HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Skies Verse
Date: 25 July, 2023

Document

Name	Smart Contract Code Review and Security Analysis Report for Skies Verse
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC20; ERC721; ERC1155; Staking;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://skiesverse.com/
Changelog	27.04.2023 - Initial Review 30.05.2023 - Second Review 22.06.2023 - Third Review 25.07.2023 - Fourth Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Skies Verse (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://gitlab.com/galaxy31/skiesverse-contracts
Whitepaper	Not provided.
Functional Requirements	Not provided.
Technical Documentation	Not provided.
Commit	ee3ffca19675c88bd96fe6bdcea7c94fdce6c233
Contracts	File: contracts/PoolGateway.sol SHA3: 255ae8c436e241ed32f4f394d725e04b277cf6989863ca89a54c32a3d37546cb File: contracts/Armor.sol SHA3: fb1d811175104f87d1ce4d4ebd7fa28187e6a6122ba2041ad6dbee8d0e4a0965 File: contracts/Hero.sol SHA3: e5ef44eb6941f03e9dcdc9ab3fef436e631d75694994e5f849080b0c8d142719 File: contracts/Weapon.sol SHA3: 856cd836c0160f3dc8895c0977cd64f7a97d6248ffaa550e68f5effc7ff96016 File: contracts/Potion.sol SHA3: 8fa5d81887a640f4d28ba5ba0c40c2e4bc4ecf3f9a42c21c6bbfab72c363e3ee File: contracts/Adaran.sol SHA3: 51a574c03a7dde781ddd8fd1bae73afbbdd9d57d86e53de00f85e90e6dde907d File: contracts/Nonces.sol SHA3: b74b03338215b42848123c385f49900b7e4413801ff6b52a83222d7f0eea384c File: contracts/Skies.sol SHA3: 2350c6d6aa9a4370611f30cb296932df978b0b50edd7aa5cb6e5b5504ac26457

Second review scope

Repository	https://gitlab.com/galaxy31/skiesverse-contracts
Whitepaper	Not provided.

Functional Requirements	Not provided.
Technical Documentation	Not provided.
Commit	4f3590a33038b6e79ae896d9644a1ceff3e9d261and
Contracts	File: contracts/PoolGateway.sol SHA3: ff7a34da10a038f2c3a432e25889bc0f9a6763e5f2dbe8a6f089b2959d9da991 File: contracts/Armor.sol SHA3: bd06435652bc31803cc4dc56cacc3092f7b6317ea5565de860e40765068fb3cd File: contracts/Hero.sol SHA3: cd6b64977d801b8b888ec3175569df01416e20a56b2da28e9142569f311c3af8 File: contracts/Weapon.sol SHA3: 351fad49b8b3641c68016d7c6cdfcdac8ff1d285fecf4a3fde49684d3197c62c File: contracts/Potion.sol SHA3: ce55f6defa004a47ffdc2fd5d29ddb00d88f082cb37886953938b358048644e5 File: contracts/Adaran.sol SHA3: fd93e60337a3938ec5ec005b5f83aafc2430a26cf728b198160b18f50ed79e19 File: contracts/Nonces.sol SHA3: 2f4e8315c471ef355af41ec48f18fc2001d781dde8d0b8c7c99747e1b6f08370 File: contracts/Skies.sol SHA3: 16afc30f182f3d29778b163f34cd627982a8382265486f4b4ef4ce4494286771

Third review scope

Repository	https://gitlab.com/galaxy31/skiesverse-contracts
Whitepaper	https://docs.google.com/presentation/d/1KwbpaH-VYhEYkWt9Z5QgxSTQZ5M_gGzcBq91EaQaj18/edit?usp=sharing
Functional Requirements	https://gitlab.com/galaxy31/skiesverse-contracts/-/blob/3a4e5f78e72e74f08bee6a8aa529930a9e7ebfb5/docs/Skies.docx
Technical Documentation	https://gitlab.com/galaxy31/skiesverse-contracts/-/blob/3a4e5f78e72e74f08bee6a8aa529930a9e7ebfb5/docs/Skies.docx
Commit	3a4e5f78e72e74f08bee6a8aa529930a9e7ebfb5
Contracts	File: contracts/PoolGateway.sol SHA3: 9461b6064177cc18aa720a08d707336aa7bc1f3eef293119c3369668898923c1 File: contracts/Armor.sol SHA3: 2f95c6da0de60795271523b3ba0316d6b34646631972679ed8f1b3ed447c5de0 File: contracts/Hero.sol SHA3: 289e3ed42a8c196b3fca9801dc804b927add78ff61bd23d0ef391af694eb6c3a

	File: contracts/Weapon.sol SHA3: c91c8026f9404b10e3fb843456b5089de09c1bf01aab63dcfb9b962d61c26b73 File: contracts/Potion.sol SHA3: cf04276e4ec0dc674de0b17407344672ecc401216dc835a5c58ca3c538fba996 File: contracts/Adaran.sol SHA3: d1f43ef76323f029f60eade9a2c2d3449e27d4144dd707fa25b0ec7f1b01aafa File: contracts/Nonces.sol SHA3: 9c4b2497fac5c82f61510bc98906662e4b850f5aca8b7bb5f1f5c1bd27aa399e File: contracts/Skies.sol SHA3: 1053f02e3dc30a7bce224802dd847f01e27b534d3e6a7f709d85da65bce85cea

Fourth review scope

Repository	https://gitlab.com/galaxy31/skiesverse-contracts
Whitepaper	Whitepaper.pdf
Functional Requirements	https://gitlab.com/galaxy31/skiesverse-contracts/-/blob/bd9a5456337272438a342c798b8afd940a9bd8da/docs/Skies.docx
Technical Documentation	https://gitlab.com/galaxy31/skiesverse-contracts/-/blob/bd9a5456337272438a342c798b8afd940a9bd8da/docs/Skies.docx
Commit	bd9a5456337272438a342c798b8afd940a9bd8da
Contracts	File: contracts/PoolGateway.sol SHA3: e739da7f300dd41e3aa4e1ebd77dc8b6e9293823c916182f56a901712d5b9a26 File: contracts/Armor.sol SHA3: 8c565cfb70152a905b57d955d907517f8a7e67d38065fff36a3e1d8c4d581c4c File: contracts/Hero.sol SHA3: 570e5d3fd4b7c15482b56df744bc8033143b1c95da642bff4f58cd92c9eae674 File: contracts/Weapon.sol SHA3: 1dd3684bb521341a237ef409713ce251425dac2e28566a54aeb6c067b12ba17f File: contracts/Potion.sol SHA3: 48a74902d600d535d214bfde1c1246f7cf75b13ec5b09bc97849ccb8cd6f5181 File: contracts/Adaran.sol SHA3: e7c978f20a6a90a0b9cec9d2135510285340be8a5f6f83bf5a992ec502f60c0b File: contracts/Skies.sol SHA3: 8d493e11050be76e083acffdff5030a5b1e20a9bca5a4dd57c3c41e6d28d299d

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 **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- Development environment is described.
- NatSpec is provided.

Code quality

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

- The development environment is configured.

Test coverage

Code coverage of the project is **95.37%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Security score

As a result of the audit, the code contains **3** low severity issues. The security score is **10** 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: **9.8**.

The system users should acknowledge all the risks summed up in the risks section of the report.

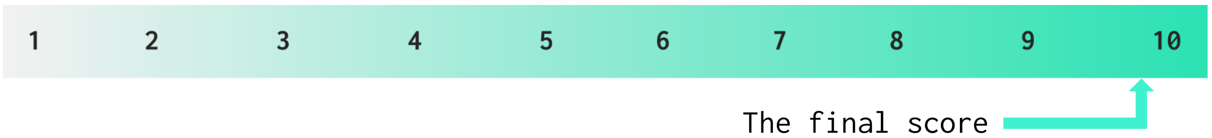

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
27 April 2023	13	6	2	6
30 May 2023	1	2	3	3
22 June 2023	4	2	0	1

25 July 2023	3	0	0	0

Risks

- Most of the interactions with the various ERC721, ERC20, ERC1155, and PoolGateway contracts involve a certain middleman who has the role of the SIGNER. If the address with this role loses access to their wallet, most of the interactions with the protocol will be rendered impossible to perform.
- There are upgradable contracts in the protocol; it is impossible to assure that this audit will still be valid for any version with code that differs in any way from this audit.
- The owner can set himself as a _deposit_allowed address and withdraw user funds that have been approved to the contract.

System Overview

Skies Verse is a mixed-purpose system with the following contracts:

- *Adaran (Token)* — A custom ERC20 token that allows minting by approved signers, based on EIP-712 signatures. It has the following attributes:
  - Name: Adaran
  - Symbol: ADR
  - Decimals: 18
  - Total supply: Unlimited
- *Armor* — An ERC721 token contract that allows minting of unique Armor tokens for ethers or Skies tokens, using EIP-712 signatures from approved signers. It also includes base token URI management.
- *Hero* — An ERC721 token contract that allows minting of unique Hero tokens for ethers or Skies tokens, using EIP-712 signatures from approved signers. It also includes base token URI management.
- *Nonces* — An abstract contract that provides nonce functionality for EIP-712 signatures, keeping track of nonces for each address and providing the DOMAIN_SEPARATOR.
- *PoolGateway* — A contract that allows users to stake Skies tokens and earn rewards. The contract handles deposits, withdrawals, and reward calculations, while managing the staking pools.
- *Potion* — An ERC1155 token that has the following attributes:
  - Name: Potion
  - Symbol: PTN
- *Skies* — An ERC20 token contract used for various purposes, including staking in the PoolGateway and minting Armor and Hero tokens. It has the following attributes:
  - Name: Skies
  - Symbol: SKS
  - Decimals: 18
  - Total supply: Depending on the constructor arguments.
- *Weapon* — An ERC721 token contract that allows minting of unique Weapon tokens for ethers or Skies tokens, using EIP-712 signatures from approved signers. It includes base token URI management.

Privileged roles

- Armor, Hero, Weapon, Potion contracts: Only the admin can change the base token URI (Role: DEFAULT_ADMIN_ROLE).

- PoolGateway contract: Only the admin can add or remove roles for the Developer and Signer role (Role: DEFAULT_ADMIN_ROLE).
- Adaran, Armor, Hero, Weapon, Potion contracts: Only approved signers can mint tokens using EIP-712 signatures (Role: SIGNER).
- PoolGateway contract: Only the developer can withdraw fees (Role: DEVELOPER)
- PoolGateway contract: Only approved signers can interact with the functions spendTokens(), getTokens() and addStake()

Recommendations

- In the poolGateway contracts token1 and token2 in the initialize() function are misleading names, it is possible to change them with more meaningful names.
- There are typos in the code. It says _withdrawed, TokensSpended and TokensWithdrawed; however, it should be _withdrawn, TokensSpent and TokensWithdrawn, it is recommended to fix the typos.

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.	Not Relevant
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.	Passed
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.	Not Relevant
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.	Passed
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.	Passed
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.	Not Relevant
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
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. Funds Lock

In the function *finishStake()* there is a require statement that will eventually cause funds lock, *stakes[msg.sender][id].finish_amount + _staked <= _totalSkiesBalance * STAKING_POOL / 100*, in this statement the variable *finish_amount + _staked* needs to be equal or less than *_totalSkiesBalance* multiplied by *STAKING_POOL* and divided by 100.

finish_amount is defined in the function *addStake()*, and it is equal to *amount* multiplied by the *reward* decided for that stake, *_totalSkiesBalance* is a sum of all the amounts added by the *addStake()* function, it is important to note that there is no subtraction to the *_totalSkiesBalance* when the various stakes are finished, *_staked* is a variable to which no value is assigned, so it is equal to 0, and *STAKING_POOL* is a constant that is equal to 50.

In the requirements, it is stated that the *amount* multiplied by the *reward*, called *finish_amount*, needs to be less than 50% of the total staked amount, this will eventually lead to funds lock, because if *_totalSkiesBalance* is high enough as a value, this will allow users to withdraw the tokens summed with the interest accrued with the stake, but the rewards might come from other users staked token, when those are finished, users will not be able to withdraw their original *amount* anymore, causing funds lock.

Path:
./contracts/PoolGateway.sol : finishStake()

Recommendation: Use a different logic in the *finishStake()* function that will prevent users from being able to withdraw other users' Skies token.

Remove the increment of *_totalSkiesBalance* in *addStake()* and check that there are enough finish tokens in *addStake()* instead of *finishStake()*

Found in: ee3ffca

Status: Fixed (Revised commit: 3a4e5f7)

C02. Data Consistency

There are flaws in the cash flow system within smart contracts, which can result in issues with token transfers both between and within contracts.

The amounts deposited to the PoolGateway contract from other contracts are not tracked in the *_totalSkiesBalance* variable, resulting in the contract not being able to track the earned tokens correctly.

The amount of tokens staked by the users is not tracked inside the *_staked* variable. This results in a lack of validation of user-deposited funds and unauthorized extraction of the users' funds from the contract.

The amount of rewards given to users is not tracked in the *_rewarded* variable. This results in an inability to track rewards paid to users and limits the overall rewards given to the required 20% of the earned tokens.

There is also a lack of tracking of the staking rewards given to the users, leading to an incorrect calculation of the funds that should be used for the staking rewards, and should be limited to 50% of the earned tokens.

The cash flow system is full of invalid calculations based on the invalid values from the global variables.

Path:
./contracts/PoolGateway.sol

Recommendation: Update the cash flow system of the PoolGateway contract to meet the requirements and prevent the extraction of funds belonging to the stakers.

Found in: ee3ffca

Status: Fixed (Revised commit: 3a4e5f7)

C03. Denial of Service

In the *getTokens()* function of the PoolGateway contract, the *safeTransferFrom* is used incorrectly.

The *safeTransfer* function should be used instead, as the *safeTransferFrom* requires the *approval* of the funds before transfer, and this is not done inside the contract.

This function will always revert with insufficient allowance when called leading to Denial of Service.

Path:
./contracts/PoolGateway.sol : getTokens()

Recommendation: Use *safeTransfer*.

Status: Fixed (Revised commit: 4f3590a3)

C04. Funds Lock

In the *permitMintForEthers()* function, users can transfer more ETH than required; however, those funds are locked inside the contract and are lost, as only the value amount is forwarded to the *_pool* address.

Additionally, there is no refund function that will allow users to withdraw their funds in case of overpayment.

Paths:
./contracts/Armor.sol : permitMintForEthers()
./contracts/Hero.sol : permitMintForEthers()
./contracts/Weapon.sol : permitMintForEthers()

Recommendation: Validate strictly that msg.value == value, or implement a refund system for users who transferred more than required.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

C05. Funds Lock

In the *permitMint()* function, the Skies tokens are transferred directly to the PoolGateway contract using the safeTransferFrom function.

This transfer is not accounted for in any way in the PoolGateway contract, and the _totalSkiesBalance variable is not updated by the earned amount.

The transferred Skies tokens will be locked inside the PoolGateway contract.

Paths:
./contracts/Potion.sol : permitMint()
./contracts/Armor.sol : permitMint()
./contracts/Hero.sol : permitMint()
./contracts/Weapon.sol : permitMint()

Recommendation: Interact with the PoolGateway contract directly, implement a deposit function that will update the storage variables with occurred earnings.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

C06. Unauthorized Access

In the *finishStake()* function, there is no update on the amount of withdrawn funds, and any user who deposited to the staking can run this function with the same *id* parameter an unlimited number of times.

This results in the possibility of draining all the Skies tokens from the PoolGateway contract.

Path:
./contracts/PoolGateway.sol : finishStake()

Recommendation: Mark user stakes as withdrawn when the *finishStake()* function is called to prevent unlimited withdrawals.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

C07. Funds Lock

There are 3 ways of depositing Skies token into the contract: addStake(), spendTokens(), depositToken(), when creating a stake the totality of the amount of the stake is added to _totalSkiesBalance, from the _totalSkiesBalance it is possible to withdraw only 50% from the function finishStake(), this assumes that the flow of funds will remain constant with the 3 ways of depositing tokens.

If the flow of funds does not remain constant the users will not be able to finish their stake because the variable *_totalSkiesBalance* will not increment.

The users deposited funds should not be mixed in with the withdrawable rewards and there should be a way for users to withdraw their tokens safely at the end of the stake even if there are no rewards.

Path:
./contracts/PoolGateway.sol : finishStake();

Recommendation: Change the logic of the contract from a *_totalSkiesBalance* to a system with multiple variables that represent diverse kinds of Skies token, for example, a variable to keep track of the staked tokens, a variable to keep track of the rewards that arrive with *depositTokens()*, and during the flow of the contract do not allow anyone, but the user that deposited to withdraw the Skies tokens.

Found in: 4f3590a3

Status: Fixed (Revised commit: 3a4e5f7)

C08. Wrong Logic

When adding a stake it is taken into account *amount * reward / 100 + _unstaked +_stakingPoolLocked <= _stakingPool* this way the rewards and staking pool token are mixed up in the calculation to stake, it should be checked if there are enough reward tokens to be given, not if there are enough tokens stake to be withdrawn.

Path:
./contracts/PoolGateway.sol : addStake(), finishStake();

Recommendation: Check if there are enough token to reward the stake and do not touch or modify the staked pool; if a user deposits there, he should just be able to withdraw from there, no calculations should be performed on that pool.

Found in: 3a4e5f7

Status: Fixed (Revised commit: bd9a545)

■■■ High

H01. Highly Permissive Role Access

In the *getTokens()* function in the PoolGateway contract, if it works correctly after the fix to the C03 issue, the role *SIGNER* can sign a transaction to transfer funds belonging to the stakers.

There is no validation check that prevents the withdrawal of funds that were staked in the PoolGateway or the staking rewards given.

Only funds collected as payments in the system and less than 20% of the REWARD_POOL threshold, in the case of Skies token, should be able to be withdrawn in the *getTokens()* function.

Path:
./contracts/PoolGateway.sol : getTokens()

Recommendation: Add proper validation to the *getTokens()* function to limit the amount of rewards distributed to no more than 20% of collected earnings as in the requirements.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

H02. Highly Permissive Role Access

In the *withdrawSkies()* function in the PoolGateway contract, the *DEVELOPER* role can withdraw funds that belong to the stakers.

The *_totalSkiesBalance* variable is increased inside the *addStake()* function by the amount of stakers' deposited funds. The calculations of the *DEVELOPER*'s earnings inside the *withdrawSkies()* function are done based on the incorrect value of the *_totalSkiesBalance*, resulting in the ability to withdraw more than was actually collected from earnings.

Path:
./contracts/PoolGateway.sol : withdrawSkies()

Recommendation: Fix the flaws in the cash flow system and update the *_totalSkiesBalance* correctly. Privileged roles of the system should have no access to user-deposited funds or given rewards for staking.

Found in: ee3ffca

Status: Fixed (Revised commit: 3a4e5f7)

H03. Undocumented Functionality

The functions *spendTokens()* and *getTokens()* are not documented but are vital for the normal flow of the contract

Path:
./contracts/PoolGateway.sol : spendTokens(), getTokens();

Recommendation: Document the functionalities or remove them.

Found in: 4f3590a3

Status: Mitigated (They will be needed to update the backend, it is not possible to ensure fully the functionality in the scope of this audit)

H04. Arbitrary “From”

An argument of the function *depositToken()* is the address *from* which the tokens are being transferred, since there is no restriction to who can call that function, if a user has approved the tokens to be spent on the contract, the tokens could be sent from his address to the contract and withdrawn as rewards by other users/developer address.

Path:
./contracts/PoolGateway.sol : depositToken();

Recommendation: The NFT contracts that call *depositToken()* should be the only contracts that can call *depositToken()*, the user should call a function inside the NFT contract that should receive the tokens and subsequently send it to *PoolGateway.sol*

Found in: 4f3590a3

Status: Mitigated (The arbitrary “from” is mitigated, the owner of the contract can still set himself as a rightfully “from” address and withdraw funds, it is up to the owner to not attack maliciously the contract, added as a risk also.)

■■ Medium

M01. Best Practice Violation

In the *initialize()* function, the *AccessControlUpgradeable.sol* is never initialized.

All upgradeable contracts should be initialized properly.

When working with upgradable smart contracts, it is best practice to use *_disableInitializers()* in the implementation constructor.

Path:
./contracts/PoolGateway.sol

Recommendation: Initialize the *AccessControlUpgradeable* inside the *initialize()* function of the *PoolGateway.sol* contract. Add a constructor() with *_disableInitializers()* function inside.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

M02. Usage of Built-in Transfer

The built-in transfer and send functions process a hard-coded amount of Gas. In case the receiver is a contract with receive or fallback function, the transfer may fail due to the “out of Gas” exception.

Path:
./contracts/PoolGateway.sol : withdrawEthers()

Recommendation: Replace transfer and send functions with call.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

M03. Contradiction

According to the NatSpec comment of *permitMint()* the *to* parameter should be different from the 0 address. However, in the function, the validation is missed.

According to the NatSpec comment of *permitMint()* the *value* parameter should be different from 0. However, in the function, the validation is missed.

This can lead to unexpected value processed by the contract.

Path:
./contracts/Adaran.sol : permitMint()

Recommendation: Implement the validations according to the NatSpec comment.

Found in: ee3ffca

Status: Fixed (Revised commit: 3a4e5f7)

M04. Missing Validation

In *spendTokens()* the *from* parameter should be equal to *_msgSender()*. However, in the function, the validation is missed.

In *spendTokens()* the *value* parameter should be higher than 0. However, in the function, the validation is missed.

This can lead to unexpected value processed by the contract.

Path:
./contracts/PoolGateway.sol : spendTokens()

Recommendation: Implement missing validations.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

M05. Inconsistent Data

Value used in the event *TokensSpended* is the function parameter *from* but in the safeTransferFrom the *_msgSender()* is used. There is not any validation of *from* parameter and these two values can be different.

This may lead to wrong assumptions on the front-end about the current contract state.

Path:
./contracts/PoolGateway.sol : spendTokens()

Recommendation: Keep the data emitted in the events with the data present in the functions.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

M06. Missing Validation

In *addStake()* the *user* parameter should be different from the 0 address. However, in the function, the validation is missed.

In *addStake()* the *start* parameter should be lower than the *end* parameter. However, in the function, the validation is missed.

In the *addStake()* function, there is no validation to check if there are sufficient rewards to distribute for the stake being added.

This can lead to unexpected value processed by the contract.

Path:
./contracts/PoolGateway.sol : addStake()

Recommendation: Implement missing validations. Consider adding *(amount * (100 + reward) / 100) - amount + _stakeRewarded <= _totalSkiesBalance * STAKING_POOL / 100* to ensure that there are enough tokens for rewards, where _stakeRewarded is a helper global variable to track stake rewards gifted.

Found in: ee3ffca

Status: Fixed (Revised commit: bd9a545)

M07. CEI Pattern Violation

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The Checks-Effects-Interactions pattern is violated. During the functions, some state variables are updated after the external calls.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during implementation of new functionality.

Paths:
./contracts/PoolGateway.sol : addStake(),

./contracts/Armor.sol : permitMint(), permitMintForEthers(),

./contracts/Hero.sol : permitMint(), permitMintForEthers(),

./contracts/Weapon.sol : permitMint(), permitMintForEthers()

Recommendation: Follow common best practices, and implement the functions according to the Checks-Effects-Interactions pattern.

Found in: 3a4e5f7

Status: Fixed (Revised commit: bd9a545)

Low

L01. Unused Variable

The variables *_adaran_token*, *REWARD_POOL* and *_rewarded* are never used.

Path:
./contracts/PoolGateway.sol

Recommendation: Remove unused variables or update the code with missed functionality.

Found in: ee3ffca

Status: Fixed (Revised commit: 3a4e5f7)

L02. 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/PoolGateway.sol
./contracts/Armor.sol
./contracts/Adaran.sol

./contracts/Potion.sol
./contracts/Weapon.sol
./contracts/Hero.sol
./contracts/Nonce.sol
./contracts/Skies.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: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L03. Style Guide Violation

The provided projects should follow the official guidelines.

Paths:
./contracts/PoolGateway.sol
./contracts/Armor.sol
./contracts/Adaran.sol
./contracts/Potion.sol
./contracts/Weapon.sol
./contracts/Hero.sol

Recommendation: Follow the official Solidity guidelines.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L04. Redundant Import

The import of *Initializable.sol* is unnecessary for the contract.

Initializable is already inherited by *AccessControlUpgradeable* and *EIP712Upgradeable*.

Path:
./contracts/PoolGateway.sol

Recommendation: Remove the redundant import.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L05. Functions That Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

Path:
./contracts/PoolGateway.sol : initialize()

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

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L06. Inefficient Gas Model

The variable *_staked* is declared and used in the function *finishStake()*, but no value is assigned to it.

Path:
./contracts/PoolGateway.sol

Recommendation: Remove the variable or assign it a value.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L07. Using Storage Instead Of Memory

Using the local storage variable will not allocate memory for its value but instead will make calls to the storage each time accessing it.

Path:
./contracts/PoolGateway.sol : finishStake()

Recommendation: Use memory for a local variable to save Gas and then change updated values to the state at the end of the function.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L08. Naming Convention

The *_PERMIT_TYPEHASH* variables are misleading and reduce code readability, the proper naming convention when working with the EIP712 is to name them with function name + TYPEHASH for example *permitMint* should be *PERMIT_MINT_TYPEHASH*, *spendTokens* *SPEND_TOKENS_TYPEHASH*. Those names are too similar and can be mistaken.

Paths:
./contracts/PoolGateway.sol
./contracts/Armor.sol
./contracts/Adaran.sol
./contracts/Potion.sol
./contracts/Weapon.sol
./contracts/Hero.sol

Recommendation: Give variables more meaningful names to increase readability.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L09. Contradiction

NatSpec is contradicting the code; a lot of NatSpec comments refer to the owner’s approval, when in fact, it is the signer's approval.

Paths:
./contracts/Weapon.sol : permitMintForEthers(), permitMint()
./contracts/Potion.sol : permitMint()
./contracts/Armor.sol : permitMintForEthers(), permitMint()
./contracts/Adaran.sol : permitMint()
./contracts/Hero.sol : permitMintForEthers(), permitMint()

Recommendation: Fix the contradiction in the NatSpec.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L10. No Messages In Require Conditions

The require condition can be used to check for conditions and throw an exception if the condition is not met. It is possible to provide a message string for require. Without providing a string argument to require, it will revert with empty error data, not even including the error selector.

Paths:
./contracts/Armor.sol : permitMintForEthers()
./contracts/Hero.sol : permitMintForEthers()
./contracts/Weapon.sol : permitMintForEthers()

Recommendation: Some require statements are missing error messages. This makes code harder to test and debug.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L11. Redundant Block

The usage of *`virtual`* is unnecessary in some functions.

Paths:
./contracts/Hero.sol : _baseURI(), tokenURI(), supportInterface()
./contracts/Armor.sol : _baseURI(), tokenURI(), supportInterface()
./contracts/Potion.sol : _baseURI(), uri(), supportInterface()
./contracts/Weapon.sol : _baseURI(), tokenURI(), supportInterface()

Recommendation: Remove the redundant code block.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L12. Misleading Error Messages

A message in a require condition is misleading.

This makes code harder to test and debug.

Path:
./contracts/Adaran.sol : permitMint()

Recommendation: Refactor the message in the require conditions to fit code behavior.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L13. Unused Imports

The import of *Ownable.sol* is unnecessary for the contract.

Path:
./contracts/Nonces.sol

Recommendation: Remove the redundant import.

Found in: ee3ffca

Status: Fixed (Revised commit: 4f3590a3)

L14. Unused Argument

Inside *addStake()* there is a check *require(user == _msgSender(), "Illegal sender");* that checks if the passed address is equal to *msg.sender*. If this check is present, then user argument is redundant, because the same value can be accessed with *_msgSender()*. Unused arguments should be removed from the contracts. This will help lower the Gas cost.

Path:
./contracts/PoolGateway.sol : addStake()

Recommendation: Remove redundant argument.

Found in: 3a4e5f7

Status: Fixed (Revised commit: bd9a545)

L15. Variables That Should Be Declared Constant

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

Path:
./contracts/Potion.sol : name, symbol

Recommendation: Declare the above-mentioned variables as constants.

Found in: 3a4e5f7

Status: Reported

L16. Commented Code Parts

In the contract *Skies* lines 167-169 are commented parts of code.

This reduces code quality.

Path:
./contracts/Skies.sol : lock()

Recommendation: Remove commented parts of code.

Found in: 3a4e5f7

Status: Reported

L17. Redundant Mathematical Operation

The mathematical operation *require(paymentPlan <= paymentPlans.length - 1)* is redundant. The same check can be performed without mathematical operation: *require(paymentPlan < paymentPlans.length)*

Path:
./contracts/Skies.sol : planNotRevoked()

Recommendation: Remove redundant mathematical operations.

Found in: 3a4e5f7

Status: Reported

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.