AUDITS

MH SWIFTSCAN REVIEW

AGE OF TANKS

JUNE 27TH 2022

AUDITS

TABLE OF CONTENTS

LEGAL DISCLAIMER

MH Audits are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts MH Audits to perform a security review.

MH Audits does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

MH Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

The report is provided only for the contract(s) mentioned in the report and does not include any other potential additions and/or contracts deployed by Owner. The report does not provide a review for contract(s), applications and/or operations, that are out of this report scope.

MH Audits' goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

MH Audits represents an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. MH Audits' position is that each company and individual are responsible for their own due diligence and continuous security.

The security audit is not meant to replace functional testing done before a software release. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits and a public bug bounty program to ensure the security of the smart contracts.

INTRODUCTION

MH Audits is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Secure your project with MH Audits

We offer field-proven audits with in-depth reporting and a range of suggestions to improve and avoid contract vulnerabilities.

Industry-leading comprehensive and transparent smart contract auditing on all public and private blockchains.

Vunerability checking

A crucial manual inspection carried out to eliminate any code flaws and security loopholes. This is vital to avoid vulnerabilities and exposures incurring costly errors at a later stage.

Contract verification

A thorough and comprehensive review in order to verify the safety of a smart contract and ensure it is ready for launch and built to protect the end-user.

Risk assessment

Analyse the architecture of the blockchain system to evaluate, assess and eliminate probable security breaches. This includes a full assessment of risk and a list of expert suggestions.

In-depth reporting

A truly custom exhaustive report that is transparent and depicts details of any identified threats and vulnerabilities and classifies those by severity.

Fast turnaround

We know that your time is valuable and therefore provide you with the fastest turnaround times in the industry to ensure that both your project and community are at ease.

Best-of-class blockchain engineers

Our engineers combine both experience and knowledge stemming from a large pool of developers at our disposal. We work with some of the brightest minds that have audited countless smart contracts over the last 4 years.

PROJECT SUMMARY

PROJECT INTRODUCTION

Age of Tanks is a 3D turn-based strategy card game, set in an immersive metaverse, where players can assemble their preferred tanks, build their tank teams, outwit opponents, and rule the battlefield.

Built on the BSC Chain to underpin the ecosystem, the basic aim of presenting AOT tokens is to bring a fair and transparent payment system between gamers to make the ecosystem more reliable and safer. Gamers can earn AOT tokens by winning PVP battle conquer, achieve seasonal top-ranking, owning of AOT Refiner, selling of Tank NFT or parts at the in-game marketplace.

Project Name *Age Of Tanks*

Contract Name *AOT Token*

Contract Address *0x9589014f7a8547b89a6331eeee32b7fbd5852af9*

Contract Chain *Mainnet*

Contract Type *Smart Contract*

Platform *EVM*

Language *Solidity*

Codebase *https://bscscan.com/address/0x9589014f7a8547b89a6331eeee32b7fbd5852af9#code*

INFO & SOCIALS

Network *BNB Chain (BEP20)*

Max Token Supply *300.000.000*

Website *https://ageoftanks.io/*

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

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

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

Discord *https://discord.gg/ageoftanks*

Facebook *https://www.facebook.com/AgeofTanksOfficial*

Youtube *https://www.youtube.com/c/ageoftanksofficial*

TikTok *https://www.tiktok.com/@ageoftanks*

Instagram *https://www.instagram.com/ageoftanksofficial/*

Medium *https://ageoftanks.medium.com/*

GitHub *https://github.com/DEFINATION-PTE-LTD/ageoftanks*

PolygonScan *https://bscscan.com/token/0x9589014f7a8547b89a6331eeee32b7fbd5852af9*

AUDIT SCORES

Issues	11
Critical	0
Major	1
Medium	3
Minor	4
Informational	3
Discussion	0

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

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

AUDIT SCOPE

FILE	LOCATION
contract.sol	*BNB Chain Deployment: /address/0x9589014f7a8547b89a6331eeee32b7fbd5852af9#code*

REVIEW METHODOLOGY

TECHNIQUES

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

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective in the comments below.

TIMESTAMP

Version	v1.0
Date	2022/06/27
Description	Layout project
	Automated / Static security testing
	Summary

KEY FINDINGS

TITLE	SEVERITY	STATUS
Locked Ether Inside A Contract	Major	*Pending*
Unprotected Ether Withdrawal	Medium	*Pending*
Integer Overflow/Underflow	Medium	*Pending*
Incorrect BEP20 Interface	Medium	*Pending*
Outdated Complier Version	Minor	*Pending*
Cheaper Inequalities In Require()	Minor	*Pending*
Use Of Floating Pragma	Minor	*Pending*
Require Vs Assert	Minor	*Pending*
Block Values As A Proxy For Time	Informational	*Pending*
Presence Of Overpowered Role	Informational	*Pending*
Hard-Coded Address Detected	Informational	*Pending*

Description:

The contract is programmed to receive Ether, but no method was found that allowed the Ether to be withdrawn, i.e., call, transfer, transferFrom, send, ***or*** call.value ***at least once.***

Without a withdrawal function, the Ethers will forever be locked inside the contract if the contract's code is not upgradeable leading to loss of funds.

Location: *contract.sol L36-L136*

Issue: *Locked Ether Inside A Contract*

Level: *Major*

Recommendation: *Implement a withdraw function or reject payments (contracts without a fallback function do it automatically).*

Alleviation:

Description:

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

Location: *contract.sol L129-L135*

Issue: *Unprotected Ether Withdrawal*

Level: *Medium*

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

Alleviation:

Description:

An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum storage of a variable type. Integers overflow or underflow may prove fatal when during an arithmetic operation, the number goes over or under the designated limit. This may prove fatal during calculations related to ether or tokens.

Location: *contract.sol L11*

Issue: *Integer Overflow/Underflow*

Level: *Medium*

Recommendation: *Solidity compiler versions* >=0.8.0 *automatically handle overflow and underflow validations. If you're using a lower solidity version, it is recommended to use the SafeMath library to protect the arithmetic operations.*

Alleviation:

Description:

The BEP20 Token standard has a specific format for all the functions. In Solidity, a function selector is derived from its function name and the type of the input parameters. So if the return value is omitted from function description, function selector will still be BEP20 compliant but the actual function won't be. The contract was found to be using BEP20 interfaces. The implementations were improperly done and do not follow the standard BEP20 implementation. Either the return values or the parameter type was missing or incorrect. This will cause errors when the function that called these interfaces expects some return value but instead, nothing or the default values will be returned.

Location: *contract.sol L33*

Issue: *Incorrect BEP20 Interface*

Level: *Medium*

Recommendation: *Make sure that the functions used by the contract is BEP20 compliant and follows the same format with respect to the arguments passed and the return values expected so that it remains compliant with the token standard.*

Alleviation:

Description:

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

Location: *contract.sol L06*

Issue: *Outdated Complier Version*

Level: *Minor*

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

Alleviation:

Description:

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

Location: *contract.sol L69-L71; L86-L89; L98*

Issue: *Cheaper Inequalities In* Require()

Level: *Minor*

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

Alleviation:

Description:

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.

The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.

Location: *contract.sol L06*

Issue: *Use Of Floating Pragma*

Level: *Minor*

Recommendation: *It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.*

pragma solidity ^0.4.17; not recommended -> compiles with 0.4.17 and above

pragma solidity 0.8.4; recommended -> compiles with 0.8.4 only

Alleviation:

Description:

Assert *should only be used to test for internal errors, and to check invariants. Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract that you should fix. It should only be used in case of invariant checking* Require *should only be used for validating return values and user input.*

Location: *contract.sol L11; L18; L23*

Issue: *Require Vs Assert*

Level: *Minor*

Recommendation: *If* assert *is not used to check an invariant consider replacing it with require. Use* require *in case of return and user input values.*

Alleviation:

Description:

Contracts often need access to time values to perform certain types of functionality. Values such as block.timestamp *and* block.number *can be used to determine the current time or the time delta. However, they are not recommended for most use cases.*

For block.number*, as Ethereum block times are generally around 14 seconds, the delta between blocks can be predicted. The block times, on the other hand, do not remain constant and are subject to change for a number of reasons, e.g., fork reorganizations and the difficulty bomb.*

Due to variable block times, block.number *should not be relied on for precise calculations of time.*

Location: *contract.sol L116: L121*

Issue: *Block Values As A Proxy For Time*

Level: *Informational*

Recommendation: *Smart contracts should be written with the idea that block values are not precise, and their use can have unexpected results. Alternatively, oracles can be used.*

Alleviation:

Description:

The overpowered owner (i.e., the person who has too much power) is a project design where the contract is tightly coupled to their owner (or owners); only they can manually invoke critical functions.

Due to the fact that this function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

Location: *contract.sol L108; L114*

Issue: *Presence Of Overpowered Role*

Level: *Informational*

Recommendation: *We recommend designing contracts in a trust-less manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use a MultiSig wallet for this address.*

*For systems that are provisioned for a single user, you can use [Ownable.sol](*https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/ownership/Ownable.sol***).***

For systems that require provisioning users in a group, you can use [@openzeppelin/Roles.sol](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/access/Roles.sol***) or [@hq20/Whitelist.sol](***https://github.com/HQ20/contracts/blob/v0.0.2/contracts/access/Whitelist.sol***).***

Alleviation:

Description:

The contract contains an unknown address. This address might be used for some malicious activity. Please check the hard-coded address and its usage.

These hard-coded addresses may be used everywhere throughout the code to define states and interact with the functions and external calls.

Location: *contract.sol L64*

Issue: *Hard-Coded Address Detected*

Level: *Informational*

Recommendation: *It is required to check the address. Also, it is required to check the code of the called contract for vulnerabilities.*

Ensure that the contract validates if there's an address or a code change or test cases to validate if the address is correct.

Alleviation:

https://bscscan.com/address/0x9589014f7a8547b89a6331eeee32b7fbd5852af9#contract

FINDING CATEGORIES

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

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

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

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The scanner modules find and flag issues related to gas optimizations that help in reducing the overall gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

AUDIT SCORES

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

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

MH Audits AuditScores are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports and scores are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts MH Audits to perform a security review.

WEBSITE
MHAUDITS.IO

TWITTER
@MHAUDITS