Hytopia Security Review

Pashov Audit Group

Conducted by: Hash, Said

February 17th 2024 - February 19th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **system-contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Hytopia

HYCHAIN is a blockchain (Ethereum L2 Rollup). The HYCHAIN Node Key is a non-fungible token (NFT) essential for operating a Guardian Node on the HYCHAIN network.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash **- ec45c8357aca526d4a83005f2e9b120abbba08ae**

fixes review commit hash **- 3d342858ad9022f44de2df66e64b2d1c921b2fe3**

Scope

The following smart contracts were in scope of the audit:

- `HychainNodeKeyPricing`
- `HychainNodeKeyPricingStorage`
- `HychainNodeKey`
- `HychainNodeKeyStorage`
- `HychainNodeKeyTransferManager`
- `HychainNodeKeyTransferManagerStorage`
- `HychainNodeKeyWhitelist`
- `HychainNodeKeyWhitelistStorage`

7. Executive Summary

Over the course of the security review, Hash, Said engaged with Hytopia to review Hytopia. In this period of time a total of **10** issues were uncovered.

Protocol Summary

Protocol Name	Hytopia
Repository	https://github.com/HYCHAIN/system-contracts
Date	February 17th 2024 - February 19th 2024
Protocol Type	Ethereum L2 Rollup

Findings Count

Severity	**Amount**
High	1
Medium	2
Low	7
Total Findings	**10**

Summary of Findings

ID	Title	Severity	Status
[H-01]	Mint price will be inaccurate for quantities that jumped more than one price tier	High	Resolved
[M-01]	Incorrect excessFeeRefundAddress and callValueRefundAddress are provided when bridging tokens to the L2	Medium	Resolved
[M-02]	Incorrect address provided as the receiver when bridging and minting the Node Key token on the L2	Medium	Resolved
[L-01]	Precomputed quantity and price disallow certain forms of buying	Low	Acknowledged
[L-02]	Using transfer instead of call may disallow interaction with some contracts	Low	Acknowledged
[L-03]	lack of _refBps value validation.	Low	Acknowledged
[L-04]	Reentrancy point inside mint functions could become an issue in the future	Low	Resolved
[L-05]	_refRecipient can prevent users from getting benefit from valid referral data	Low	Acknowledged
[L-06]	mintWhitelist will always revert when not providing referral data	Low	Resolved
[L-07]	Malicious user can intentionally split their mint operations to consume huge amounts of Topia tokens for bridging	Low	Acknowledged

8. Findings

8.1. High Findings

[H-01] Mint price will be inaccurate for quantities that jumped more than one price tier

Severity

Impact: High, prices can be significantly different and not considering the in-between price tier.

Likelihood: Medium, it is possible for users that buy the token in high quantity or when the current token id is near the end of the current price tier.

Description

When users mint the `HychainNodeKey`, it will eventually calculate the price that needs to be paid by the users, given the current token id and the quantity of tokens that the user wants to mint by calling `getPriceForQuantity`.

```
function getPriceForQuantity
    (uint256 _startingId, uint256 _qty) public pure returns (uint256) {
      if (_qty == 1) {
          return getCurrentPrice(_startingId);
      }
      uint256 _startingPrice = getCurrentPrice(_startingId);
      uint256 _endingPrice = getCurrentPrice(_startingId + _qty - 1);
      if (_endingPrice == _startingPrice) {
          return _startingPrice * _qty;
      }
      // find the quantity in starting price and ending price
      //(ex qty = 5, starting at id #3182 means 2 are at starting price, 3 are at endi
      uint256 _lastTierId = getLastIdForTier(_startingId);
      // @audit - this will not work properly if jump more than 1 tier
      uint256 _startingPriceQty = _lastTierId - _startingId + 1;
      return (_startingPrice * _startingPriceQty) + (_endingPrice *
        (_qty - _startingPriceQty));
  }
```

However, as can be observed, it will only consider `_startingPrice` and `_endingPrice` from the provided starting token id and quantity. If the provided quantity passes through multiple price tiers, it will only consider the first and last price tier, ignoring the information from the in-between price tiers.

PoC scenario :

- `_startingId` is 3183 (the end of price tier 1), which means the `_startingPrice` will use tier 1 pricing.
- The provided `_qty` is 3026, and it will mint tokens until the token id is 6208 (the start of price tier 3). Consequently, the `_endingPrice` will use tier 3 pricing.
- The price calculation will use the `_endingPrice` (tier 3) for tokens that have surpassed the tier 1 token id. This means tier 3 pricing will be applied to all tokens including those that should have been using tier 2 pricing, resulting in an incorrect price calculation.

Coded PoC :

```
function test_mint_three_tiers() public {
        _warpPublicMint();

        uint256 _lastIdForTier1 = _getLastIdForTier();

        vm.deal(leet, 10000 ether);

        vm.prank(leet);
        hychainNodeKey.mint{ value: 1000 ether }(address
          (this), _lastIdForTier1 - 1);

        // second tier
        uint256 _lastIdForTier2 = hychainNodeKey.getLastIdForTier
          (_lastIdForTier1 + 1);

        uint256 _qty3 = _lastIdForTier2 + 1 - (hychainNodeKey.totalSupply
          () + 1) + 1;

        // price if we mint in 1 call (one call with jump from 1 -> 3)
        uint256 price_ = hychainNodeKey.getPriceForQuantity
          (hychainNodeKey.totalSupply() + 1, _qty3);

        console.log("total price in 1 call : ");
        console.log(price_);
        console.log("quantity : ");
        console.log(_qty3);

        // now, we compare with price with multiple calls
        //(one with jump from 1 -> 2, one call with jump from 2 -> 3)

        uint256 stepQty1 = _lastIdForTier2 - (hychainNodeKey.totalSupply
          () + 1) + 1;
        uint256 stepQty2 = _qty3 - stepQty1;

        uint256 price_1 =  hychainNodeKey.getPriceForQuantity
          (hychainNodeKey.totalSupply() + 1, stepQty1);
        uint256 price_2 =  hychainNodeKey.getPriceForQuantity
          (hychainNodeKey.totalSupply() + 1 + stepQty1, stepQty2);

        uint256 totalPrice = price_1 + price_2;
        console.log("total price in 2 calls : ");
        console.log(totalPrice);
        console.log("step qty 1 : ");
        console.log(stepQty1);
        console.log("step qty 2 : ");
        console.log(stepQty2);
    }
```

Log output :

```
Logs:
  total price in 1 call :
  40015625000000000000
  quantity :
  3026

  total price in 2 calls :
  347992250000000000000
  step qty 1 :
  3025
  step qty 2 :
  1
```

Recommendations

Consider modifying the pricing to account for multiple price tier changes, or restrict the quantity to ensure that no more than one price tier change occurs.

8.2. Medium Findings

[M-01] Incorrect `excessFeeRefundAddress` and `callValueRefundAddress` are provided when bridging tokens to the L2

Severity

Impact: Medium, `address(this)` is provided for the `excessFeeRefundAddress` and `callValueRefundAddress` parameters when calling `createRetryableTicket`, which could lead to a loss of excess fee and the ability to cancel the L1 -> L2 ticket.

Likelihood: Medium, The ability to cancel a ticket might needed when there is potentially malicious behavior that needs to be prevented when `_mintAndBridge` is triggered.

Description

HYCHAIN's blockchain uses the same technology that powers Arbitrum Nova (L2). One of the capabilities that is available and used is L1 -> L2 bridging for the minted node key token. According to the docs, `excessFeeRefundAddress` is L2 address to which the excess fee is credited and `callValueRefundAddress` is address that has the capability to cancel the bridging ticket if needed.

```
function _mintAndBridge(address _to, uint256 _qty) internal {
        uint256 _startingTokenId = _nextTokenId();
        _mint(_to, _qty);
        HychainNodeKeyStorage.Layout storage $ = HychainNodeKeyStorage.layout();
        // require enough nativeToken to bridge
        if ($._topia != address(0)) {
            // TODO: figure out the exact amount
            require(IERC20($._topia).balanceOf(address
              (this)) >= $._transferCost, "Not enough $TOPIA to mint");
        }
        // approve inbox to transfer token
        IERC20($._topia).approve($._inbox, $._transferCost);
        // register ownership via retryable ticket
        uint256 ticketID = IERC20Inbox($._inbox).createRetryableTicket(
            $._l2NodeKeyAddress, // to
            0, // l2CallValue
            $._maxSubmissionCost, // maxSubmissionCost
>>>         address(this), // excessFeeRefundAddress
>>>         address(this), // callValueRefundAddress
            $._l2GasLimit, // gasLimit
            $._l2GasPrice, // maxGasPrice
            4e15, // tokenTotalFeeAmount
            abi.encodeWithSignature("mint
              (address,uint256,uint256)", msg.sender, _startingTokenId, _qty)
        );

        emit InboxTicketCreated(msg.sender, ticketID, _startingTokenId, _qty);
    }
```

However, `address(this)` is provided for those two parameters, which could become an issue when the excess fee is non-zero or when the cancel action is required to prevent unexpected behavior.

Recommendations

Consider putting a configurable L2 address for `excessFeeRefundAddress` and `callValueRefundAddress`

[M-02] Incorrect address provided as the receiver when bridging and minting the Node Key token on the L2

Severity

Impact: Medium, The user may not expect the `msg.sender` to receive the token on L2 instead of the `_to` address provided.

Likelihood: Medium, as this will happen all the time when users mint Node Key token and `_mintAndBridge` is triggered and `msg.sender` is not the same

with `_to` address provided.

Description

When users mint Node Key token, it will eventually trigger `_mintAndBridge`, which mints the token to the `_to` address provided and bridges the mint information to the L2. However, inside the calldata provided to L2, it provides `msg.sender` instead of `_to` parameter provided by users.

```
function _mintAndBridge(address _to, uint256 _qty) internal {
        uint256 _startingTokenId = _nextTokenId();
        _mint(_to, _qty);
        HychainNodeKeyStorage.Layout storage $ = HychainNodeKeyStorage.layout();
        // require enough nativeToken to bridge
        if ($._topia != address(0)) {
            // TODO: figure out the exact amount
            require(IERC20($._topia).balanceOf(address
              (this)) >= $._transferCost, "Not enough $TOPIA to mint");
        }
        // approve inbox to transfer token
        IERC20($._topia).approve($._inbox, $._transferCost);
        // register ownership via retryable ticket
        uint256 ticketID = IERC20Inbox($._inbox).createRetryableTicket(
            $._l2NodeKeyAddress, // to
            0, // l2CallValue
            $._maxSubmissionCost, // maxSubmissionCost
            address(this), // excessFeeRefundAddress
            address(this), // callValueRefundAddress
            $._l2GasLimit, // gasLimit
            $._l2GasPrice, // maxGasPrice
            4e15, // tokenTotalFeeAmount
>>>         abi.encodeWithSignature("mint
  (address,uint256,uint256)", msg.sender, _startingTokenId, _qty)
        );

        emit InboxTicketCreated(msg.sender, ticketID, _startingTokenId, _qty);
    }
```

Users might expect the `_to` parameter provided to also be the receiver in the L2. This could lead to unexpected behavior if the `msg.sender` is a contract that may not be available on L2 or cannot handle the minted token.

Recommendations

Provide `_to` instead of `msg.sender` to the calldata passed to L2.

8.3. Low Findings

[L-01] Precomputed quantity and price disallow certain forms of buying

To buy NFTs, a user is required to specify the quantity and pass the estimated cost along with the call. This disallows certain possible scenarios that users might consider when buying.

1. In the final stages of the sale, attempting to obtain as much NFT as possible can lead to reverts since the calculated quantity could overflow the cap due to other user's buying.
2. Due to possible price changes, users would not be able to maximize their buying power. If the user has 100 Eth, the user doesn't have the option to specify buying the maximum possible amount of NFTs for 100 Eth. The user has to either pass in the quantity as calculated with the current price for 100Eth in which case increased price due to other user's buying can cause the tx to revert while if the user attempts to solve this by passing in a lower number of NFT's and the price doesn't change, the user misses the remaining amount.

Adding functionality to specify the intent to buy the maximum amount of NFTs can resolve this

[L-02] Using transfer instead of call may disallow interaction with some contracts

To return overpaid amounts, the `mint` functions use the `transfer` method instead of the `call` method

```
function mint(address _to, uint256 _qty) public payable {
        if (_to == address(0)) {
            revert InvalidRecipient();
        }
        if (getPointsCostPerNodeKey() != 0) {
            revert NotPublicMintPhase();
        }

        uint256 _price = getPriceForQuantity(_nextTokenId(), _qty);

        if (msg.value < _price) {
            revert InsufficientPaymentAmount(msg.value, _price);
        } else if (msg.value > _price) {
=>          payable(msg.sender).transfer(msg.value - _price);
        }
```

Since `transfer` method only passes 2300 gas, it can cause the transaction to be reverted if the interacting contract uses more gas in its receive/fallback function.

Hence either communicate this prior or use the call method with reentrancy guards in order to mitigate this issue

[L-03] lack of `_refBps` value validation.

When users mint via `mintWithReferral` or `mintWhitelist` and provide valid referral data, it will transfer native ETH to `_refRecipient` with an amount calculated as `(_price * _refBps) / 10000`. However, there is no proper validation of `_refBps` that can be used inside the calls. Consider checking the provided `_refBps` and reverting when the value is greater than the configured cap to prevent unexpected behavior.

[L-04] Reentrancy point inside mint functions could become an issue in the future

mint functions inside `HychainNodeKey` have a potential reentrancy point because they transfer the excess native and referral payout before `_mintAndBridge` is triggered. If, in the future, `.call` is used instead of `.transfer`, malicious users can exploit this reentrancy point to buy Node Key tokens at a lower price tier than the actual price they have to pay or to mint tokens that can exceed the maximum mint limit (50000). While this is

prevented by the gas limitation when using `.transfer`, it is advisable to also implement reentrancy lock functionality.

[L-05] `_refRecipient` can prevent users from getting benefit from valid referral data

When users mint Node Key and provide valid referral data, it will eventually transfer the referral payout to the `_refRecipient` within the same operation. This could cause an issue if the `_refRecipient` is a contract intentionally preventing the transfer operations, as users cannot benefit from the valid referral data for price discounts. Consider implementing a push-over-pull design, tracking `_refRecipient`'s referral payout, and having a separate function for claiming the payout.

[L-06] `mintWhitelist` will always revert when not providing referral data

Referral data is an optional parameter inside `mintWhitelist`. When `_refCode` provided is `0x0`, it is indicated that no referral code is used. However, inside the function, it checks the signature provided by calling `_validateAndUseReferralSignature`, regardless of the `refCode` value. This will cause the call will always revert when referral data is not provided, even though the referral data is optional. Consider only checking the referral signature when `refCode` is not `0x0`.

[L-07] Malicious user can intentionally split their mint operations to consume huge amounts of Topia tokens for bridging

When the whitelist phase is over, users can `mint` any non-zero amount of node key tokens, and eventually, it will be bridged to the L2 for every `mint` operation. This allows malicious users to split their `mint` requests into the smallest quantity possible, causing a huge amount of Topia tokens to be used since the bridging token fee is on a per `mint` request basis. Consider adding a

maximum amount of Node Key tokens that can be minted per address or implementing a time delay restriction for each address between `mint` operations.