CHAIN AUDIT
PROFESSIONAL AUDIT

SMART CONTRACT AUDIT

SPACE BASE

SEPTEMBER 4, 2023.

Prepared by

Francis K.

Approved by

Chain Audit Team

INTRODUCTION

AUDITING FIRM	CHAIN AUDIT
CLIENT FIRM	SPACE BASE
METHODOLOGY	MANUAL CODE REVIEW AND AUTOMATED ANALYSIS
LENGUAGE	SOLIDITY
CONTRACT	0xA2493539e1035f75bf040288802d3BF2FcFD2Bf6
BLOCKCHAIN	ETHEREUM
CENTRALIZATION	YES
WEBSITE	https://spacebase.site/

You can check the authenticity of this audit on our website.

EXECUTIVE SUMMARY

Objective and Scope

The purpose of this audit was to evaluate and ensure the integrity, security, and functionality of the "SpaceBase" smart contract developed in Solidity for the Ethereum platform. A thorough review of the code was conducted, encompassing both manual and automated testing, with the aim of identifying and mitigating potential vulnerabilities and risks associated with the contract.

Methodology

A combined review approach was adopted:

- **Manual Review:** Our team of Solidity experts carried out a detailed and systematic review of the source code, paying particular attention to common vulnerabilities and insecure code patterns. Each function and modularity of the contract, its internal logic, and interactions amongst them were assessed.

- **Automated Review:** We utilized top-tier automated audit tools to scan the code for known vulnerabilities, compilation errors, and other potential security issues.

STATUS	CRITICAL 🔴	MAJOR 🟠	MEDIUM 🟡	MINOR 🟢	UNKNOWN 🟤
OPEN	0	1	1	1	0
ACKNOWLEDGED	0	0	0	0	0
RESOLVED	0	0	0	0	0

IMPORTANCE OF SMART CONTRACT AUDITS

Smart contracts have ushered in a new era of trustless and decentralized operations on the blockchain. While they have the potential to revolutionize numerous sectors, from finance to supply chain, their immutable nature means that any vulnerability or flaw in their code is permanent once deployed. This underscores the crucial importance of smart contract audits.

Why are Audits Necessary?

1. **Immutability:** Once a contract is deployed, it cannot be changed. A bug or vulnerability can be exploited repeatedly unless it's fixed in a new version of the contract.
2. **Financial Implications:** Smart contracts often handle and manage valuable assets. Vulnerabilities can lead to substantial financial losses.
3. **Reputation:** A flawed contract can tarnish the reputation of a project, leading to a loss of trust and confidence among its users.
4. **Complexity:** Solidity, the primary language for Ethereum contracts, has its quirks. Even experienced developers might overlook subtleties that can become vulnerabilities.

Types of Smart Contract Attacks:

Smart contracts are vulnerable to a variety of attacks. Some of the most common include reentrancy attacks, overflow and underflow attacks, timestamp dependence attacks, and more. Each of these attacks exploits specific vulnerabilities in contract code, and a comprehensive audit aims to safeguard against all known vulnerabilities.

RISK CATEGORIES

Risk Type	**Definition**
Critical 🔴	A vulnerability that, if exploited, could have a catastrophic impact, potentially leading to substantial financial loss or irreversible damage to the contract's operations.
Major 🟠	A significant vulnerability that might not lead to total loss but can hamper the contract's functionality and compromise its objectives.
Medium 🟡	Issues that are of concern but might require specific conditions to be exploited. They can pose risks if combined with other vulnerabilities.
Minor 🟢	These vulnerabilities pose a limited threat and have a lower probability of being exploited. Often, they relate to best practices rather than direct exploitable flaws.
Unknown 🟤	Risks that haven't been fully understood or classified yet. They could be new or unique to the contract's specific design or context.

Status of Identified Risks

Status Type	Definition
Open	Vulnerabilities that have been identified but have not yet been addressed or rectified by the development team.
Acknowledged	The development team has recognized the issue but might be in the process of determining the best solution or mitigation strategy.
Resolved	The vulnerability has been effectively addressed and resolved by the development team, eliminating the risk it posed.

AUDITING IS AN ESSENTIAL STEP IN THE DEVELOPMENT AND DEPLOYMENT OF SMART CONTRACTS. IT ENSURES NOT ONLY THE SECURITY AND RELIABILITY OF THE CONTRACT BUT ALSO BUILDS TRUST AMONG ITS USERS AND STAKEHOLDERS. AS SMART CONTRACTS CONTINUE TO GROW IN COMPLEXITY AND IMPORTANCE, ROBUST AUDITING MECHANISMS WILL REMAIN A CORNERSTONE OF THE BLOCKCHAIN ECOSYSTEM.

IMPORTANT CONTRACT DETAILS

- **CHAIN:** ETHEREUM
- **TOKEN:** ETH
- **DAPP TYPE:** TOWER AND MAGE-BASED INVESTMENT GAME.
- **CONVERSION RATE:**
  - 1 MANA = 0,0001 ETH
  - 1 EMERALDS = 0,012 MANA
- **REFERRAL PERCENTAGE** :
  - STANDARD REFERRAL: 5% IN MANA
  - AFFILIATE REFERRAL: 7% IN EMERALDS FOR AFFILIATE
- **FEE :** 4%
- **MANAGER:** UNIQUE ADDRESS DEFINED IN THE CONSTRUCTOR, HAS LIMITED ADMINISTRATIVE POWERS.
- **REWARD ACCUMULATION:** BASED ON "YIELD", WHICH IS CALCULATED FROM THE TYPE AND LEVEL OF MAGES.

ANALYSIS OF CONTRACT FUNCTIONS

Function Name	Description	Usage
updateAffiliateStatus(address user, bool status)	Allows the manager to update the affiliate status of a user.	Used by the contract owner to mark or unmark a user as an affiliate.
addMana(address ref)	Allows users to add 'mana' to the contract by sending ETH. Also handles referral bonuses.	Users send ETH to this function to increase their 'mana' and, if they have a referrer, to give referral bonuses.
compound(uint256 emeralds)	Allows users to convert their 'emeralds' into 'mana' at a given rate.	Used by users to increase their 'mana' using their 'emeralds'.
withdrawMoney(uint256 emeralds)	Allows users to withdraw their 'emeralds' as ETH.	Used by users to withdraw their earnings.
collectMoney()	Syncs and collects accumulated 'emeralds' for the calling user.	Used by users to update and collect their 'emeralds' without withdrawing them.
upgradeTower(uint256 towerId)	Allows users to upgrade their tower, which increases the yield of 'emeralds'.	Users invoke this function to spend 'mana' and upgrade their tower.
upgradeTreasury()	Allows users to upgrade their treasury, which might increase capacity or yield.	Users use this function to spend 'mana' and upgrade their treasury.
getMages(address addr)	Queries and returns the 'mages' status for a given address.	To retrieve information about the 'mages' of a specific user.
syncTower(address user)	Syncs the user's tower, updating values based on yield and time.	This is an internal function used by other functions to ensure a user's tower data is up-to-date before any other operation.
getUpgradePrice(uint256 towerId, uint256 mageId)	Returns the price to upgrade a tower to a specific 'mageId'.	Used internally to calculate upgrade costs.
getYield(uint256 towerId, uint256 mageId)	Returns the 'emeralds' yield for a specific 'mageId'.	Used internally to calculate the yield of a tower after an upgrade.
getTreasure(uint256 treasureId)	Returns the price and value for a specific 'treasureId'.	Used internally to calculate costs and values related to treasures.

OWNER PRIVILEGES

THE SPACEBASE CONTRACT DISPLAYS THE FOLLOWING FUNCTION THAT GRANTS EXCLUSIVE RIGHTS TO THE CONTRACT MANAGER:

- **UPDATEAFFILIATESTATUS(ADDRESS USER, BOOL STATUS):**
  - **DESCRIPTION:** ALLOWS THE MANAGER TO UPDATE THE AFFILIATE STATUS OF A USER.
  - **RESTRICTION:** ONLY THE CONTRACT MANAGER CAN CALL THIS FUNCTION. THIS ENSURES THAT ONLY THE MANAGER HAS THE AUTHORITY TO MARK OR UNMARK A USER AS AN AFFILIATE.

IT'S PARAMOUNT TO UNDERSTAND THAT THIS FUNCTIONALITY CAN SIGNIFICANTLY INFLUENCE THE DYNAMICS OF THE CONTRACT, ESPECIALLY CONCERNING THE AFFILIATE SYSTEM. THEREFORE, IT SHOULD BE INVOKED JUDICIOUSLY.

CONTRACT OVERVIEW CHECKLIST

VULNERABILITY/PROBLEM DESCRIPTION	STATUS
Reentrancy Attack	Warning
Integer Overflow/Underflow	Pass
Delegatecall Vulnerability	Pass
Front-Running	Pass
Visibility of functions	Pass
Fallback Function Vulnerability	Pass
State Variable Mutable	Pass
Erroneous External Calls	Pass
Immutable Keyword Misuse	Pass
Storage Layout & Proxy Contracts	Pass
Timestamp Dependence	Warning
Compiler error	Pass
Short Address Attack	Pass
Using inline assembly	Pass
Weak sources of randomness	Pass
Gas limit and loops	Pass
Use of tx.origin	Pass

CONTRACT OVERVIEW CHECKLIST

VULNERABILITY/PROBLEM DESCRIPTION	STATUS
Oracle security	Warning
Malicious libraries	Pass
Missing event emission	Pass
Uninitialised Storage Pointers	Pass
Under-optimized Code	Pass
Inadequate Testing	Pass
Magic Numbers	Pass
Inadequate Permissions & Governance	Warning
Presence of unused code	Pass
Self-destruct interaction	Pass
User balance manipulation	Pass
Access Control and Authorization	Warning
Ownership Control	Pass
Assets Manipulation	Pass
Liquidity Access	Pass
Stop and Pause Trading	Pass
Missing zero address validation	Pass

CONTRACT OVERVIEW CHECKLIST

VULNERABILITY/PROBLEM DESCRIPTION	STATUS
Race Conditions	Pass
Sybil Attack	Pass
Data consistency	Pass
Divide before multiply	Pass
Unnecessary use of SafeMath	Pass
Solidity Naming Guides	Pass
Signature unique id	Pass
Optimize code & gas fee	Pass
Phishing with contract addresses	Pass
Array Length Manipulation	Pass
Unchecked Return Values	Pass
Forced Ether Reception	Pass
Transfer Block	Pass
Floating pragma	Pass
Deprecated solidity functions	Pass
Lack of Arbitrary limits	Pass
Incorrect Inheritance Order	Pass

CONTRACT OVERVIEW CHECKLIST

VULNERABILITY/PROBLEM DESCRIPTION	STATUS
Typographical Errors	Pass
Requirement Violation	Pass
Coding Style Violations	Pass
Third-Party Dependencies	Pass
Dos with revert Passed	Pass

CRITICAL VULNERABILITY FINDINGS

Vulnerability	Risk Type	Vulnerable	Reason	Affected Code
-	-	-	-	-

FINDINGS AND RECOMMENDATIONS

1. REENTRANCY ATTACK:

- **RATING:** MEDIUM 🟡
- **OBSERVATION:** THE CONTRACT CONTAINS FUNCTIONS THAT MAKE EXTERNAL CALLS FOLLOWING STATE UPDATES, WHICH COULD MAKE CERTAIN FUNCTIONS SUSCEPTIBLE TO REENTRANCY ATTACKS.
- **RECOMMENDATION:** UTILIZE PATTERNS LIKE THE CHECK-EFFECTS-INTERACTION PATTERN TO ENSURE EXTERNAL CALLS ARE MADE AT THE END OF FUNCTIONS. YOU MAY ALSO CONSIDER USING MODIFIERS LIKE `NONREENTRANT` TO GUARD AGAINST SUCH ATTACKS.

2. TIMESTAMP DEPENDENCE:

- **RATING:** MINOR 🟢
- **OBSERVATION:** THE CONTRACT USES `BLOCK.TIMESTAMP` TO CALCULATE ELAPSED TIME, WHICH CAN BE MANIPULATED BY MINERS TO SOME EXTENT.
- **RECOMMENDATION:** CONSIDER USING ALTERNATIVE SOLUTIONS FOR TIME HANDLING OR CONSIDER TOLERANCES TO MINIMIZE THE POTENTIAL IMPACT OF MANIPULATION.

3. ACCESS CONTROL AND AUTHORIZATION:

- **RATING:** MAJOR 🟠
- **OBSERVATION:** SOME FUNCTIONS WERE FOUND TO LACK PROPER ACCESS CONTROL RESTRICTIONS.
- RECOMMENDATION: IMPLEMENT A ROBUST ACCESS CONTROL SYSTEM AND ENSURE ONLY AUTHORIZED ADDRESSES CAN ACCESS CRITICAL FUNCTIONS.

CONCLUSION

AFTER A THOROUGH EXAMINATION OF THE SPACEBASE SMART CONTRACT, WE IDENTIFIED A SERIES OF POTENTIAL SECURITY WARNINGS. WHILE NO CRITICAL VULNERABILITIES WERE DISCOVERED THAT COULD LEAD TO SUBSTANTIAL FINANCIAL LOSS OR IRREVERSIBLE DAMAGE TO THE CONTRACT'S OPERATIONS, THE PRESENCE OF CERTAIN ISSUES, PARTICULARLY AROUND ACCESS CONTROL, WARRANTS ATTENTION. MOREOVER, THE SUSCEPTIBILITY TO A REENTRANCY ATTACK, EVEN IF IT'S OF MEDIUM SEVERITY, REQUIRES PROACTIVE MEASURES TO FORTIFY THE CONTRACT'S RESILIENCE AGAINST POTENTIAL THREATS.

SECURITY RATING 7/10

THIS RATING INDICATES THAT WHILE THE CONTRACT HAS IMPLEMENTED SEVERAL SECURITY MEASURES AND ADHERES TO MANY BEST PRACTICES, THERE REMAINS ROOM FOR IMPROVEMENT. ADDRESSING THE IDENTIFIED WARNINGS WILL BOLSTER THE CONTRACT'S SECURITY PROFILE AND INSTILL GREATER CONFIDENCE AMONG ITS USERS.

CHAINAUDIT
FRANCIS K.
AUDITOR FOR CHAINAUDIT