HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Blockscape Finance AG
Date: March 10, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Blockscape Finance AG
Approved By	Yevhenii Bezuhlyi | Head of SC Audits at Hacken OU
Type	NFT token; Staking
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://blockscape.network/
Changelog	10.03.2023 - Initial Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Blockscape Finance AG (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 review and security analysis of the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/BlockscapeNetwork/rocketscape/tree/main/src
Commit	5219b39c6c74a0d3a533a6f06fe589e574744da4
Whitepaper	-
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./src/BlockscapeETHStakeNFT.sol SHA3: 9d4933b2c299dbbff5a4c2ad745884c3bd418621af460d09efef12eb995f6945 File: ./src/BlockscapeValidatorNFT.sol SHA3: f8988656d7ce9a2aff129063b82974b2ae8f814eb19e193659bd4944743cbdd6 File: ./src/utils/RocketNodeStakingInterface.sol SHA3: 0c51c1a4c125d3a3f4eadd5beb51eed0e36e760ec473d4c7c94e83e086c23cfe File: ./src/utils/RocketStorageInterface.sol SHA3: 420753d03e17da97d1c36aa0cb9d457e1c708d7d31a24bbcb4266203f364dd1b

Second review scope

Repository	https://github.com/BlockscapeNetwork/rocketscape/tree/main/src
Commit	73fe19ee5b8c62af2c04b22ee7a6b2972ce1aa2e
Contracts	File: ./src/BlockscapeETHStakeNFT.sol SHA3: f1dca8384363cc7fec6ee657732d3798b855200b59d283cc6288af06f641d781 File: ./src/BlockscapeValidatorNFT.sol SHA3: 4dcf51470864fc5f88302b04d3d9119b22e7d6880ed8f6483b325aab3cac3f77 File: ./src/utils/BlockscapeShared.sol SHA3: 26f3d6eb98c74ad86326c860672ee7478f89e51648e55ccd2fb63338169c3072 File: ./src_1_audit_submission/utils/RocketNodeStakingInterface.sol SHA3: 74442e513072fa859af07b625b799047b563e6217e95130e8780b920502decc4

	File: ./src_1_audit_submission/utils/RocketStorageInterface.sol SHA3: ccee66f464ec6f1b44ad69e7b3f6d3a724724c67eff9ddb103c804dc1a9bdf05

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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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

- Functional requirements are actually API descriptions and a Readme file. These can be made more detailed to a non-technical user.
- Technical description consists of deployment instructions.
- Code is documented with NatSpec.
- Running tests requires a lot of extra configuration. Testing should be possible to be executed without providing your own API keys.

Code quality

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

- Code follows language style guidelines and best practices.
- The development environment is configured.

Test coverage

Code coverage of the project is **59.68%** (branch coverage).

- Basic user interactions are covered with tests.
- Negative cases coverage is missing.
- Interactions with several users are not tested thoroughly.

Security score

As a result of the audit, the code contains **no** 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: **8.0**.

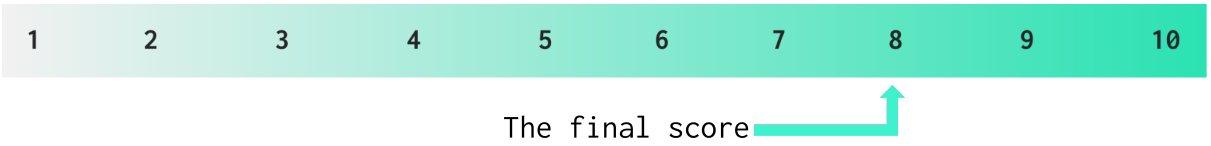

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
10 March 2023	23	6	2	1

10 April 2023	0	0	0	0

System Overview

Blockscape Finance is an NFT staking system with the following contracts:

- *BlockscapeValidatorNFT* — an NFT issued to a user who stakes 16 ETH with Blockscape. This NFT represents a Validator on the Blockscape platform. The user can monitor their validator performance, rewards, and penalties through the NFT.
- BlockscapeETHStakeNFT - an NFT issued to a user who stakes any amount of ETH. This NFT entitles the user to participate in a pool of stakers that are staking for them on the Blockscape platform.

Privileged roles

- The owner of the *BlockscapeValidatorNFT* contract can arbitrarily withdraw funds, stop the staking period, update the validator for a given token ID, set a withdrawal fee and change limits.
- The owner of the *BlockscapeETHStakeNFT* contract can withdraw funds or change the withdrawal fee at any time.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	Not Relevant
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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	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.	Not Relevant
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.	Not Relevant
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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Requirements Violation

The function *userRequestWithdraw()* is supposed to send the user tokens when unstaking. However the function doesn't contain any logic for sending funds by itself and the user is left at the mercy of the backend.

1) As Smart Contracts are meant to be trustless by design, this logic defeats the purpose of on-chain code.

2) the *userRequestWithdraw()* function is a non-*view* function and returns a value which needs to be consumed by the Back-end code. This is a bad practice since the Back-end will only be able to get the transaction receipt, but not the return value itself. The proper way to consume data from a non-view function is by listening to events.

Paths:

./src/BlockscapeValidatorNFT.sol : userRequestWithdraw()

Recommendation: 1) Implement sending withdrawn funds from the Smart Contract. 2) If any return value is needed - send an event instead of returning a value.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed (the function *prepareWithdrawalProcess* is now added and implements the functionality properly)

■■■ High

H01. Highly Permissive Role Access

The owner of the contract (the deployed) can withdraw any amount of the money at any moment in time. This can lead to loss of funds for users.

Paths:

./src/BlockscapeValidatorNFT.sol : withdraw();

Recommendation: Implement constraints on the owner's privilege to withdraw the contract's funds.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed (the *withdrawFunds* function implements the process without the third-party governance)

H02. Highly Permissive Role Access

The owner of the contract can set a withdrawal fee at any value and at any time. This can mislead users over expected withdrawal of funds and cause withdrawal amount manipulation.

Paths:

./src/BlockscapeValidatorNFT.sol : setWithdrawFee();

./src/BlockscapeValidatorNFT.sol : setWithdrawFee();

Recommendation: Implement constraints on the owner's privilege to change the withdrawal fee and document it or store the withdrawal fee together with token metadata.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed (the function *changeWithdrawFee* sets a limit on increasing fee up to 20% maximum)

Medium

M01. Wrong Data Type

The state variable *tokenIDtoValidator* is expected to store a mapping of token IDs to validator addresses. The fact that it must be an address and not a bytes array is reinforced by the NatSpec comments, e.g. in BlockscapeETHStakeNFT:getMetadata() - "@return the validator address". There is no justification for storing addresses as a dynamic bytes array, as it increases Gas usage and complexity unnecessarily.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol;

Recommendation: Change the datatype of *tokenIDtoValidator* to *mapping(uint256 => address)* and update its occurrence in the code.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

M02. Misleading Documentation

The function updateValidator() is documented by NatSpect as "set validator address for given token id". However this function allows setting a validator only once for a given tokenID.

Paths:

./src/BlockscapeValidatorNFT.sol : updateValidator();

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed (the function can still be called only once, but this behavior is explicitly stated in the NatSpec.)

M03. Requirements Violation

According to the documentation: “how much fees would the user have to pay if he would unstake now”. However if the user calls this function with a Token ID which has 0 balance of a non-existing Token ID, the function will return the value of *initWithdrawFee*, which equals to *20 * 1e18* initially, while logically it should be 0.

Paths:

./src/BlockscapeValidatorNFT.sol : viewUserRequestWithdraw();

Recommendation: Reconsider the logic of the function.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed (functionality moved to *calcWithdrawFee* and returns 0 in the said scenario)

M04. Writing Blank Data

The function *_metadataValidatorNFTInternal()* writes blank bytes to *tokenIDtoValidator[_tokenID]*. This might be an error, because the function is called inside the external function *depositValidatorNFT()*, which as the name implies requires that the validator is set.

Paths:

./src/BlockscapeETHStakeNFT.sol : _metadataValidatorNFTInternal();

./src/BlockscapeValidatorNFT.sol : _metadataValidatorNFTInternal();

Recommendation: Reconsider the logic of the function.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed (functionality removed in *BlockscapeETHStakeNFT*, and moved in *BlockscapeValidatorNFT* to *estRewardsNoMEV* where the said problem is not encountered.)

M05. Bad Practice

The function *updateValidator()* checks for RPL in the node, but in case they are insufficient it emits an event instead of reverting. This may lead to a false assumption on the on-chain that the method was executed correctly and all gas consumed before being lost instead of being returned.

Paths:

./src/BlockscapeValidatorNFT.sol : updateValidator();

Recommendation: Use a revert statement instead of emitting an event in case of insufficient staked RLP.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed (the logic has been changed to read the state variable vaultOpen instead and revert in the unhappy path)

M06. Using transfer() to Send Native Tokens

It is considered a bad practice to send native tokens using withdraw() and send() since the Istanbul fork, because those functions rely on Gas limitation and can give an unclear error in gas of insufficient funds to send. If the owner Smart Contract uses any logic in *receive()* or sole *payable fallback()*, this may lead to failed executions.

Paths:

./src/BlockscapeValidatorNFT.sol : withdraw();

Recommendation: Use the sendValue() function from OZ Utils library.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

Low

L01. Floating Pragma

The project uses floating pragmas: “>0.5.0 <0.9.0” and “^0.8.16” . Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol;
./src/utils/RocketNodeStakingInterface.sol;
./src/utils/RocketStorageInterface.sol;

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L02. Style Guide Violation - Constant Naming

The provided project violates the official guidelines: constant names should be in all-uppercase with underscore as the word separator.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol;

Recommendation: Rename constants according to the naming conventions, e.g. *rocketStorage* to *ROCKET_STORAGE*. Follow the official Solidity Style Guide: https://docs.soliditylang.org/en/v0.8.17/style-guide.html.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L03. Unused Variables

Unused variables should be removed from the contracts. Unused variables are allowed in Solidity and do not pose a direct security issue. However, it is best practice to avoid them as they can cause an increase in computations (and unnecessary gas consumption) and decrease readability.

The project files contain *name* and *symbol* constants. Unless these values are parameterized, which they are not, there is no value in keeping those values on-chain. Values such as contract name are available via block explorer, while storing *symbol* makes sense only if the contract implements a standard, where this variable can be used.

Paths:

./src/BlockscapeETHStakeNFT.sol : name, symbol;

./src/BlockscapeValidatorNFT.sol : name, symbol;

Recommendation: Remove constant values that are not used on-chain.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Mitigated (the client has decided to retain these values to be read from on-chain)

L04. State Variables Default Visibility

Variables' *blockscapeRocketPoolNode*, *initWithdrawFee*, *curETHlimit*, *allowPubDeposit*, *tokenID*, *rocketNodeStakingAddress*, *rocketNodeStaking*, *tokenIDtoMetadata*, *tokenIDtoValidator* visibility is not specified. Specifying state variables visibility helps to catch incorrect assumptions about who can access the variable. This improves the contract`s code quality and readability.

The explicit visibility makes it easier to catch incorrect assumptions about who can access the variable.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol;

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

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L05. Redundant Zero Value Initialization

Variables *allowPubDeposit, poolSupply* are initialized with their default data type zero value. This assignment adds no value, since unassigned variables are assigned their zero values by default in Solidity. Making an explicit zero value assignment increases Gas cost unnecessarily.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol;

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

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L06. Redundant State Variable

The state variable *rocketNodeStakingAddress* is never used directly in the code except for assignment to another state variable *rocketNodeStaking*. The state variable *rocketNodeStakingAddress* can be removed altogether in order to save Gas.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol;

Recommendation: Remove the state variables *rocketNodeStakingAddress* and make a direct assignment to *rocketNodeStaking*.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L07. Unused Fields in Struct

The struct *Metadata* contains 5 fields, of which 3 - *institution*, *institutionName*, *institutionVerified* are never accessed or written.

This increases the Gas cost because of redundant storage writing and might be a sign of unfinished code.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol;

Recommendation: Remove the unused fields of the struct *Metadata* or revise its usage in the code.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L07. State Variables Can Be Declared Immutable

The state variables *rocketStorage* and *rocketNodeStaking* are never modified in the code. These variables can be made immutable or constant.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol;

Recommendation: Make the state variables, which are never changed after the deployment, immutable or constant.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L08. Unindexed Events

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Paths:

./src/BlockscapeETHStakeNFT.sol : RPLStakeRequired, RocketPoolNodeAddressChanged, StakeUpdated;

./src/BlockscapeValidatorNFT.sol : RPLStakeRequired, UserRequestedWithdrawal, ETHLimitChanged, RocketPoolNodeAddressChanged;

Recommendation: Make the state variables, which are never changed after the deployment, immutable or constant.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

L09. Code Duplication

Many functions, events and state variables are duplicated in both Smart Contracts, among some of them: name, symbol, tokenID, userRequestFullWithdraw() and others. It is a violation of software development best practice and increases the complexity of maintaining the codebase.

Paths:

./src/BlockscapeETHStakeNFT.sol;

./src/BlockscapeValidatorNFT.sol

Recommendation: Move the recurring logic and interface to abstract contracts and higher-level interfaces respectively.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L10. Redundant Usage of Reentrancy Guard

The contracts use OZ's *nonReentrant* modifier on functions where there are no external calls. This practice adds no additional security, but increases Gas usage unnecessarily.

Paths:

./src/BlockscapeValidatorNFT.sol : depositValidatorNFT();

./src/BlockscapeETHStakeNFT.sol : depositStakeNFT();

Recommendation: Remove the *nonReentrant* modifier from functions which don't have external calls.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L11. Unused Function Parameter

The function *userRequestWithdraw()* declares the *_amount* return parameter, however it is never returned or used.

Paths:

./src/BlockscapeValidatorNFT.sol : userRequestWithdraw();

Recommendation: Remove the unused parameter.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L12. Missing Events

Events for critical state changes should be emitted for tracking things off-chain.

Paths:

./src/BlockscapeValidatorNFT.sol : setWithdrawFee(), closeValidatorNFT(), updateValidator(), openValidatorNFT();

./src/BlockscapeETHStakeNFT.sol : setWithdrawFee();

Recommendation: Create and emit related events.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L13. Potential Underflow Possibility

The arithmetic equation *nodeRPLStake* - *minimumReqRPL* might throw an underflow error if the *minimumReqRPL* is greater than nodeRPLStake. This can block using this view function including for the Front-end.

Paths:

./src/BlockscapeValidatorNFT.sol : hasNodeEnoughRPLStake();

Recommendation: Implement a check to guarantee that *minimumReqRPL* will never be greater than *nodeRPLStake* in the equation.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L14. Misleading Documentation

According to the documentation - “how many stakers are there totally”, however the function does intend to return the total supply of all tokens.

Paths:

./src/BlockscapeValidatorNFT.sol : totalSupply();

./src/BlockscapeETHStakeNFT.sol : totalSupply();

Recommendation: Update the NatSpec documentation to mention that the function returns the total supply of all tokens.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L15. Redundant External Call

The functions contain calls to other functions in the same contract through the keyword “this”. This turns a cheap code jump into an external call, increasing Gas usage.

Paths:

./src/BlockscapeValidatorNFT.sol : totalSupply();

./src/BlockscapeETHStakeNFT.sol : totalSupply();

Recommendation: Remove the keywords “this” before calls to the functions inside the same contract.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L16. Commented code parts

In the contract BlockscapeValidatorNFT, on line 470 there are commented parts of code.

Paths:

./src/BlockscapeValidatorNFT.sol : _metadataValidatorNFTInternal();

Recommendation: Remove the commented code parts

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L17. Typos in Documentation

The BlockscapeETHStakeNFT contract contains typos on lines 29, 111 and 360.

Paths:

./src/BlockscapeETHStakeNFT;

Recommendation: Fix the typos

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L18. Style Guide Violation

The function *viewuserRequestFullWithdraw()* should be renamed to *viewUserRequestFullWithdraw()* to comply with the Stolidity Style Guide.

Paths:

./src/BlockscapeETHStakeNFT : viewuserRequestFullWithdraw();

Recommendation: Rename the function to *viewUserRequestFullWithdraw()*.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L19. Error in Documentation

According to NatSpec, the function getMetadata() must return “the validator address”, however it returns a Metadata object and a dynamic bytes array.

Paths:

./src/BlockscapeETHStakeNFT : getMetadata();

Recommendation: Update the NatSpect of the function.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L20. Redundant Use of Assembly

The contracts contain several small assembly blocks, which do not perform any operations feasible only using the Yul language, but for trivial operations like accessing or incrementing state variables. This pattern doesn’t add value, but makes maintaining and understanding the code more difficult for no reason.

Paths:

./src/BlockscapeValidatorNFT.sol;

./src/BlockscapeETHStakeNFT.sol;

Recommendation: Replace assembly blocks with Solidity unless assembly provides an advantage.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L21. Empty Revert Messages

The revert conditions are written using assembly code which contains commands revert(0, 0). This will make the contract code revert with empty error data, not even including the error selector and might complicate debugging and user experience.

Paths:

./src/BlockscapeValidatorNFT.sol;

./src/BlockscapeETHStakeNFT.sol;

Recommendation: Replace assembly blocks with Solidity, provide error data in the revert clauses.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L22. Unused Event

The BlockscapeETHStakeNFT contract contains an event *RPLStakeRequired* which is not used in the code. This is likely a consequence of copying the code of *BlockscapeETHStakeNFT*. This leaves redundant logic in code and extra bytes to deploy with the contract.

Paths:

./src/BlockscapeETHStakeNFT.sol : RPLStakeRequired;

Recommendation: Remove the unused event.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

L23. Storage Abuse

The function updateValidator() always calls openValidatorNFT() in case of a happy path. This performs a redundant duplicated check of RPL stake and in case of *allowPubDeposit = true* - a redundant storage writing. Since writing to storage is one of the most Gas-expensive operations, this leads to (in case of *allowPubDeposit = true*) unnecessary expensive Gas consumption.

Paths:

./src/BlockscapeValidatorNFT.sol : updateValidator();

Recommendation: Perform a check of *allowPubDeposit == false* inside *updateValidator()* and update *allowPubDeposit* only if required and directly without calling *openValidatorNFT()*.

Found in: 5219b39c6c74a0d3a533a6f06fe589e574744da4

Status: Fixed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.