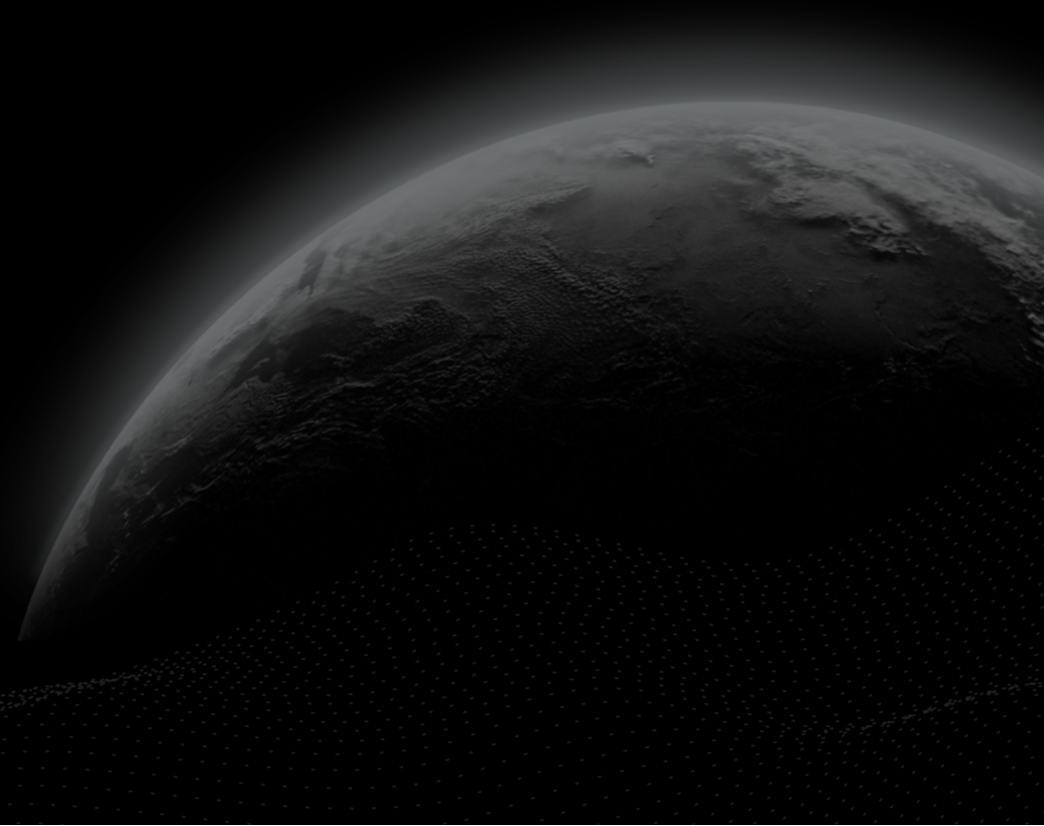
CERTIK

Security Assessment

Space Nation

CertiK Assessed on Mar 23rd, 2024



CertiK Assessed on Mar 23rd, 2024

Space Nation

The security assessment was prepared by CertiK, the leader in Web3.0 security.

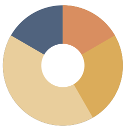
Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Binance Smart Chain (BSC) | Ethereum (ETH)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 03/23/2024	N/A

CODEBASE	COMMITS
updated	9635ce7556a1976dc41f0fc83e0c08c3d409f16f
View All in Codebase Page	View All in Codebase Page

Vulnerability Summary



12	11	0	0	1	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	1 Resolved, 1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
3	Medium	3 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
5	Minor	5 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
2	Informational	2 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | SPACE NATION

CODEBASE | SPACE NATION

Repository

updated

Commit

9635ce7556a1976dc41f0fc83e0c08c3d409f16f

AUDIT SCOPE | SPACE NATION

1 file audited ● 1 file with Acknowledged findings

ID	File	SHA256 Checksum
● ANF	AuctionNFT.sol	cd88967ace3c589a339791f79726b2199bcd841e82b1fe44b9f1b156b088d7ba

APPROACH & METHODS | SPACE NATION

This report has been prepared for Space Nation to discover issues and vulnerabilities in the source code of the Space Nation project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

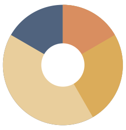
The auditing process pays special attention to the following considerations:

- 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 produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical 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:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | SPACE NATION



12	0	2	3	5	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Space Nation . Through this audit, we have uncovered 12 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
ANF-07	Ether Locked Via Reentrancy	Concurrency	Major	Resolved
ANF-08	**Centralization Risks In AuctionNFT.Sol**	**Centralization**	**Major**	**Acknowledged**
ANF-01	Potential Out-Of-Gas Exception	Logical Issue	Medium	Resolved
ANF-09	Already Refunded Ticket Can Win	Volatile Code	Medium	Resolved
ANS-01	Wrong Data Returned By `getRaffledId()`	Incorrect Calculation	Medium	Resolved
ANF-10	Deprecated Usage Of `Counters.sol`	Logical Issue	Minor	Resolved
ANF-11	Missing Zero Address Validation	Volatile Code	Minor	Resolved
ANF-12	Potential Reentrancy Attack (Out-Of-Order Events)	Concurrency	Minor	Resolved
ANF-13	Unnecessary `receive()` Function	Inconsistency	Minor	Resolved
ANF-14	Low-Level Call Usage	Coding Style	Minor	Resolved
ANF-15	Inaccurate Comments	Coding Issue	Informational	Resolved

ID	Title	Category	Severity	Status
ANF-16	Visibility Naming Convention	Coding Style	Informational	Resolved

ANF-07 | ETHER LOCKED VIA REENTRANCY

Category	Severity	Location	Status
Concurrency	Major	AuctionNFT.sol (c927ed2): 399, 402~405, 413~415, 427	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

The attacker can skip the revenue sending and NFT minting.

Scenario

Consider the scenario:

1. `stakes.length` is 100, the first stake is owned by the attacker and won some NFT
2. The attacker calls `airdrop(10)`
3. During the NFT minting the attacker gains the control and calls `airdrop(10)` again
4. Since `airdropIndex` was not yet updated, the `airdrop()` starts processing the first 10 stakes again
5. Already processed stakes are skipped since `info.hasClaimedNFT` is true
6. The second call to `airdrop()` increments `airdropIndex` and exits
7. The first call to `airdrop()` increments `airdropIndex` **again** and exits

As a result, stakes from 10 to 20 were not processed since `airdropIndex` is 20 already. The corresponding ether will be locked in the contract.

Recommendation

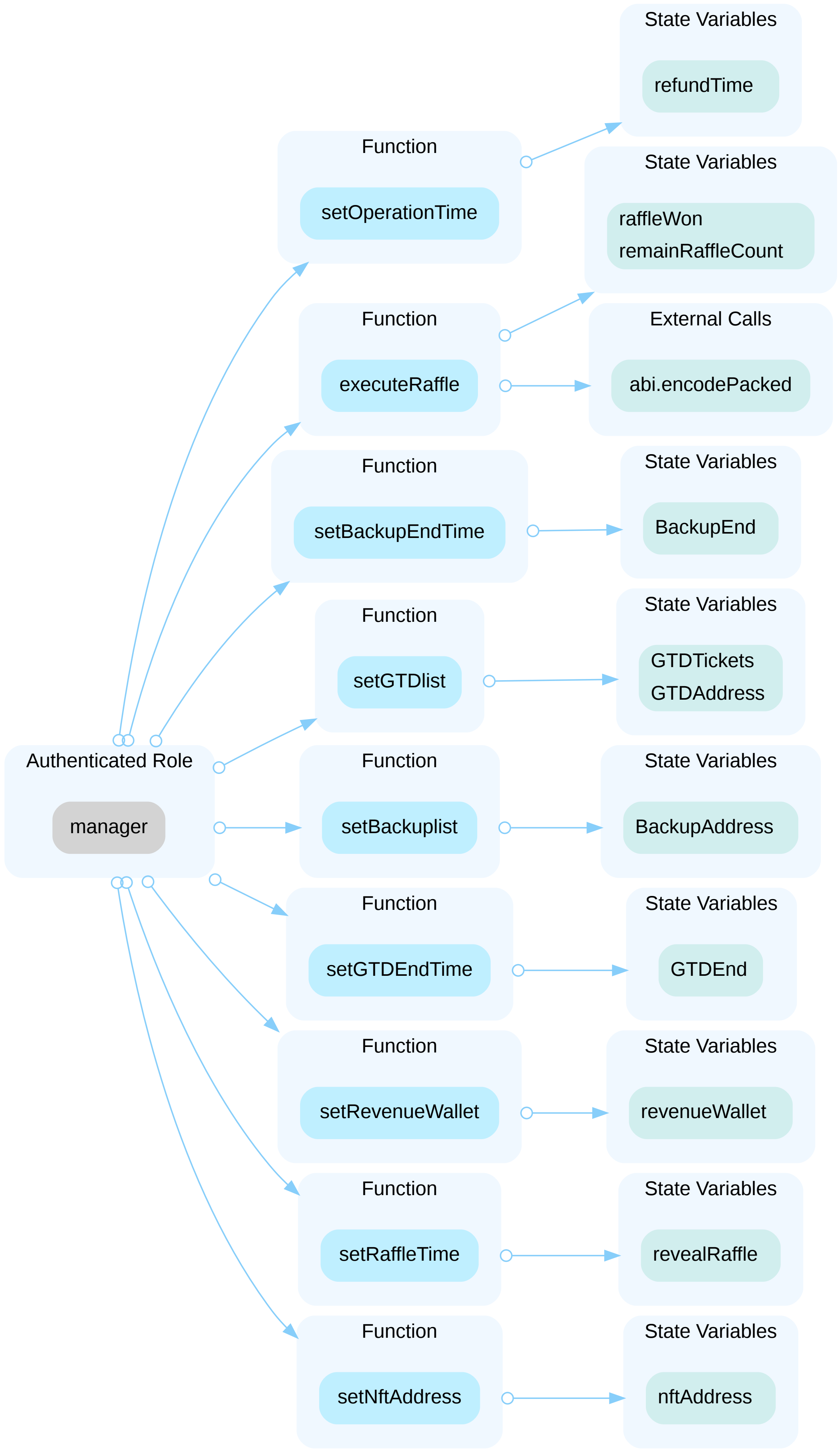
We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

ANF-08 | CENTRALIZATION RISKS IN AUCTIONNFT.SOL

Category	Severity	Location	Status
Centralization	**Major**	**AuctionNFT.sol (base): 137, 143, 149, 156, 163, 170, 177, 193, 243, 290, 324**	**Acknowledged**

Description

In the contract `StakeNFT` the role `manager` has authority over the functions shown in the diagram below. Any compromise to the `manager` account may allow the hacker to take advantage of this authority and `executeRaffle()` with any desired `seed` to influence the winners list. They can also `setGTDlist()` and `setBackuplist()` allowing to buy tickets with a lower price and always get NFT.


State Variables
refundTime
Function
setOperationTime
State Variables
raffleWon
remainRaffleCount
Function
executeRaffle
External Calls
abi.encodePacked
Function
setBackupEndTime
State Variables
BackupEnd
Function
setGTDlist
State Variables
GTDTickets
GTDAddress
Authenticated Role
manager
Function
setBackuplist
State Variables
BackupAddress
Function
setGTDEndTime
State Variables
GTDEnd
Function
setRevenueWallet
State Variables
revenueWallet
Function
setRaffleTime
State Variables
revealRaffle
Function
setNftAddress
State Variables
nftAddress

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[**Project Team**]: We do carefully manage the privileged account's private key in a Hardware wallet to avoid any potential risks of being hacked and we place limitations on the `onlyManager` modifier functions as recommended.

ANF-01 | POTENTIAL OUT-OF-GAS EXCEPTION

Category	Severity	Location	Status
Logical Issue	Medium	AuctionNFT.sol (c927ed2): 184, 198, 215, 330, 341, 376, 395, 457, 493	Resolved

Description

When a loop allows an arbitrary number of iterations or accesses state variables in its body, the function may run out of gas and revert the transaction.

Function `executeRaffle()` contains a loop and its loop condition depends on state variables: `_publicStakesId` .

Function `_claimInfo()` contains a loop and its loop condition depends on state variables: `userStakes` .

Function `getUserStakes` contains a loop and its loop condition depends on state variables: `userStakes` .

Function `getRaffledId` contains a loop and its loop condition depends on state variables: `_publicStakesId` .

Recommendation

It is recommended to place limitations on the loop's bounds like in `airdrop()` function.

ANF-09 | ALREADY REFUNDED TICKET CAN WIN

Category	Severity	Location	Status
Volatile Code	Medium	AuctionNFT.sol (base): 328	Resolved

Description

`executeRaffle()` allows it to be executed any time starting from `BackupEnd` . `refund()` allows it to be executed after `refundTime` . As a result, the same ticket can be refunded, become winning, and then airdropped. That breaks the contract's invariant: the sum of all ether to be refunded and to be sent to `revenueWallet` equals the contract balance.

Scenario

Consider the scenario:

1. The user stakes via `allStake()`
2. `BackupEnd` time comes
3. The `manager` calls `executeRaffle()` , some tickets marked as `raffleWon`
4. `refundTime` comes
5. The user calls `refund()` and gets `refundAmount` for non winning ticket
6. The `manager` calls `executeRaffle()` again, the refunded ticket marked as `raffleWon`
7. The user calls `airdrop()` and gets NFT minted as a winning ticket
8. The ticket price is sent to `revenueWallet`

Recommendation

We recommend preventing of the same ticket to be refunded and minted.

ANS-01 | WRONG DATA RETURNED BY `getRaffledId()`

Category	Severity	Location	Status
Incorrect Calculation	Medium	AuctionNFT.sol (9a2a5aa): 531	Resolved

Description

```
526         uint256 ncount = start + count >= length ? length : start + count;
527         uint256 counts = ncount - start;
528         uint256[] memory raffleId = new uint256[](counts);
529         uint256 index;
530
531         for (uint256 j = start; j < counts; j++) {
```

`counts` is the size of the resulting array. The `for` loop finish at `ncount`.

Recommendation

We recommend updating the `for` loop.

ANF-10 | DEPRECATED USAGE OF `Counters.sol`

Category	Severity	Location	Status
Logical Issue	Minor	AuctionNFT.sol (base): 41	Resolved

Description

The linked contracts import and use OpenZeppelin's `Counters` contract. OpenZeppelin has deprecated the usage of the `Counters` contract.

Recommendation

We recommend removing the usage of deprecated 3rd party contracts.

ANF-11 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	AuctionNFT.sol (c927ed2): 109, 110, 111	Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

- `_manager` is not zero-checked before being used.
- `_nftAddress` is not zero-checked before being used.
- `_revenueWallet` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

ANF-12 | POTENTIAL REENTRANCY ATTACK (OUT-OF-ORDER EVENTS)

Category	Severity	Location	Status
Concurrency	Minor	AuctionNFT.sol (c927ed2): 402~405, 413~415, 424, 440, 442	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is considered minor because the reentrancy only causes out-of-order events.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

ANF-13 | UNNECESSARY `receive()` FUNCTION

Category	Severity	Location	Status
Inconsistency	Minor	AuctionNFT.sol (base): 134	Resolved

Description

The `receive()` function allows the contract to accept the undesired or unexpected ether. There is no reason to allow the users to send ether and there is no way to extract it.

Recommendation

We recommend removing the function `receive()`.

ANF-14 | LOW-LEVEL CALL USAGE

Category	Severity	Location	Status
Coding Style	Minor	AuctionNFT.sol (base): 410	Resolved

Description

```
409       bytes4 SELECTOR = bytes4(
410           keccak256(bytes("nftcallermint(address,uint256)"))
411       );
412
413       (bool nftcallsuccess, bytes memory data) = nftAddress.call(
414           abi.encodeWithSelector(SELECTOR, staker, info.nftCount)
415       );
416
417       require(
418           nftcallsuccess &&
419               (data.length == 0 || abi.decode(data, (bool))),
420           "Mint_NFT_Faliled"
421       );
```

The smart contract contains low-level `call()`. Since such calls bypass some of the automatic checks that Solidity provides, like function type checks, they can introduce vulnerabilities, logic errors, or unexpected behavior.

An interface can be declared:

```
interface INFTMinter {
  function nftcallermint(address recipient, uint256 count) external returns (bool);
}
```

Then executed:

```
  require(INFTMinter(nftAddress).nftcallermint(staker, info.nftCount),
"nftcallermint failed");
```

Recommendation

We recommend using of interfaces instead of low-level calls.

ANF-15 | INACCURATE COMMENTS

Category	Severity	Location	Status
Coding Issue	Informational	AuctionNFT.sol (base): 169, 176, 181, 192, 298, 322	Resolved

Description

Some comments are inaccurate or outdated.

- "update revealRaffle" is supposed to be "update refundTime"
- "deplicated" is supposed to be "duplicated"
- "MisMatchged" is supposed to be "Mismatched"
- "alrerady" is supposed to be "already"

Recommendation

We recommend updating the comments.

ANF-16 | VISIBILITY NAMING CONVENTION

Category	Severity	Location	Status
Coding Style	Informational	AuctionNFT.sol (base): 361	Resolved

Description

The name of a function or state field with `public` or `external` visibility should not start with an underscore, "`_`".

The name of a function or state field with `private` or `internal` visibility should start with an underscore, "`_`".

`_claimInfo()` is a `public` function but starts with an underscore.

Recommendation

We recommend renaming the items to follow Solidity naming convention.

OPTIMIZATIONS | SPACE NATION

ID	Title	Category	Severity	Status
ANF-02	Unnecessary Use Of SafeMath	Coding Issue	Optimization	Resolved
ANF-03	Costly Operation Inside Loop	Coding Issue	Optimization	Resolved
ANF-04	Unnecessary Storage Read Access In `for` Loop	Coding Issue	Optimization	Resolved
ANF-05	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
ANF-06	Redundant Code	Code Optimization	Optimization	Resolved

ANF-02 | UNNECESSARY USE OF SAFEMATH

Category	Severity	Location	Status
Coding Issue	● Optimization	AuctionNFT.sol (c927ed2): 210~213, 214, 251~254, 260	● Resolved

Description

The `SafeMath` library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations will automatically revert in case of integer overflow or underflow.

Recommendation

We recommend removing the usage of `SafeMath` library and using the built-in arithmetic operations provided by the Solidity programming language.

ANF-03 | COSTLY OPERATION INSIDE LOOP

Category	Severity	Location	Status
Coding Issue	Optimization	AuctionNFT.sol (c927ed2): 280	Resolved

Description

Reading, initializing, and modifying storage variables cost more gas than operating local variables, and this gas cost can significantly increase when these operations are performed inside a loop. `avaWLCount` is updated `tickets` times.

Reference: https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html#storage-memory-and-the-stack

Recommendation

We recommend using `avaWLCount -= tickets` outside of the loop instead.

ANF-04 | UNNECESSARY STORAGE READ ACCESS IN `for` LOOP

Category	Severity	Location	Status
Coding Issue	Optimization	AuctionNFT.sol (c927ed2): 330, 493	Resolved

Description

The `for` loop contains repeated storage read access in the condition check. Given that the ending condition does not change in the `for` loop, the repeated storage read is unnecessary, and its associated high gas cost can be eliminated.

Loop condition `i < _publicStakesId.length` accesses the `length` field of a storage array. Storage access costs substantially more gas than memory and stack access.

Recommendation

We recommend caching the variable used in the condition check of the `for` loop to avoid unnecessary storage access.

ANF-05 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	AuctionNFT.sol (base): 44, 47, 48, 49, 54, 55, 56, 57, 58, 59	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable.

ANF-06 | REDUNDANT CODE

Category	Severity	Location	Status
Code Optimization	Optimization	AuctionNFT.sol (base): 64	Resolved

Description

- `ONEDAY` can be replaced with `1 days`
- `ExtendAllstakeEnd` is never used

Recommendation

We recommend removing of redundant code.

APPENDIX | SPACE NATION

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 CertiK to perform a security assessment. This report 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK'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.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK 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.

Copyright © CertiK