.sol | 0.00% (0/1)         | 0.00% (0/1)         | 100.00% (0/0)     | 0.00% (0/1)       |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/tokens/rNitro.sol           | 100.00% (4/4)       | 100.00% (4/4)       | 100.00% (0/0)     | 100.00% (2/2)     |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/utils/Gameable.sol          | 50.00% (1/2)        | 50.00% (1/2)        | 100.00% (0/0)     | 50.00% (1/2)      |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/utils/RaceEventFactory.sol  | 100.00% (3/3)       | 100.00% (4/4)       | 100.00% (0/0)     | 100.00% (1/1)     |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/utils/RaceFactory.sol       | 100.00% (3/3)       | 100.00% (4/4)       | 100.00% (0/0)     | 100.00% (1/1)     |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/utils/RewardFactory.sol     | 100.00% (3/3)       | 100.00% (4/4)       | 100.00% (0/0)     | 100.00% (1/1)     |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/utils/RewardManager.sol     | 75.51% (37/49)      | 76.79% (43/56)      | 36.36% (8/22)     | 100.00% (10/10)   |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| src/utils/TokenWithdrawer.sol   | 100.00% (5/5)       | 100.00% (6/6)       | 50.00% (1/2)      | 100.00% (4/4)     |
|---------------------------------+---------------------+---------------------+-------------------+-------------------|
| Total                           | 66.40% (334/503)    | 65.32% (373/571)    | 35.14% (78/222)   | 67.89% (129/190)  |
+---------------------------------+---------------------+---------------------+-------------------+-------------------+
```

Concluding Remarks

While conducting the audits of the Nitro League smart contracts, it was observed that the contracts contain Medium and Low severity issues.

Our auditors suggest that Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Nitro League platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for code refactoring by the team on critical issues.

ImmuneBytes