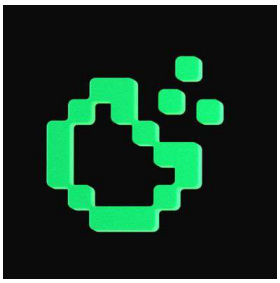

Cookie3 Community Gated Leaderboards Security Review

Duration: July 19, 2025 - July 21, 2025

July 30, 2025

Conducted by **KeySecurity**
Georgi Krastenov, Lead Security Researcher

Table of Contents

1 About KeySecurity **3**

2 About Cookie3 **3**

3 Disclaimer **3**

4 Risk classification **3**
4.1 Impact . . . 3
4.2 Likelihood . . . 3
4.3 Actions required by severity level . . . 4

5 Executive summary **5**

6 Findings **6**
6.1 Low . . . 6
6.1.1 getSubscriptions is vulnerable to DoS . . . 6
6.1.2 updateDiscount allows updates to non-existent discount IDs . . . 6
6.1.3 Use of .transfer() in _withdrawCoin may cause reverts . . . 7
6.2 Information . . . 7
6.2.1 Zero duration subscription abuse . . . 7
6.2.2 Assignment in event emission . . . 8
6.2.3 Rename _withdrawCoin to _withdrawNative for clarity . . . 8

1 About KeySecurity

KeySecurity is an innovative Web3 security company that hires top talented security researchers for your project. We have conducted over 40 security reviews for different projects which hold over $500,000,000 in TVL. For security audit inquiries, you can reach out to us on Twitter/X or Telegram @gkrastenov, or check our previous work here.

2 About Cookie3

Cookie3 utilizes off and on-chain analytics and an AI data layer to determine quality users who bring value on-chain and reward them for their contribution.

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

4 Risk classification

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

4.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	Cookie3
Repository	https://github.com/Cookie3-dev/community-gated-leaderboards
Commit hash	62e92e6be885721816f7bbca676bad933826e96e
Review Commit hash	5fd2c2950c630715ed765dd73c001d5c519e17fe
Documentation	Shared internal file
Methods	Manual review

Scope

Cookie3CommunityLeaderboardPoolsV1.sol
Withdrawable.sol
Discount.sol
Subscription.sol
Errors.sol

Timeline

July 19, 2025	Audit kick-off
July 21, 2025	Mitigation review
July 30, 2025	Final report

Issues Found

Severity	**Count**
High	0
Medium	0
Low	3
Information	3
Total	**6**

6 Findings

6.1 Low

6.1.1 getSubscriptions is vulnerable to DoS

Severity: *Low*

Context: Cookie3CommunityLeaderboardPoolsV1.sol#L179-L192

Description: The `getSubscriptions` function returns a list of all registered subscriptions. It iterates through all project IDs stored in `projectsIds` and fills the return array by casting each subscription to a DTO using `toDTO` function.

```
    /// @notice Returns the list of the subscriptions.
    function getSubscriptions() external view returns (Subscription.DTO[] memory) {
        uint256[] memory ids = projectsIds.values();

        uint256 size = ids.length;
        Subscription.DTO[] memory collection = new Subscription.DTO[](size);
        for (uint256 i = 0; i < size; i++) {
            uint256 id = ids[i];

            collection[i] = subscriptions[id].toDTO(id);
        }

        return collection;
    }
```

This implementation introduces a potential Denial of Service (DoS) vulnerability due to unbounded array growth. As the number of project IDs increases, the function may eventually exceed block gas limits, causing transactions that call it to fail. This would make the function unusable and potentially break any relying off-chain or on-chain functionality.

Recommendation: Introduce pagination by allowing the caller to specify a `start` index and a `limit` to retrieve subscriptions in batches:

```
function getSubscriptions(uint256 start, uint256 limit) external view returns (
    Subscription.DTO[] memory);
```

Resolution and Client comment: Resolved. The recommended start index and limit number were added.

6.1.2 updateDiscount allows updates to non-existent discount IDs

Severity: *Low*

Context: Cookie3CommunityLeaderboardPoolsV1.sol#L141

Description: The `updateDiscount` function allows a manager to update a discount's minimum and maximum duration, as well as its rate. While it correctly validates the discount attributes via `Discount.validate`, it does not check whether the `discountId` actually exists.

```
    function updateDiscount(uint256 discountId, Discount.Attributes memory discount)
         external onlyRole(MANAGER_ROLE) {
        if (!Discount.validate(discount)) revert Errors.UnacceptableValue();

        discounts[discountId] = discount;

        emit Discount.Updated(discountId, discount);
    }
```

This omission introduces a logic flaw: the function can silently create a “ghost” discount a discountId that exists in the mapping discounts but not in the enumerable set that tracks registered discount IDs. This can lead to inconsistencies in how discounts are retrieved or listed.

Recommendation: Before updating the discount, verify that the discountId is already registered in the enumerable set:

```
if (!discountsIds.contains(discountId)) revert Errors.DiscountNotFound();
```

Resolution and Client comment: Resolved. It was added `if (!discountsIds.contains(discountId))revert Errors.Forbidden();`.

6.1.3 Use of .transfer() in _withdrawCoin may cause reverts

Severity: *Low*

Context: Withdrawable.sol#L36

Description: Using `.transfer()` to transfer native assets is compiled to a low-level CALL opcode with a gas limitation of 2300 during compilation. In case the recipient is a smart contract with a fallback function that consumes more than 2300, the call will fail, and the entire call to `_withdrawCoin` will revert.

```
    /// @notice Withdraws the native coin to the recipient.
    /// @param to Address of the recipient.
    function _withdrawCoin(address payable to) internal virtual {
        if (to == address(0)) revert WithdrawToZeroAddress();

        to.transfer(address(this).balance);
    }
```

Recommendation: Consider replacing `.transfer()` with `.call()` instead.

Resolution and Client comment: Resolved. `Address.sendValue()` is used now.

6.2 Information

6.2.1 Zero duration subscription abuse

Severity: *Information*

Context: Cookie3CommunityLeaderboardPoolsV1.sol#L87

Description: The `subscribe()` function allows registration of a project with a duration of 0 and an invalid (non-existent) discount ID.

Recommendation: Add validation inside `subscribe()` function:

```
if(!discountIds.contains(discountId)) revert Errors.DiscountNotFound();
```

Resolution and Client comment: Resolved.

6.2.2 Assignment in event emission

Severity: *Information*

Context: Cookie3CommunityLeaderboardPoolsV1.sol#L115

Description: Avoid assigning new values directly within an event emission. It makes the code harder to read.

```
emit FeeUpdated(fee.rate = rate, fee.collector = collector);
return discount.calculate(duration * fee.rate, duration);
```

Recommendation: Separate the state mutation from the event emission for clarity and safety:

```
fee.rate = rate;
fee.collector = collector;

emit FeeUpdated(rate, collector);
```

This improves readability, avoids unintended side effects and aligns with best practices.

Resolution and Client comment: Resolved. The followed recommendation is applied.

6.2.3 Rename _withdrawCoin to _withdrawNative for clarity

Severity: *Information*

Context: Withdrawable.sol#L33

Description: Rename the `_withdrawCoin` function to `_withdrawNative` for improved readability and clarity.

The term “coin” is ambiguous and may be confused with ERC20 tokens or project-specific terminology. Using “native” more clearly communicates that the function deals with the blockchain’s native asset (e.g., ETH, AVAX, MATIC).

```
    /// @notice Withdraws the native coin to the recipient.
    /// @param to Address of the recipient.
    function _withdrawCoin(address payable to) internal virtual {
        if (to == address(0)) revert WithdrawToZeroAddress();

        to.transfer(address(this).balance);
    }
```

Recommendation: Rename `_withdrawCoin` to `_withdrawNative`.

Resolution and Client comment: Resolved. The function is renamed.