SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SpacePi
Date: May 31, 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for SpacePi
Approved By	Noah Jelich | Lead SC Auditor at Hacken OU
Type	ERC20 token; Staking, ERC721 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://space-pi.com/
Changelog	13.04.2023 - Initial Review 05.05.2023 - Second Review 31.05.2023 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by SpacePi (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/SpacePiCom/AduitTest
Commit	ca0d4cc811051d6c004f453f4311e4465410d6cf
Whitepaper	-
Functional Requirements	Link
Technical Requirements	-
Contracts	File: contracts/ERC721Distributor.sol SHA3: 3fb95fa0b7f244952e4f3b4252b2079cca4b5c88c3456f0a3d18fe694af47aa2 File: contracts/StakeSpacePi.sol SHA3: a90171e2ba9c19d0ee2d335e60102daab42099ce4201d980777e8d2a4e2c5814 File: contracts/interfaces/IRelationship.sol SHA3: b5e263999ecc6a09f25659ba0518962f6a4c239649d9330184fae5334a232eb4 File: contracts/utils/Relationship.sol SHA3: daa4a8f8adae73852ed35ff79f6f60d9fc591082880298dc8df2ab268fd03c53

Second review scope

Repository	https://github.com/SpacePiCom/AduitTest
Commit	3ff2cc9417b9007f4da4183324a78734e7bc2c61
Whitepaper	-
Functional Requirements	Link
Technical Requirements	-
Contracts	File: contracts/StakeSpacePi.sol SHA3: 71d77a9eab05776e4da35185308fb5b231bec2c32ce4abaf9163fdec5c5d1bc8

	File: contracts/interfaces/IRelationship.sol SHA3: 27c82f691997c4d4c7a73597c9918fed1b8924142de0ea082e421d1e4ff540bc File: contracts/utils/Relationship.sol SHA3: bff028ffeeef64a1218ea6bacfa936b08aa8dc4f11a52bb9658c2aacf53e81aa

Third review scope

Repository	https://github.com/SpacePiCom/AduitTest
Commit	ab617c81d3d8fac8c86f6516109bbc0c1541f1bd
Whitepaper	-
Functional Requirements	Link
Technical Requirements	-
Contracts	File: contracts/StakeSpacePi.sol SHA3: bf08fdee8569883649d57687259e6197c3b35d3070fe5d70cbbdca40647e0d6d File: contracts/interfaces/IRelationship.sol SHA3: 43ccadd5a164384609f8d3ce43991dd686703333fc70eb803384a9cd820d1465 File: contracts/utils/Relationship.sol SHA3: adba9f38182532e823502820d3812fceaa56141ac18198207c21af84350f936d

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **4** out of **10**.

- Functional requirements are partially covered.
- Technical description is not provided.
- NatSpec format is partially followed. Code explanations were mostly missing.

Code quality

The total Code Quality score is **7** out of **10**.

- Unused code was present.
- The development environment is configured.

Test coverage

Code coverage of the project is **56.41%** (branch coverage).

- Missing multi-user interactions, multi depositing & claiming
- Missing negative tests.

Security score

As a result of the audit, the code contains **2** medium severity issues. The security score is **8** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **6**.

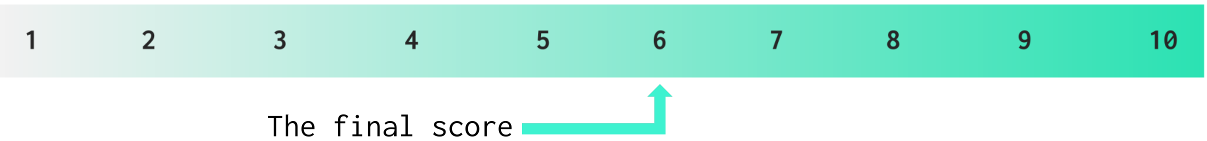

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
13 April 2023	8	7	5	4
09 May 2023	1	4	1	1

21 June 2023	0	2	0	0

System Overview

SpacePi is a system that contains a staking platform with the following contracts:

- *StakeSpacePi* — The StakeSpacePi contract is an ERC20 staking contract that allows users to deposit their tokens into different pools that offer different APYs and lock-up periods.
- *Relationship* — The Relationship contract is a contract that manages the invitation relationship between users of the platform. Each user is assigned a unique invitation code that they can share with others to invite them to join the platform.

Privileged roles

- The owner of the *StakeSpacePi* contract can add pools and can set their APRs & lock times.
- The owner of the Relationship contract can set the start and end times.

Recommendations

- Using NatSpec would make the project more professional and easier to understand. We recommend NatSpec formatted code annotations.

Risks

- The malicious user can invite alternate addresses in order to earn rewards.
- Anyone can use someone's invitation code without permission.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Failed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed

Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Failed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Failed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed

Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Findings

🟥🟥🟥🟥 Critical

C01. Access Control Violation

Users have the ability to invite themselves using an alternate wallet address. As a result, the *getParent()* function will return the wallet address of the same user, and the transfer destination will also be the user's wallet address. This allows them to pay 10% less than the total amount.

Inside the *StakeSpacePi* contract, a malicious user can invite an alternate wallet address and claim interest. In this case, the function will send the 10% of reward to the malicious user's alternate address.

This will lead malicious users to obtain a 10% referral amount.

Paths: ./contracts/StakeSpacePi.sol : claim()

./contracts/utils/Relationship.sol. binding()

Recommendation: Re-implement the logic and restrict to give invitation ability to only the owner of the contract.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Mitigated (The client stated this was a feature and mentioned it in their public documentation.)

🟧🟧🟧 High

H01. Access Control Violation

The inviter code is stored in the *mapping(bytes => address) private _codeUsed*. The stored values, including private variables, are transparent and can be accessed by anyone. This can be accessed directly from the *getInviteCode()* function.

This will lead to anyone can use someone else's invite code.

Path: ./contracts/utils/Relationship.sol: binding(), getInviteCode()

Recommendation: In order to protect sensitive data from unauthorized access, it is recommended not to store private data within the smart contracts on the blockchain, as the data can be accessed by anyone. Instead, a zero-knowledge proof mechanism can be implemented to verify and validate the private data without revealing it to any unauthorized parties. Zero-knowledge proof mechanisms allow for the validation of data without revealing the data itself, which is crucial for maintaining the privacy and security of sensitive data.

It can help that the private data remains protected from unauthorized access.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Mitigated (The client stated this was a feature and mentioned it in their public documentation.)

H02. EIP Standard Violation

The smart contract implements the *IERC20* interface, but the interface is not compatible with the actual *ERC20* token types. The *transferFrom* function defined in *IERC20* has a different signature than the transferFrom function defined in the *ERC20* standard. This can lead to incompatibility issues with *ERC20* tokens, causing the system to not support *ERC20* token types.

This will lead to incompatibility with the ERC20 tokens and the system will not support ERC20 token types.

Path: ./contracts/StakeSpacePi.sol

Recommendation: Implement the actual *IERC20* interface and transferFrom function from OpenZeppelin.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

H03. Requirements Violation

The smart contract assumes that one block is created every three seconds, which may not be accurate due to the varying block creation times on the blockchain network. This can result in the calculation of inaccurate *lockBlocks* and *perBlock* variables.

Block values are not precise, and the use of them can lead to wrong calculations.

Path: ./contracts/StakeSpacePi.sol : pending()

Recommendation: Instead of assuming a constant block generation time, use the *block.timestamp* value for calculations in the smart contract.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

H04. Highly Permissive Role Access

The contract proprietor has the ability to alter the staking period and APR subsequent to the user's token deposit. In the case of fraudulent activity, the proprietor may reduce the APR or extend the staking term, effectively locking the user's funds within the contract.

Path: ./contracts/StakeSpacePi.sol

Recommendation: The owner of the contract should not be able to change the staking time or APR after the user's deposit tokens. Limit the owner privileges.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

■■ Medium

M01. Best Practice Violation

The code does not use the *SafeERC20* library for checking the result of *ERC20 token* transfers. Tokens may not follow *ERC20* standards and return false in case of transfer failure or not return any value at all.

Paths: ./contracts/ERC721Distributor.sol : transferStranded()

./contracts/StakeSpacePi.sol: deposit(), withdraw(), claim()

Recommendation: Use the *SafeERC20* library to interact with tokens safely.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

M02. Data Consistency

The code does not include a check to ensure that the start and end times are valid. If the *beginsTime* is greater than the *endsTime*, or if the *block.timestamp* is greater than the *endsTime or beginsTime*, this can lead to unexpected behaviors.

Path: ./contracts/utils/Relationship.sol : setEnds(), setStart()

Recommendation: Implement necessary checks for setEnds() and setStart() functions.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (ab617c81d3d8fac8c86f6516109bbc0c1541f1bd)

M03. Missing Events

Critical state changes should emit events for tracking things off-chain.

The functions do not emit events on change of important values.

Paths: ./contracts/utils/Relationship.sol : setEnds(), setStart()

./contracts/StakeSpacePi.sol : addPool(), setPoolApr(), setPoolLocked(), setPool()

Recommendation: Emit events on critical state changes.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (ab617c81d3d8fac8c86f6516109bbc0c1541f1bd)

M04. Inconsistent Data - Variable Is Not Limited

Consider limiting the *apr* value in order to prevent unexpected calculation on staking rewards.

Path: ./contracts/StakeSpacePi.sol

Recommendation: Provide conscious limits for stored configuration values.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

M05. Collision Resistance Violation

The generated bytes32 hash is converted into bytes6 variable by taking hash's first 12 value. This is redundant when a longer and stronger hash is already created. Moreover, lowering hash digits reduces collision resistance. Same hash(bytes6) can be created for different addresses and due to the lack of hash existence validation, this can easily be executed.

Path: ./contracts/utils/Relationship.sol: _genCode()

Recommendation: Save the code hash as bytes32 and do not convert it.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Reported (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

M06. Bad Variable Naming

The variable *parent.indexes* is pointing to the *inviteeList* length, but the variable name *indexes* contradict the value it holds.

Path: ./contracts/utils/Relationship.sol : binding()

Recommendation: Provide variable names according to their purposes.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

M07. Unfinalized Code

setEnds and setStart functions and beginsTime and endsTime variables are declared but never used anywhere.

The functions setEnds and setStart are responsible for defining the start and end times within the contract. The variables beginsTime and endsTime, associated with these functions, are not utilized in any part of the contract's logic.

Path: ./contracts/StakeSpacePi.sol: setStart(), setEnds()

Recommendation: Remove the unused code or finalize its implementation.

Found in: 3ff2cc9417b9007f4da4183324a78734e7bc2c61

Status: Reported (EndsTime is used, but BeginsTime is still not used in the code.)

■ Low

L01. Floating Pragma

The project uses floating pragmas *^0.8.0*.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Paths: ./contracts/StakeSpacePi.sol

./contracts/utils/Relationship.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (Revised commit: ab617c81d3d8fac8c86f6516109bbc0c1541f1bd)

L02. Unfinalized Code

In the contract *StakeSpacePi* lines *42, 45, 46, 48, 49, 50, 51, 162*

are commented parts of code.

This reduces code quality.

Path:

./contracts/StakeSpacePi.sol

Recommendation: Remove commented parts of code.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

L03. Variables That Can Be Declared Constant

StakeSpacePi.inviteRewardRate, *StakeSpacePi.perBlockTime* and *Relationship.defaultCod* variables are declared once and never updated anywhere.

State variables that do not change their value should be declared constant to save Gas.

Paths: ./contracts/StakeSpacePi.sol, ./contracts/utils/Relationship.sol

Recommendation: Declare the variables as constant.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

L04. Redundant Use Of SafeMath

Since Solidity v0.8.0, the overflow/underflow check is implemented via ABIEncoderV2 on the language level - it adds the validation to the bytecode during compilation.

There is no need to use the SafeMath library.

Path: ./contracts/StakeSpacePi.sol

Recommendation: Remove the SafeMath library.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

L05. State Variables Default Visibility

Variable`s *perBlockTime* visibility is not specified. Specifying state variable`s visibility helps to catch incorrect assumptions about who can access the variable.

This makes the contract`s code quality and readability higher.

Path: ./contracts/StakeSpacePi.sol

Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

L06. Zero Valued Transactions

The function does not check if the input amount variable is greater than zero or not.

This can lead to a transaction with zero value being sent and will cause unnecessary Gas consumption.

Path: ./contracts/StakeSpacePi.sol : deposit()

Recommendation: Implement conditional checks for the zero-valued transaction.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

L07. Functions That Can Be Declared External

Some functions are declared public, although they are not called by any function. In order to save Gas, public functions that are never called in the contract should be declared as external.

Paths:
./contracts/StakeSpacePi.sol: poolLength(), setPoolApr(), setPoolLocked(), setPool(),

./contracts/utils/Relationship.sol: setEnds(), setStart(), binding(), getInviteeList(), getPlayerByCode(),getInviteCode()

Recommendation: Use the external attribute for functions never called from the contract.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

L08. Confusing Parameter Name

The parameter ‘address play’ is confusing because it represents a player/user address and the word ‘play’ is a verb.

Path: ./contracts/StakeSpacePi.sol: onlyUnLock(), onlyInvited(), pending()

Recommendation: Change the parameter name as player, user or etc.

Found in: ca0d4cc811051d6c004f453f4311e4465410d6cf

Status: Fixed (3ff2cc9417b9007f4da4183324a78734e7bc2c61)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.