CANTINA

Basenames

Security Review

Cantina Managed review by:

Optimum, Lead Security Researcher
High Byte, Security Researcher

June 11, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Base is a secure, low-cost, builder-friendly Ethereum L2 built to bring the next billion users onchain.

From May 28th to May 30th the Cantina team conducted a review of basenames on commit hash 2418f4a6. The team identified a total of **9** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	1	0	1
Gas Optimizations	4	3	1
Informational	3	1	2
Total	**9**	**5**	**4**

3 Findings

3.1 Medium Risk

3.1.1 Users who used their one-time discount in the legacy registrar will be able to use it again after the upgrade

Severity: Medium Risk

Context: UpgradeableRegistrarController.sol#L258

Description: The `validDiscount` modifier is used by the `discountedRegister()` function to enforce the following constraints:

- The provided discount must exist and be active.
- Discounts can only be used once per address.
- The caller must be eligible for the discount.

The implementation:

```
modifier validDiscount(bytes32 discountKey, bytes calldata validationData) {
    URCStorage storage $ = _getURCStorage();
    if ($.discountedRegistrants[msg.sender]) revert AlreadyRegisteredWithDiscount(msg.sender);
    DiscountDetails memory details = $.discounts[discountKey];

    if (!details.active) revert InactiveDiscount(discountKey);

    IDiscountValidator validator = IDiscountValidator(details.discountValidator);
    if (!validator.isValidDiscountRegistration(msg.sender, validationData)) {
        revert InvalidDiscount(discountKey, validationData);
    }
    _;
}
```

The `discountedRegistrants` mapping tracks addresses that have already used their discount. However, when the `UpgradeableRegistrarController` is deployed, replacing the current `RegistrarController` (stored in `legacyRegistrarController`), users who have previously claimed a one-time discount in the legacy contract will be able to claim it again in the new contract.

Recommendation: Add a check in `validDiscount` to verify whether `msg.sender` has already used their discount in the legacy contract. This ensures that users cannot claim their discount more than once, even across contract upgrades.

Coinbase: Fixed in commit e6bd1419.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Avoid griefing gas sponsors

Severity: Low Risk

Context: UpgradeableRegistrarController.sol#L620

Description/Recommendation: Makes sense to limit gas in UpgradeableRegistrarController.sol#L620 to avoid griefing gas sponsors.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 `legacyRegistrarController.hasRegisteredWithDiscount()` is executed multiple times redundantly

Severity: Gas Optimization

Context: UpgradeableRegistrarController.sol#L389

Description: The `hasRegisteredWithDiscount()` function exists in both the current `UpgradeableRegistrarController` and the previous `RegistrarController` implementations. In the current version, the function iterates over the provided `addresses` array to check whether any address has used its discount by looking it up in the local `discountedRegistrants` mapping. Additionally, for each iteration, the function redundantly calls the legacy contract's `hasRegisteredWithDiscount()` function, passing the entire `addresses` array every time. This results in an inefficient pattern where the same external call to the legacy contract is made multiple times (once per address in the array - for the entire array), instead of just once. Ideally, the legacy contract check should be performed a single time outside the loop, and its result reused within the current iteration.

```
function hasRegisteredWithDiscount(address[] memory addresses) external view returns (bool) {
    URCStorage storage $ = _getURCStorage();
    for (uint256 i; i < addresses.length; i++) {
        if (
            $.discountedRegistrants[addresses[i]]
                || RegistrarController($.legacyRegistrarController).hasRegisteredWithDiscount(addresses)
        ) {
            return true;
        }
    }
    return false;
}
```

Recommendation: Consider calling `hasRegisteredWithDiscount()` only once outside of the loop instead.

Coinbase: Fixed in commit e6bd1419.

Cantina Managed: Fix verified.

3.3.2 `base.isAvailable()` is called twice during the execution of `register()/discountedRegister()`

Severity: Gas Optimization

Context: UpgradeableRegistrarController.sol#L455

Description: The `available()` function is called during the registration process to verify that a name is valid and available for purchase in the `BaseRegistrar`. However, the call to `base.isAvailable()` within `available()` is redundant during the `register()` and `discountedRegister()` flows because this availability check is already performed later in the process by `BaseRegistrar.registerWithRecord()`.

Recommendation: Consider refactoring by implementing an internal function named `_available()` that only verifies the validity of the name. Both `available()` and `validRegistration` can call this internal function, replacing the current direct call to `available()`. This change allows `available()` to serve solely as an external view function, which can then be marked as `external`, improving clarity and gas efficiency.

Coinbase: Fixed in commit e6bd1419.

Cantina Managed: Fix verified.

3.3.3 `RegistrarController`, `BaseRegistrar`, `L2Resolver` are imported but only used for their interface

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: The contracts `RegistrarController`, `BaseRegistrar`, and `L2Resolver` are imported into `UpgradeableRegistrarController`, but only their interfaces are utilized. Importing the full contract implementations unnecessarily increases deployment gas costs.

Recommendation: Replace these full contract imports with their corresponding interface imports. This optimization reduces deployment size and gas consumption.

Coinbase: Fixed in commit e6bd1419.

Cantina Managed: Fix verified.

3.3.4 Minor gas optimization - `shl` instead of `mul` + `exp`

Severity: Gas Optimization

Context: ReverseRegistrarV2.sol#L305

Description/Recommendation: Minor gas optimization: can just shift by 96 `shl(96, a)`.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Potential reentrancy in future versions of the contract

Severity: Informational

Context: UpgradeableRegistrarController.sol#L683

Description: The `withdrawETH()` function allows anyone to withdraw the ETH balance accumulated in the contract to the designated `paymentReceiver`:

```
function withdrawETH() public {
  (bool sent,) = payable(_getURCStorage().paymentReceiver).call{value: (address(this).balance)}("");
  if (!sent) revert TransferFailed();
}
```

Although this function currently does not pose a reentrancy risk—mainly because `_refundExcessEth()` prevents such attacks—the contract's upgradeable nature means this vulnerability could be introduced in future versions. An attacker exploiting a reentrancy vulnerability could hijack the call flow by repeatedly invoking `withdrawETH()` and potentially drain user funds.

Recommendation: Add non-reentrant guards to all state-changing functions, including `withdrawETH()`, to prevent reentrancy attacks. For gas efficiency, consider using OpenZeppelin's `ReentrancyGuardTransient.sol`.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 Consider using `Ownable2Step`

Severity: Informational

Context: ReverseRegistrarV2.sol#L21

Description/Recommendation: Consider using `Ownable2Step`.

Coinbase: Fixed in commit e6bd1419.

Cantina Managed: Fix verified.

3.4.3 `discountedRegisterPrice` doesn't check discount key is valid

Severity: Informational

Context: UpgradeableRegistrarController.sol#L498-L500

Description: `discountedRegisterPrice` doesn't check discount key is valid. Mixing discount keys due to error may mislead to sending invalid register transactions.

Recommendation: Perhaps consider moving the `validDiscount` modifier from `discountedRegister` to here.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.