verichains

SECURITY AUDIT OF

BRIKY LAND SMART CONTRACTS

Public Report

Aug 01, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Aug 01, 2024. We would like to thank the Briky Land for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Briky Land Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team identified some vulnerable issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Briky Land Smart Contracts

Briky Land is a Web3 ecosystem leveraging blockchain technology for decentralization and transparency. It features a robust market for DeFi, NFTs, and smart contracts, offering innovative tools for developers, and investors. The ecosystem includes a unique token for transactions, a decentralized governance model, and an auction platform for exclusive items. Briky Land empowers users with complete control over their assets and interactions.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Briky Land Smart Contracts. It was conducted on commit `6353b4e17d53d37bbe12eb36a6c83518a43b37c4` from git repository link *https://github.com/brikyland/briky-land-smart-contract*.

SHA256 Sum	File
a753b96935d5a91e54d5e2b2f3a6ae64e70e28b82e7c918ba85c1782977dc752	interfaces/IAdmin.sol
d4539af7e9986795e6aae675bef1afcc54a66ac1215410ec61963c591406c546	interfaces/IAuction.sol
f158a4269f596a0bf3d6be2d4fa1e251ad2d7ec21bcc2bb950144232d2b07923	interfaces/ICollection.sol
49a8cee1dbfe36faaffa9662d54e525ba74bcd8837e19ade3bcbb0a2f19de0fa	interfaces/IFeeReceiver.sol
9910c796c1d93f2f9c0f1884285ef5bff7c5edb4b0d056422d4c11dc7c984ca7	interfaces/IGorvenorHub.sol
be27af7ea3d0ef456b3c022ee70bc28a63fcc4f5092743b68848d8ae2ea0131f	interfaces/IMarketplace.sol
bb2f65434ee98f2c535e6fe737a6c2b99853365a90b53c2839159384dd8b0bcb	interfaces/IPrimaryToken.sol
694e947b4b679356d5b5fd74704aea167ffa66cb0ac62422fa79d7b8cf0283f0	interfaces/IStakeToken.sol
f1cb808af1353493823c5ff807f161489a78400dadab9b2bd089d2e3f9e5e03c	interfaces/ITreasury.sol
81f407b1d14cab82cba0eba7932db5ad3c0da8c946d87f154a8f4bfb16b61c43	libraries/Constant.sol
e89e73b9f6879b1968af2500c58ba20d6496cda8fec0d75fbd23900976c86fb5	libraries/FixedMath.sol
0ea87ee4fbf2164d2af23371e9d0baa6dac2dea7936f10f1c429edf56c13d009	libraries/Formula.sol
814eb1dd029576871f730e7f03e362f905355290150cc08cba9458f10a09fd13	libraries/MulDiv.sol
111f723926f5327dc7e5d5ff503c4fc836953eb551c5c1a6ffab3092768ac6ee	libraries/Signature.sol
df2f4bb97aa8b85a5e66727bfcd22bbe9cd2a56111f9ddfbaadd84a2880a590b	storages/AdminStorage.sol

2ff0206f64e2b89c8b8a5e88e80547d1a076e87378ce26728fda96fdac76ee65	storages/AuctionStorage.sol
67f6c522f2750a4b0b08cd3eadff6f81fd59af71eb8c4276295d0769ace58e43	storages/CollectionStorage.sol
3db65f87cd79ae8dea7d17779e08c77d560591aacb700f81879fa82cc83d0760	storages/FeeReceiverStorage.sol
ba01c94b7381258a2cae26792bfa65b2d73dc55fd039c3fcf0d1487441ac7cf6	storages/GovernorHubStorage.sol
ee4815b3ebed222a57ef5ac5610eee89c7bb9da127ffebf74904cf7c8143f120	storages/MarketplaceStorage.sol
fd00133ba9db65bef64aafd73461b867aad7f6f79a8eda6563e88445494fa63b	storages/PrimaryTokenStorage.sol
fd627dd64fe97f0786a9afcd3779f71df5e2716e7e913aeec1f3d20e18c66f89	storages/StakeTokenStorage.sol
fab192cea4c45f72389760da8bab960a1349988509cfdd6f9659b4e5ee6e256d	storages/TreasuryStorage.sol
971c50ea6e989d2a218019a864cf2ab7d09d1243db9c88bb0c5fd086cb4db5b2	Admin.sol
97cfe89c08fe7739c1c0ff429da57006b4d9135b3e488c1c3ccc611ba1fb3e6d	Auction.sol
6ce15aa1a226a7b321b349c47841dab69f7a195d1e3df1b68e459e7514eb840c	Collection.sol
c89142b94a16961af381a6ac58ad00b401c6bd319ec5bace30fd359c4ab29813	FeeReceiver.sol
064d2b85092d2428afc779e40e9bb1d83c82aa3b999909f7a1ad5cf5104f84df	GovernorHub.sol
48be32ca11554f030b8e392b9b6fcaac105da50cc0459b8f8f30141d26f6ea14	Marketplace.sol
047476c33c36234540c7c50c887e277edc82d6bc65cb92ee684c4a6761a2a3a6	PrimaryToken.sol
a9bf2fa061a3015056cb6060374da108f7d5faedad7876541b7959732924c41e	StakeToken.sol
57383eec22087a22f71fb0469979da324e8b407a7a1cb86932d9b98a51fae6b6	Treasury.sol

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence

- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Briky Land acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Briky Land understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Briky Land agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Briky Land will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Briky Land, the final report will be considered fully accepted by the Briky Land without the signature.

2. AUDIT RESULT

2.1. Overview

The Briky Land Smart Contracts was written in `Solidity` language, with the required version is `^0.8.20`.

2.1.1. Collection

The collection is an ERC1155 contract that allows users to create and manage NFTs within the Briky Land ecosystem. It is designed to be flexible and secure, enabling users to mint and transfer NFTs with ease. A core feature of this contract is that NFTs can be used in the GovernorHub contract to vote on proposals and make decisions about the future of the ecosystem.

2.1.2. GovernorHub

The GovernorHub contract is a decentralized governance platform that allows users to vote on proposals and shape the future of the Briky Land ecosystem. It is designed to be secure and transparent, ensuring all votes are accurately counted and the results are binding. Voting power is based on the number of NFTs from the Collection contract that a user holds.

2.1.3. Marketplace

The Marketplace is where users can buy and sell NFTs created by the Collection contract within the Briky Land ecosystem.

2.1.4. PrimaryToken

PrimaryToken is the core ERC20 token of the Briky Land ecosystem. It is used for staking and transactions in the Marketplace.

2.1.5. StakingToken

StakingToken is used for staking within the Briky Land ecosystem and for earning PrimaryTokens.

2.1.6. Auction

The Auction contract is designed for auctioning PrimaryTokens within the Briky Land ecosystem. It is flexible and secure, allowing users to deposit currency and receive it after the auction ends.

2.1.7. Admin

Admin contract is used for verifying action of other contracts in the Briky Land ecosystem. It is designed to be secure and transparent, ensuring that only actions signed by the admins are executed.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Briky Land Smart Contracts. Briky Land team acknowledged these issues according to Verichains's draft report. The following table shows the vulnerabilities found during the audit:

Title	Severity	Status
Admin.sol - DoS attack by front-running call payload to other functions	CRITICAL	Fixed
Admin.sol - DoS attack by front-running call `verifyAdminSignature` function	CRITICAL	Fixed
Collection.sol - Lack of verification for caller in `updateGovernorHub` and `updateRoyaltyFeeRate` functions	CRITICAL	Fixed
GovernorHub.sol - Wrong condition in `disableProposal` function cause the function always revert	CRITICAL	Fixed
GovernorHub.sol - Setting wrong value for `proposal.endAt` in `proposeExtracting` function	CRITICAL	Fixed
GovernorHub.sol - Wrong endTime when verify several proposals in `verifyProposal` function	CRITICAL	Fixed
Collection.sol - User can withdraw Tokenization Deposit before publicSaleEnds	HIGH	Fixed
StakeToken.sol - Error when user transfers all token balance	HIGH	Fixed
GovernorHubg.sol - Cannot repropose start letting for tokenID with canceled letting	MEDIUM	Fixed

Title	Severity	Status
Collection.sol - Missing Refund to User When msg.value Exceeds Price in `depositTokenization` function	MEDIUM	Fixed
Collection.sol - Not validating _maxSellingAmount <= _totalSupply in `requestTokenization` function	MEDIUM	Fixed
GovernorHub.sol - Can't propose Cancelling Letting Action	MEDIUM	Fixed
Collection.sol - No restrict for royaltyFeeRate, commissionRate, _tokenizationFeeRate	LOW	Fixed
GovernorHub.sol - No setter of `isForRent[_tokenId]`	LOW	Fixed
Marketplace.sol - User can buy wrong expected tokenID when chain reorg	LOW	Fixed
Marketplace.sol - Lack of whitelisting for currencies in marketplace	INFORMATIVE	Fixed
Marketplace.sol - Lack of restriction on buying zero amounts	INFORMATIVE	Fixed

Table 2. Vulnerability List

2.2.1. Admin.sol - DoS attack by front-running call payload to other functions CRITICAL

Five functions (transferAdministration1, transferAdministration2, transferAdministration3, authorizeManager, deauthorizeManager) in the Admin.sol contract use the same message format to verify the payload. An attacker can monitor the mempool and front-run the payload, disrupting the expected logic. For instance, they could promote a manager to an admin or change a new admin to admin2 instead of admin3. As a result, all actions invoking `verifyAdminSignature` would be stuck.

```
function transferAdministration2(
        address _admin2,
        bytes calldata _signature1,
        bytes calldata _signature2,
        bytes calldata _signature3
    ) external {
        verifyAdminSignatures(
            abi.encodePacked(address(this), _admin2),
```

```
            _signature1,
            _signature2,
            _signature3
        );

        admin2 = _admin2;
        emit Administration2Transfer(_admin2);
    }

    function transferAdministration3(
        address _admin3,
        bytes calldata _signature1,
        bytes calldata _signature2,
        bytes calldata _signature3
    ) external {
        verifyAdminSignatures(
            abi.encodePacked(address(this), _admin3),
            _signature1,
            _signature2,
            _signature3
        );

        admin3 = _admin3;
        emit Administration3Transfer(_admin3);
    }

    function authorizeManager(
        address _account,
        bytes calldata _signature1,
        bytes calldata _signature2,
        bytes calldata _signature3
    ) external {
        verifyAdminSignatures(
            abi.encodePacked(address(this), _account),
            _signature1,
            _signature2,
            _signature3
        );

        if (isManager[_account]) revert Authorized();
        isManager[_account] = true;
        emit ManagerAuthorization(_account);
    }
```

RECOMMENDATION

The message format must contain the function name to prevent using payload for other functions.

UPDATES

- **Jul 18, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.2. Admin.sol - DoS attack by front-running call `verifyAdminSignature` function CRITICAL

Several contracts in the project use the `verifyAdminSignature` function to ensure actions are authorized by admins. However, the `verifyAdminSignature` function does not restrict the caller. An attacker can monitor the mempool for user transactions using `verifyAdminSignature` and front-run them with the same data and signature to the admin contract, causing the nonce to increase. Consequently, the user's transaction will be reverted because the nonce has already been used.

For instance, an attacker can monitor a pause transaction in the `GovernorHub` contract and front-run its payload, causing the nonce to increase and the pause transaction to be reverted.

```
//pause function in GovernorHub.sol
    function pause(
        bytes calldata _signature1,
        bytes calldata _signature2,
        bytes calldata _signature3
    ) external whenNotPaused {
        IAdmin(admin).verifyAdminSignatures(
            abi.encode(address(this)),
            _signature1,
            _signature2,
            _signature3 //@VerichainsAudit: listen mempool and front-running with the same
data and signature to admin contract to make the nonce increasing
        );
        _pause();
    }

    //verifyAdminSignatures function in Admin.sol
    function verifyAdminSignatures(
        bytes memory _message,
        bytes calldata _signature1,
        bytes calldata _signature2,
        bytes calldata _signature3
    ) public {
        uint256 currentNonce = nonce++;

        if (!Signature.verify(admin1, _message, currentNonce, _signature1)) revert
InvalidFirstSignature();
        if (!Signature.verify(admin2, _message, currentNonce, _signature2)) revert
InvalidSecondSignature();
        if (!Signature.verify(admin3, _message, currentNonce, _signature3)) revert
InvalidThirdSignature();
```

```
        emit AdminSignaturesVerification(
            _message,
            currentNonce,
            _signature1,
            _signature2,
            _signature3
        );
    }
```

RECOMMENDATION

The `verifyAdminSignature` function should restrict the caller to prevent front-running attacks.

UPDATES

- **Jul 18, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.3. Collection.sol - Lack of verification for caller in `updateGovernorHub` and `updateRoyaltyFeeRate` functions CRITICAL

The `updateGovernorHub` and `updateRoyaltyFeeRate` functions in the Collection.sol contract do not verify the caller's address. An attacker can call these functions to change the governorHub addresses and royaltyFeeRate value, disrupting the project's intended logic.

```
function updateGovernorHub(address _governorHub) external {
        require(governorHub == address(0));
        governorHub = _governorHub;
        emit GovernorHubUpdate(_governorHub);
    }

    function updateRoyaltyFeeRate(uint256 _royaltyFeeRate) external {
        if (_royaltyFeeRate > Constant.COMMON_PERCENTAGE_DENOMINATOR) revert
InvalidPercentage();
        royaltyFeeRate = _royaltyFeeRate;
        emit RoyaltyFeeRateUpdate(_royaltyFeeRate);
    }
```

UPDATES

- **Jul 18, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.4. GovernorHub.sol - Wrong condition in `disableProposal` function cause the function always revert CRITICAL

The second if statement in `disableProposal` is always true, causing the function to always revert. Consequently, the intended logic is never executed.

```
function disableProposal(uint256 _proposalId) external onlyManager {
        if (_proposalId == 0 || _proposalId > proposalNumber) revert InvalidProposalId();
        Proposal storage proposal = proposals[_proposalId];

        ProposalState state = getProposalState(_proposalId);
        if (state != ProposalState.Passed || state != ProposalState.Failed) {
//@Verichains: always true and revert with current expression
            revert InvalidDisablingProposal();
        }
        proposal.state = ProposalState.Disabled;

        emit ProposalDisablement(_proposalId);
    }
```

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.5. GovernorHub.sol - Setting wrong value for `proposal.endAt` in `proposeExtracting` function CRITICAL

There are five proposal actions in the contract that use `verifyProposal`, but `proposeExtracting` has a different format for endTime. As a result, the `verifyProposal` function will fail to set the startAt and endAt times for the proposals.

```
function proposeExtendingExpiration(
        uint256 _tokenId,
        uint40 _duration,
        uint256 _extendYears,
        bool _usePrimaryToken
    ) external payable nonReentrant returns (uint256) {
        ICollection collectionContract = ICollection(collection);
        if (!collectionContract.exists(_tokenId)) revert InvalidTokenId();
        if (collectionContract.balanceOf(msg.sender, _tokenId) == 0) revert Unauthorized();
        ...
        proposal.proposer = msg.sender;
        proposal.endAt = _duration;

    function proposeChangingUsage(
        uint256 _tokenId,
        uint40 _duration,
        uint256 _newUsageId,
        bool _usePrimaryToken
    ) external payable nonReentrant returns (uint256) {
        ICollection collectionContract = ICollection(collection);
        if (!collectionContract.exists(_tokenId)) revert InvalidTokenId();
        ...
        proposal.proposer = msg.sender;
        proposal.endAt = _duration;
```

```
    function proposeCancellingLetting(
        uint256 _tokenId,
        uint40 _duration,
        bool _usePrimaryToken
    ) external payable nonReentrant returns (uint256) {
        ICollection collectionContract = ICollection(collection);
        if (!collectionContract.exists(_tokenId)) revert InvalidTokenId();
        if (collectionContract.balanceOf(msg.sender, _tokenId) == 0) revert Unauthorized();
        ...
        proposal.proposer = msg.sender;
        proposal.endAt = _duration;

    function proposeExtracting(
        uint256 _tokenId,
        uint40 _duration,
        uint256 _value,
        address _currency,
        bool _usePrimaryToken
    ) external payable nonReentrant returns (uint256, uint256) {
        ICollection collectionContract = ICollection(collection);
        if (!collectionContract.exists(_tokenId)) revert InvalidTokenId();
        ...
        proposal.proposer = msg.sender;
        proposal.startAt = uint40(_duration);//@Verichains: have the different handling
with duration which cause conflict in verifyProposal
```

The `proposeExtraction` function should set the `proposal.endAt` to `_duration` instead of `proposal.startAt = uint40(_duration);`.

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.6. GovernorHub.sol - Wrong endTime when verify several proposals in `verifyProposal` function CRITICAL

The `verifyProposal` function incorrectly sets `proposal.endAt` to `proposal.startAt + uint40(block.timestamp);` instead of `proposal.endAt + uint40(block.timestamp);`. Most proposals do not contain duration in the `proposal.startAt` state; instead, they store the duration in `proposal.endAt`. So almost proposal will end immediately after the verification.

```
function verifyProposal(
        uint256 _proposalId,
        uint256 _budget,
        address _currency
    ) external onlyManager {
        if (_proposalId == 0 || _proposalId > proposalNumber) revert InvalidProposalId();
        Proposal storage proposal = proposals[_proposalId];
```

```
        if (proposal.state != ProposalState.Pending) revert InvalidApprovingProposal();
        proposal.state = ProposalState.Voting;
        proposal.endAt = proposal.startAt + uint40(block.timestamp); //@Verichains: should
be proposal.endAt = proposal.endAt + uint40(block.timestamp);
        proposal.startAt = uint40(block.timestamp);
        proposal.currency = _currency;
        proposal.budget = _budget;

        emit ProposalVerification(
            _proposalId,
            _budget,
            _currency
        );
    }
```

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.7. Collection.sol - User can withdraw Tokenization Deposit before publicSaleEnds HIGH

There is an issue in the `withdrawTokenizationDeposit` function where the comparison of time is incorrect, allowing users to withdraw their tokenization deposit before `publicSaleEnds`.

```
if (request.totalSupply != 0) {
            uint256 publicSaleEndsAt = request.publicSaleEndsAt;
            if (publicSaleEndsAt < block.timestamp) revert StillSelling(); //@Verichains:
always false in publicSale time, so user can withdraw Tokenization Deposit before
publicSaleEnds
            if (publicSaleEndsAt + Constant.COLLECTION_OWNERSHIP_TRANSFERRING_TIME_LIMIT <
block.timestamp
                && request.soldAmount >= request.minSellingAmount) revert
InvalidWithdrawing(); //@Verichains: have the same problem with comparing with
block.timestamp
        }
```

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.8. StakeToken.sol - Error when user transfers all token balance HIGH

There is an error in the third statement of the _transfer function. The current statement reads:`require(weight < weights[_from], "ERC20: transfer amount exceeds balance");`.

This condition is incorrect because it checks if weight is strictly less than weights[_from], which means transferring the entire balance (weight == weights[_from]) would fail the check and revert the transaction erroneously.

```
function _transfer(address _from, address _to, uint256 _amount) private whenNotPaused {
        require(_from != address(0), "ERC20: transfer from the zero address");
        require(_to != address(0), "ERC20: transfer to the zero address");

        uint256 weight = Formula.tokenToWeight(_amount, accumulatedInterestRate);
        require(weight < weights[_from], "ERC20: transfer amount exceeds balance");
//@Verichains: should be require(weight <= weights[_from], "ERC20: transfer amount exceeds
balance");
        unchecked {
            weights[_from] = weights[_from].sub(weight);
            weights[_to] = weights[_to].add(weight);
        }

        emit Transfer(_from, _to, _amount);
    }
```

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.9. GovernorHubg.sol - Cannot repropose start letting for tokenID with canceled letting MEDIUM

The `proposeStartLetting` function in the GovernorHub.sol contract allows users to propose starting a letting action for a token. However, if the letting action for the token has been canceled, the `isForRent[_tokenId]` variable will be set to true, and there is no logic to reset it to false. Consequently, the token will never be available for letting again. This restriction prevents users from re-proposing a letting action for a token that has been canceled.

```
function accomplishProposal(uint256 _proposalId, address _operator)
    external payable nonReentrant onlyManager {
            ...

            ProposalLabel label = proposal.label;
            if (label == ProposalLabel.StartLetting) {
                isForRent[proposal.tokenId] = true;
            } else if (label == ProposalLabel.CancelLetting) {
                isForRent[proposal.tokenId] = true; //@VerichainsAudit: the action set
isForRent to true, so the token will never letting anymore
            } else if (label == ProposalLabel.Extract) {
                ...
    }

    function proposeStartingLetting(
        uint256 _tokenId,
        uint40 _duration,
        bool _usePrimaryToken
    ) external payable nonReentrant returns (uint256) {
```

```
        ICollection collectionContract = ICollection(collection);
        if (!collectionContract.isAvailable(_tokenId)) revert InvalidTokenId();
        if (collectionContract.balanceOf(msg.sender, _tokenId) == 0) revert Unauthorized();
        if (isForRent[_tokenId]) revert AlreadyForRent(); //@VerichainsAudit: when
isForRent[_tokenId] true, so always revert
```

UPDATES

- **Jul 18, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.10. Collection.sol - Missing Refund to User When msg.value Exceeds Price in `depositTokenization` function MEDIUM

When the contract uses a native token for exchanges, the user must send the correct amount of the native token, which may sometimes be higher than the required price. Currently, the contract does not handle cases where the user sends more than the price, resulting in the excess amount being stuck in the contract indefinitely. It is recommended to add a check to ensure that any excess amount paid by the user is refunded.

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.11. Collection.sol - Not validating _maxSellingAmount <= _totalSupply in `requestTokenization` function MEDIUM

The `requestTokenization` function lacks a validation check to ensure that `_maxSellingAmount` is less than or equal to `_totalSupply`. If `_maxSellingAmount` exceeds `_totalSupply`, it will cause an error during the token minting process in the `confirmTokenization` step. It is recommended to add a validation check to prevent this issue.

```
function confirmTokenization(
        uint256 _requestId,
        address _commissionReceiver
    ) external nonReentrant onlyManager returns (uint256) {
        if (_requestId == 0 || _requestId > tokenizationRequestNumber) revert
InvalidRequestId();
        TokenizationRequest storage request = tokenizationRequests[_requestId];
        if (request.totalSupply == 0) revert Cancelled();
        ...
        _mint(
            requester,
            tokenId,
            request.totalSupply - soldAmount, //@Verichains: soldAmount can be larger than
soldAmount if _maxSellingAmount > _totalSupply
            ""
        );
```

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.12. GovernorHub.sol - Can't propose Cancelling Letting Action MEDIUM

The `proposeCancellingLetting` function can propose cancelling a letting action even when the token is not for rent. However, since the `isForRent[_tokenId]` variable is always false, this function will always revert with a `NotForRent()` error.

```
function proposeCancellingLetting(
        uint256 _tokenId,
        uint40 _duration,
        bool _usePrimaryToken
    ) external payable nonReentrant returns (uint256) {
        ICollection collectionContract = ICollection(collection);
        if (!collectionContract.exists(_tokenId)) revert InvalidTokenId();
        if (collectionContract.balanceOf(msg.sender, _tokenId) == 0) revert Unauthorized();
        if (!isForRent[_tokenId]) revert NotForRent(); //@Verichains: `isForRent[_tokenId]`
always false, so always revert
```

UPDATES

- **Jul 11, 2024**: This issue has been acknowledged by Briky Land team.

2.2.13. Collection.sol - No restrict for royaltyFeeRate, commissionRate, _tokenizationFeeRate LOW

Three values `royaltyFeeRate`, `commissionRate`, `_tokenizationFeeRate` are not restricted in the range of 0-100%. It is recommended to add a check to ensure that these values are within the range of 0-100%.

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.14. GovernorHub.sol - No setter of `isForRent[_tokenId]` LOW

The `isForRent[_tokenId]` variable is not set in the contract, resulting in `isForRent[_tokenId]` always being false. It is recommended to add a setter for the `isForRent[_tokenId]` variable to properly manage the rental status of tokens.

UPDATES

- **Jul 11, 2024**: This issue has been acknowledged by Briky Land team.

2.2.15. Marketplace.sol - User can buy wrong expected tokenID when chain reorg LOW

The functions `buyToken` and `buyTokenWithAmount` currently do not verify the `offerID` against the `tokenID` that the user intends to purchase. This oversight means that if a chain reorganization occurs(some block skipped and listToken tx is also skipped) before the user's transaction executes, the `offerID` could be filled with a different `tokenID`. Consequently, users may unintentionally buy the wrong expected `tokenID`. It is advisable to implement a check to ensure that users buy the intended `tokenID`.

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.16. Marketplace.sol - Lack of whitelisting for currencies in marketplace INFORMATIVE

Currently, the contract allows sellers to set any currency, which opens up the possibility for attackers to use valuable currencies to trick users into buying tokens that are not related to the project or are otherwise undesirable. It is recommended to implement a whitelist for currencies to prevent this issue.

UPDATES

- **Jul 12, 2024**: This issue has been acknowledged and fixed by Briky Land team.

2.2.17. Marketplace.sol - Lack of restriction on buying zero amounts INFORMATIVE

The contract does not currently restrict users from buying zero amounts of tokens in the `buyTokenWithAmount` function. This oversight could enable attackers to trigger events with zero amounts, which might inadvertently lead to unintended actions if the project's server relies on these events to trigger certain actions. To prevent this potential issue, it is recommended to add a check to ensure that users cannot buy tokens with zero amounts in buyTokenWithAmount.

UPDATES

- **Jul 22, 2024**: This issue has been acknowledged and fixed by Briky Land team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jul 11, 2024*	Private Report	Verichains Lab
1.1	*Jul 12, 2024*	Private Report	Verichains Lab
1.2	*Jul 18, 2024*	Public Report	Verichains Lab
1.3	*Jul 22, 2024*	Public Report	Verichains Lab
1.4	*Aug 01, 2024*	Public Report	Verichains Lab

Table 3. Report versions history