



Gacha Security Review



Contents

1. About SBSSecurity	3
2. Disclaimer	3
3. Risk classification	3
4. Executive Summary	4
5. Findings	5
Low severity	6
[L-01] purchase() is not EIP712 compliant	6
[L-02] Remove per element array hashing in GachaMallPaymentV2.sol#getBuyBackHash	7
[L-03] purchase() is missing nonce in the signature	8
[L-04] redeeming card locks the tokens used in purchase	9

1. About SBSecurity

SBSecurity is a team of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter [@Slavcheww](https://twitter.com/Slavcheww).

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

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

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.



4. Executive Summary

A time-boxed security review of the **sleekcore/gacha-audit-pokemon** repository was conducted by **SBSecurity**. The review was performed by a team of 3 security researchers, who identified **4 issues** in total.

Overview

Project	Gacha
Commit Hash	b2963b8039b4d4d903099efff60a15ae497ed263
Resolution	6b7f007f748b04100cd6e7f59bba9a136c8c7c6c
Timeline	Part 1: October 15 - October 17, 2025 Part 2: October 23, 2025

Scope

GachaMallV2.sol, GachaMallConfig.sol, GachaMallStorage.sol,
GachaMallPaymentV2.sol, GachaPvP.sol, GachaPvPConfig.sol,
GachaPvPStorage.sol, GachaPvPPayment.sol

Post Audit Condition

The codebase is well-structured and follows solid development practices. No Critical/High severity issues were found.

5. Findings

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low/Info Risk	4

ID	Title	Severity	Resolution
[L-01]	<code>purchase()</code> is not EIP712 compliant	Low	Acknowledged
[L-02]	Remove per element array hashing in <code>GachaMallPaymentV2.sol#getBuyBackHash</code>	Low	Acknowledged
[L-03]	<code>purchase()</code> is missing nonce in the signature	Low	Acknowledged
[L-04]	redeeming card locks the tokens used in purchase	Low	Acknowledged

Low severity

[L-01] `purchase()` is not EIP712 compliant

Severity: Low Risk

Description: When buying an NFT, the signature must be signed by `config.signer`, but EIP712 is not used as in `buyBack()`.

```
function purchase(
  uint32 presetId,
  address token,
  uint256 amount,
  address player,
  uint256 deadline,
  bytes calldata signature
)
  external
  nonReentrant
  returns (uint256 seqNo)
{
  Storage storage $ = _getOwnStorage();

  if (block.timestamp > deadline) revert DeadlineExceeded();
  if (amount == 0) revert InvalidAmount();

  bytes32 callDataHash = keccak256(abi.encodePacked(presetId, token, amount, player, deadline));
  -----
  address signer = callDataHash.recover(signature);
  if (signer != $.config.signer) revert Unauthorized();

  seqNo = ++$.currentSeqNo;

  IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
  _safeMint(player, seqNo);

  emit GachaMallLib.Purchased(seqNo, presetId, player, token, amount);
}
```

Recommendation: Enforce EIP712 for the `callDataHash`.

Resolution: Acknowledged

[L-02] Remove per element array hashing in `GachaMallPaymentV2.sol#getBuyBackHash`

Severity: Low Risk

Description: `BUYBACK_TYPEHASH` has an array of `uint256` and the hashing of the array by the EIP712 specification is different.

The array values are encoded as the `keccak256` hash of the concatenated `encodeData` of their contents (i.e. the encoding of `SomeType[5]` is identical to that of a struct containing five members of type `SomeType`).

Now each element of the `cardIds` array is hashed, but this is not necessary since `uint256` is an atomic type, not a Struct.

```
function getBuyBackHash(
    uint256[] calldata cardIds,
    address token,
    uint256 amount,
    uint256 deadline
)
public
view
returns (bytes32)
{
    bytes32[] memory lines = new bytes32[](cardIds.length);

    for (uint256 i = 0; i < cardIds.length; i++) {
        uint256 item = cardIds[i];
        lines[i] = keccak256(abi.encode(item)); <-----
    }

    return _hashTypedDataV4(
        keccak256(
            abi.encode(BUYBACK_TYPEHASH, keccak256(abi.encodePacked(lines)), token, amount, deadline)
        )
    );
}
```

Recommendation: Use `keccak256(abi.encodePacked(cardIds))` instead of `keccak256(abi.encodePacked(lines))`.

Resolution: Acknowledged

[L-03] `purchase()` is missing nonce in the signature

Severity: Low Risk

Description: The signature signed inside `purchase()` can be replayed until the `deadline > block.timestamp`. This is because there is no `nonce` that is automatically incremented in the contract and the user has no control over it.

```
function purchase(
    uint32 presetId,
    address token,
    uint256 amount,
    address player,
    uint256 deadline,
    bytes calldata signature
)
    external
    nonReentrant
    returns (uint256 seqNo)
{
    Storage storage $ = _getOwnStorage();

    if (block.timestamp > deadline) revert DeadlineExceeded();
    if (amount == 0) revert InvalidAmount();

    bytes32 callDataHash = keccak256(abi.encodePacked(presetId, token, amount, player, deadline));
    <-----
    address signer = callDataHash.recover(signature);
    if (signer != $.config.signer) revert Unauthorized();

    seqNo = ++$.currentSeqNo;

    IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
    _safeMint(player, seqNo);

    emit GachaMallLib.Purchased(seqNo, presetId, player, token, amount);
}
```

Recommendation: Add a `nonce` for the signing data.

Resolution: Acknowledged

[L-04] redeeming card locks the tokens used in purchase

Severity: Low Risk

Description: Tokens, used in to **purchase** will be locked once cards are being **redeemed**:

```
function redeem(uint256 cardId) public {  
    safeTransferFrom(msg.sender, _getOwnStorage().config.operator, cardId, "");  
    emit GachaMallLib.Redeem(cardId);  
}
```

The reason is that, unlike **buyBack**, here only the card is taken from the user and transferred to the **operator**. Therefore, unless the operator himself does **buyBack**, which will burn the card, tokens will be locked in the **GachaMallV2** contract.

Recommendation: Determine the desired functionality and add a feature to extract locked tokens.

Resolution: Acknowledged

Notes

Severity: Informational Risk

Description:

1. If you decide to do new deployment of **GachaMallV2** you must consider invoking the **initialize** function. In the deployment transaction. Otherwise, anyone can frontrun and steal the ownership.
2. remove **batchPurchase**, V2 does the same operations.
3. move **PurchaseParams** to **PvPState**
4. in **cancelGame**, delete the games mapping, don't default it.

```
$.games[battleId] = Game(address(0), address(0), 0, 0, 0);
```

5. To follow the CEI pattern, consider moving the winner selection before reward NFTs distribution.

```
$.winners[battleId] = winner == game.player1 ? game.player1 : game.player2;
```

Resolution: Fixed - 2, 3, 4, 5