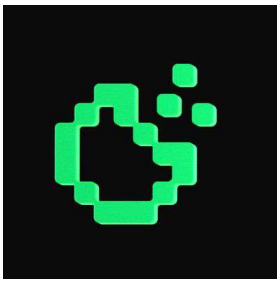

Cookie3 Security Review

Duration: May 28, 2024 - May 31, 2024

June 3, 2024

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

Table of Contents

1 About KeySecurity

KeySecurity is an innovative Web3 security company that hires top talented security researchers for your project. We have conducted over 25 security reviews for different projects which hold over $300,000,000 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/airdrop-contracts
Commit hash	177cfaddd92bf065456c75953fea7583040b819f
Review Commit hash	ac12cb4eae094bde43e51963780223b10a12b9ba
Documentation	N/A
Methods	Manual review

Scope

AirdropClaim.sol
Swap.sol
Quoter.sol

Timeline

May 28, 2024	Audit kick-off
May 31, 2024	Preliminary report
June 3, 2024	Mitigation review

Issues Found

Severity	**Count**
High	0
Medium	1
Low	2
Information	6
Code Improvement	0
Total	**10**

6 Findings

6.1 Medium

6.1.1 Possible second preimage attack

Severity: *Medium*

Context: AirdropClaim.sol#L124

Description: The `msg.sender` and `amount` take part in the leaf encoding. These two variables are concatenated and used as leaves in this Merkle tree. `msg.sender` is of type `address`, which is equal to 20 bytes and `amount` is of type `uint256`, which is 32 bytes. Therefore, their total combined size is 52 bytes. The problem arises when `abi.encode` is used. `abi.encode` adds padding when necessary; in our case, it will add 12 bytes before the `msg.sender`. So, the total number of bytes will be 64, not 52.

```
abi.encodePacked: 20 + 32 = 52 bytes 0
    x5b38da6a701c568545dcfcb03fcb875f56beddc4000000000..0002de09ddb3
abi.encode: 32 + 32 = 64 bytes       0
    x000000000000000000000000005b38da6a701c568545dcfcb03fcb875f56beddc4000...00002
    de09ddb3
```

Merkle trees that use 64-byte leaves are vulnerable to the second preimage attack. Since internal nodes are 32 byte long (as these are the output of a keccak hash), then these can be combined to prove the presence of certain values that aren't actually leaves in the tree.

Recommendation: Make the following changes:

```
- return keccak256(abi.encode(_account, _amount));
+ return keccak256(abi.encodePacked(_account, _amount));
```

Resolution and Client comment: Resolved.

6.2 Low

6.2.1 The slippage amount is calculated on-chain

Severity: *Low*

Context: Swap.sol#L137-L140

Description: Currently, `minAmountOut` is calculated on-chain using the TWAP and after that, a 2% fee is deducted. That's the slippage amount which the user will use. If somehow the current price from the TWAP is manipulated by a malicious user, the `minAmountOut` will be affected.

The most popular solution to calculate the slippage amount is to do it off-chain and pass it as an input parameter directly into the `swap` function. Of course, considering that we are using the TWAP and that the contracts are deployed on the Base, it is very unlikely for the price to be manipulated. However, for added safety, I would recommend calculating the slippage off-chain.

Recommendation: he slippage amount should be calculated off-chain. When the user selects the amount in your front-end, deduct a 2% fee. Then, call the quote function from the Quoter contract and use the result as an input parameter to the swap function of the Swapper contract.

Resolution and Client comment: Resolved.

6.2.2 The cardinality should be increased after initializing the WETH <> Cookie pool

Severity: *Information*

Context: Global

Description: Currently, a cardinality of 256 means that there will be 256 slots for observation. So, if there is a price movement in the pool every block, this equals 256 * 13 = 3328 observations, approximately every 55 minutes. Therefore, our TWAP interval of 30 minutes will be satisfied.

Unfortunately, the WETH <> Cookie pool will be created on Base and 256 can be problematic. On Base, there is a block every 2 seconds, so 256 * 2 = 512 observations, approximately every 8 minutes. This will affect the precision of the timing and obtaining the correct price

Recommendation: Increase the cardinality to [1000-1800].

Resolution and Client comment: Resolved.

6.3 Information

6.3.1 Redundant error

Severity: *Information*

Context: irdropClaim.sol#L30

Description: In the AirdropClaim contract, the error TransferFailed is never used.

Recommendation: Remove the redundant error.

Resolution and Client comment: Resolved.

6.3.2 Use msg.sender instead of owner()

Severity: *Information*

Context: AirdropClaim.sol#L158

Description: When a function is only accessible by the owner due to the presence of the onlyOwner modifier, there is no need to call an internal function to obtain the owner's address. In this case, msg.sender is equal to owner().

Recommendation: Make the following changes:

```
- return i_token.transfer(owner(), i_token.balanceOf(address(this)));
+ return i_token.transfer(msg.sender, i_token.balanceOf(address(this)));
```

Resolution and Client comment: Resolved.

6.3.3 The return param from transfer/transferFrom is not handled

Severity: *Information*

Context: Global

Description: In several places in the codebase, the transfer and transferFrom functions are used. Both functions return a boolean to indicate if the transfer is successful. Currently, the return result is not handled.

Recommendation: Check the return result; if it is false, revert the whole transaction.

Resolution and Client comment: Acknowledged. The contracts will work only with WETH and USDC.

6.3.4 Unnecessary checks in the Quoter constructor

Severity: *Information*

Context: Quoter.sol#L48-L55

Description: The addresses of the `factory`, `WETH`, `USDC`, and `Cookie` contracts are checked to see if they are empty (equal to `address(0)`). These checks are unnecessary because the Quoter contract is created in the constructor of the Swapper contract, where these checks are already done.

Recommendation: Remove the unnecessary checks.

Resolution and Client comment: Resolved.

6.3.5 Use directly encoded leave instead of hashed proofs

Severity: *Information*

Context: AirdropClaim.sol#L95-L104

Description: The mapping `proofUsed` uses hashed proofs as key values. Instead of calling the `hashBytes32Array` function to encode and hash the given proofs, you can use the encoded leaf `encodeLeaf(msg.sender, _amount)` as a unique value.

The similar approach is used in Blur and LooksRare airdrop contracts.

Recommendation: Use directly encoded leave instead of hashed proofs.

Resolution and Client comment: Resolved.

6.3.6 The Swap contract unnecessarily inherits the Ownable contract

Severity: *Information*

Context: Swap.sol#L15

Description: The Swap contract unnecessarily inherits the Ownable contract. The functionality of the Ownable contract is never used within the Swap contract and the owner does not have privileged rights to change anything within it. Inheriting the Ownable contract lacks purpose because its functionality is never utilized.

The similar approach is used in Blur and LooksRare airdrop contracts.

Recommendation: Inheriting of the Ownable contract should be removed.

Resolution and Client comment: Resolved.