CANTINA

Base usernames

Security Review

Cantina Managed review by:

Saw-mon and natalie, Lead Security Researcher

Rustyrabbit, Security Researcher

Akshay srivastav, Associate Security Researcher

July 14, 2024

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 and low-cost Ethereum layer-2 solution built to scale the userbase on-chain.

From Jun 19th to Jun 25th the Cantina team conducted a review of base-usernames on commit hash 082ed1b9. The team identified a total of **22** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 5
- Gas Optimizations: 7
- Informational: 9

***Note**: During the security review, it was assumed that Base L2 will always have a private mempool. In absence of which the usernames protocol can face front-running related issues.*

3 Findings

3.1 Medium Risk

3.1.1 Same address can use multiple discounts through reentrancy

Severity: Medium Risk

Context: RegistrarController.sol#L231, RegistrarController.sol#L425-L436, RegistrarController.sol#L523

Description: The `discountRegistrants` mapping is used to keep track of which addresses have already used a discount and block them from registering with another or the same discount again.

The mapping is checked in the `validDiscount` modifier and updated after `_registration` function. The user can specify a custom `resolver` address in the `request` on which `multicallWithNodeCheck` is called in the `_registration` function. This can be used to reenter the contract and use the same or a different discount before the `discountRegistrants`is updated.

Note that as the `validDiscounts` checks the discount eligbility against the `msg.sender` the specified `resolver` must be the address that is eligible for the discount. This could be the case for smart contract wallets that can provide the needed `multicallWithNodeCheck` function to perform this exploit.

Recommendation: Move the update to the `discountRegistrants` state variable before doing the call to the resolver. This could be just before the `_register`statement or even in the `validDiscount` modifier.

Coinbase: Addressed in PR 46.

Cantina Managed: Fixed.

3.2 Low Risk

3.2.1 Price discovery not active on first registration

Severity: Low Risk

Context: RegistrarController.sol#L352

Description: After names expire there is a price discovery mechanism where the price start high and decays over time. However as the expiration of names that have never been registered is `0` the price discovery will result in a premium of `0`. This will attract a lot for squatters at the launch as it will be relatively cheap for them to buy a lot of names.

Recommendation: Consider only allowing registering of new names after a start auction transaction which sets the expiration time to the current block timestamp. This will ensure a price discovery is possible after a certain name becomes interesting enough for squatters.

Alternatively initiate the price discovery after deployment.

Coinbase: Acknowledged, won't do. This is something the product team has been considering and we've ultimately decided not to pursue for a few reasons:

- It kills day-1 momentum: names would be prohibitively expensive on the launch day.
- It adds complicated branches to the pricing mechanism given discount vs. no discount and letter length.
- Revenue generation is not a primary driver for this product.

Cantina Managed: Acknowledged.

3.2.2 L1Resolver should catch `OffchainLookup` errors on subcalls

Severity: Low Risk

Context: L1Resolver.sol#L224

Description: As specified in EIP-3668 CCIP aware contracts like the `L1Resolver` should not bubble up any `OffchainLookup` errors from calls to other contracts unless the `sender` corresponds to the callee address.This ensures any nested CCIP read isn't wrongly interpreted as the response to the top level query.

The EIP only discusses recursive calls as `OffchainLookup` from other contracts would not have the correct `sender`, but this assumes that the callee:

1. Is not malicious and ...
2. Does catch any `OffchainLookup` errors in subcalls.

As the intended `RootResolver` is the ENS public resolver we can safely rule out `1.` but as it sits behind an upgradable proxy we cannot guarantee `2.`.

To illustrate the maximum impact, assume that the ENS public resolver in a future code base does forward calls to an untrusted (malicious) contract that reverts with a `OffchainLookup` where the `sender` is set to the `L1Resolver`. The Dapp will see this as a legitimate CCIP read revert and issue the call to the gateway with the data provided in the `OffchainLookup`. The `callData` and `extraData` however are completely under control of the malicious contract and as such could be a query to another name (ie. `attacker.base.eth`). As the response is fetched from the official Base gateway it is properly signed by the trusted signer and will be validated by the `L1Resolver`. The result is that a query for the name `victim.base.eth` is wrongly resolved to the address of `attacker.base.eth`.

Recommendation: Catch any `OffchainLookup` errors when doing calls to other contracts and revert with a custom error in order not to continue the CCIP read worklfow.

Note: The severity for this issue is low as we can assume that ENS as authors of EIP-3668 would correctly implement the safety guards of the EIP, but this is not guaranteed.

Coinbase: Acknowledged, won't do. This one is a bit tricky. Ultimately, we need to trust that ENS isn't going to deploy a malicious Public Resolver behind the `rootResolver`.

If that did happen, it likely means that the ENS DAO has experienced a serious exploit and this would be on of many repercussions of such an attack. If this did happen in the wild, we could always update the `rootResolver` to some other contract or even an empty address.

Cantina Managed: Acknowledged.

3.2.3 Missing appending contract's address while generating signature hash

Severity: Low Risk

Context: SybilResistanceVerifier.sol#L29

Description: As per EIP-191 the following bytes after `1900` should be the `<intended validator address>` (i.e, `address(this)`). This is done so that the signatures cannot be replayed across different contracts.

The `SybilResistanceVerifier._makeSignatureHash` currently appends the `signer` address after `1900`.

Recommendation: Consider appending `address(this)` after `1900`:

```
keccak256(abi.encodePacked(hex"1900", address(this), /*...*/));
```

Coinbase: Fixed in PR 47.

Cantina Managed: Verified. The validator address is now included in signature hash generation.

3.2.4 Ensure changing ownership is also reflected on the `ReverseRegistrar`'s node owner

Severity: Low Risk

Context: RegistrarController.sol#L28

Description: `RegistrarController` inherits from `Ownable` where it allows for changing ownership. When the ownership transfers to a new owner, the node associated with `x.addr.reverse` where `x` is the address of this `RegistrarController` keeps/retains its old owner.

Recommendation: Make sure when the ownership of this contract changes the owner of the corresponding node at `x.addr.reverse` is also updated. Just like the `constructor` call:

```
reverseRegistrar.claim(newOwner);
```

Coinbase: Acknowledged. We will enforce this operationally and ensure that ownership transfers also accommodate the transfer of any relevant `owner`-owned nodes.

Cantina Managed: Acknowledged.

3.2.5 `setReverseRegistrar` does not call the `claim` endpoint to update the node owner on the new reverse registrar

Severity: Low Risk

Context: RegistrarController.sol#L306-L309

Description: When `reverseRegistrar` is updated the associated node corresponding to the address of this `RegistrarController` is not claimed.

Recommendation: Just like the initialisation in the `constructor`, make sure to call `reverse_.claim(owner_)` to claim the ownership of the node associated with `x.addr.reverse` where `x` is the address of the `RegistrarController`.

Coinbase: Acknowledged, won't fix in contract. We will assume operational responsibility to ensure that new upon deploying a new `ReverseRegistrar` we will maintain the reverse record for the `RegistrarController`

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 Parameters can be defined as `immutable`

Severity: Gas Optimization

Context: AttestationValidator.sol#L25, BaseRegistrar.sol#L32, ERC1155DiscountValidator.sol#L15, ERC1155DiscountValidator.sol#L18

Description: The variables in this context have only been set in the constructor and there is no other way to modify them.

Recommendation: It would be best to define them as `immutable`:

contract	parameter
`AttestationValidator`	`schemaID`
`ERC1155DiscountValidator`	`token`
`ERC1155DiscountValidator`	`tokenId`
`BaseRegistrar`	`registry`

Coinbase: Addressed in PR 48.

Cantina Managed: Fixed.

3.3.2 `_isContract` can be optimised

Severity: Gas Optimization

Context: ReverseRegistrar.sol#L202-L208

Description: One can avoid extra operations by using a named return variable.

Recommendation: Use a named return variable and let the compiler `solc` take care of the conversion to `bool`:

```
function _isContract(address addr) internal view returns (bool result) {
    assembly {
        result := extcodesize(addr)
    }
}
```

`forge snapshot --diff .gas-review`:

```
test_allowsOwnerOfContract_toSetName_forOwnedContractAddress() (gas: -6 (-0.002%))
test_allowsOwnerOfContract_toClaimForAddr_forOwnedContractAddress() (gas: -6 (-0.002%))
test_reverts_ifNotAuthorized() (gas: -6 (-0.025%))
test_constructor_setsTheRootNodeOwner() (gas: -27 (-0.211%))
test_constructor() (gas: -8061 (-29.447%))
test_constructor() (gas: 1149708 (4199.847%))
Overall gas change: 1141602 (4.934%)
```

Coinbase: Addressed in PR 53. The following is used instead:

```
addr.code.length == 0
```

Cantina Managed: Gas diff needs to be perfomed to ensure the alt optimisation used.

3.3.3 `_isApprovedOrOwner` can be optimised

Severity: Gas Optimization

Context: BaseRegistrar.sol#L236-L240, BaseRegistrar.sol#L377-L380, BaseRegistrar.sol#L377-L381

Description: Reuse the implementation from `solady` for better optimisation and also refactor the non-expiry check into a `modifier`.

Recommendation: Apply the following patch:

```
diff --git a/src/L2/BaseRegistrar.sol b/src/L2/BaseRegistrar.sol
index 823beef..e170339 100644
--- a/src/L2/BaseRegistrar.sol
+++ b/src/L2/BaseRegistrar.sol
@@ -137,6 +137,14 @@ contract BaseRegistrar is ERC721, Ownable {
         _;
     }

+    /// @notice Decorator for determining if a name is not expired.
+    ///
+    /// @param id The id being checked for non-expiry.
+    modifier onlyNonExpired(uint256 id) {
+        if (nameExpires[id] <= block.timestamp) revert Expired(id);
+        _;
+    }
+
     /*´:°•.°+.*•´.*:.°*.•´.°:°•.°•.*•´.*:.°*.•´.°:°•.°+.*•´.*:*/
     /*                        IMPLEMENTATION                      */
     /*.•°:°.´+.*°.:*.´•*.+°.•°:´*.´•*.•°.•°:°.´:•°.*°.:*.´+°.•*/
@@ -234,8 +242,7 @@ contract BaseRegistrar is ERC721, Ownable {
     /// @param tokenId The id of the name to query the owner of.
     ///
     /// @return address The address currently marked as the owner of the given token ID.
-    function ownerOf(uint256 tokenId) public view override returns (address) {
-        if (nameExpires[tokenId] <= block.timestamp) revert Expired(tokenId);
+    function ownerOf(uint256 tokenId) public view override onlyNonExpired(tokenId) returns (address) {
         return super.ownerOf(tokenId);
     }

@@ -374,8 +381,13 @@ contract BaseRegistrar is ERC721, Ownable {
     ///
     /// @return `true` if msg.sender is approved for the given token ID, is an operator of the owner,
     ///         or is the owner of the token, else `false`.
-    function _isApprovedOrOwner(address spender, uint256 tokenId) internal view override returns (bool) {
-        address owner = ownerOf(tokenId);
-        return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(owner, spender));
+    function _isApprovedOrOwner(address spender, uint256 tokenId)
+        internal
+        view
+        override
+        onlyNonExpired(tokenId)
+        returns (bool)
+    {
+        return super._isApprovedOrOwner(spender, tokenId);
     }
 }
```

forge snapshot --diff .gas-review:

```
test_reverts_whenNameHasExpired() (gas: 6 (0.004%))
test_successfullyRegisters() (gas: 8 (0.004%))
test_successfullyRegisters() (gas: 8 (0.005%))
test_renewsOwnershipSuccessfully_whenInGracePeriod() (gas: 8 (0.005%))
test_renewsOwnershipSuccessfully_whenNotExpired() (gas: 8 (0.005%))
test_returnsTheOwner(address) (gas: 8 (0.005%))
test_successfullyRegistersOnly() (gas: 8 (0.006%))
test_successfullyRegisters_afterExpiry(address) (gas: 16 (0.007%))
test_successfullyRegisters_afterExpiry(address) (gas: 16 (0.008%))
test_successfullyRegisters_afterExpiry(address) (gas: 16 (0.010%))
test_reverts_whenCalledAfterExpiry() (gas: -18 (-0.011%))
test_reclaimsOwnership_whenCalledByOwner_beforeExpiry() (gas: -56 (-0.034%))
test_constructor_setsTheRootNodeOwner() (gas: -27 (-0.211%))
test_reverts_whenCalledByNonOwnerOrApprovedOperator(address) (gas: -348 (-0.211%))
test_reclaimsOwnership_whenCalledByOperator_beforeExpiry(address) (gas: 1843 (0.961%))
test_reclaimsOwnership_whenCalledByOperator_approvedForAll(address) (gas: -2468 (-1.261%))
test_constructor() (gas: -8061 (-29.447%))
test_constructor() (gas: 1149708 (4199.847%))
Overall gas change: 1140675 (4.930%)
```

Coinbase: Addressed in PR 52.

Cantina Managed: Fixed.

3.3.4 `nameExpires[id]` can be cached in `BaseRegistrar.renew`

Severity: Gas Optimization

Context: BaseRegistrar.sol#L262-L268

Description: `nameExpires[id]` has been read multiple times from storage. To save gas it can be cached.

Recommendation: Cache `nameExpires[id]` in a stack variable:

```
function renew(uint256 id, uint256 duration) external live onlyController returns (uint256) {
    uint256 nameExpiry = nameExpires[id];
    if (nameExpiry + GRACE_PERIOD < block.timestamp) revert NotRegisteredOrInGrace(id);

    nameExpiry += duration;
    nameExpires[id] = nameExpiry;
    emit NameRenewed(id, nameExpiry);
    return nameExpiry;
}
```

forge snapshot --diff .gas-review:

```
test_reverts_whenNotInGracePeriod() (gas: -6 (-0.004%))
test_reverts_whenNotRegistered() (gas: -6 (-0.010%))
test_constructor_setsTheRootNodeOwner() (gas: -27 (-0.211%))
test_renewsOwnershipSuccessfully_whenInGracePeriod() (gas: -456 (-0.276%))
test_renewsOwnershipSuccessfully_whenNotExpired() (gas: -456 (-0.276%))
test_constructor() (gas: -8061 (-29.447%))
test_constructor() (gas: 1149708 (4199.847%))
Overall gas change: 1140696 (4.930%)
```

Coinbase: Addressed in PR 51

Cantina Managed: Fixed.

3.3.5 Use `FixedPointMathLib.mulWad` in `EDAPrice.currentPrice`

Severity: Gas Optimization

Context: EDAPrice.sol#L36

Description: In this context we have:

```
uint256 price = (startPrice * uint256(multiplier)) / FixedPointMathLib.WAD;
```

But one can use the already imported `FixedPointMathLib` from `solady`.

Recommendation: The context can be replaced by the more optimised version from `solady` library:

```
uint256 price = FixedPointMathLib.mulWad(startPrice, uint256(multiplier));
```

`forge snapshot --diff .gas-review`:

```
test_constructor_setsTheRootNodeOwner() (gas: -27 (-0.211%))
test_decayedPremium_decreasingPrices(uint256) (gas: -55 (-0.444%))
test_decayedPremium_zeroElapsed() (gas: -55 (-0.445%))
test_decayedPremium_threePeriods() (gas: -55 (-0.536%))
test_decayedPremium_halfPeriod() (gas: -55 (-0.537%))
test_decayedPremium_alwaysDecreasing(uint256,uint256) (gas: -110 (-0.830%))
test_decayedPremium_boundaryValues(uint256) (gas: -165 (-1.057%))
test_constructor() (gas: -8061 (-29.447%))
test_constructor() (gas: 1149708 (4199.847%))
Overall gas change: 1141125 (4.932%)
```

Coinbase: Addressed in PR 59.

Cantina Managed: Fixed.

3.3.6 In `SignatureVerifier.verify` revert early when the signature is expired

Severity: Gas Optimization

Context: SignatureVerifier.sol#L31-L32

Description: In `SignatureVerifier.verify` revert early when the signature is expired.

Recommendation: Swap the lines in this context:

```
if (expires < block.timestamp) revert SignatureExpired();
address signer = ECDSA.recover(makeSignatureHash(address(this), expires, request, result), sig);
```

`forge snapshot --diff .gas-review`:

```
test_returnsResultsWithValidSignature(string) (gas: -3 (-0.006%))
test_revertsWhenTheSignerIsInvalid(string) (gas: -3 (-0.006%))
test_constructor() (gas: -200 (-0.017%))
test_constructor() (gas: -200 (-0.017%))
test_constructor() (gas: -200 (-0.017%))
test_constructor() (gas: -200 (-0.017%))
test_constructor() (gas: -200 (-0.017%))
test_constructor() (gas: -200 (-0.017%))
test_constructor_setsTheRootNodeOwner() (gas: -27 (-0.211%))
test_revertsWhenTheSignatureIsExpired(string) (gas: -4029 (-8.703%))
test_constructor() (gas: -8061 (-29.447%))
test_constructor() (gas: 1149508 (4199.116%))
Overall gas change: 1136185 (4.911%)
```

Coinbase: Addressed in PR 50.

Cantina Managed: Fixed.

3.3.7 `hexAddress` can be optimised

Severity: Gas Optimization

Context: ReverseRegistrar.sol#L164-L181, Sha3.sol#L4

Description: `hexAddress` can be optimised by replacing:

- `div(addr, 0x10)` with `shr(4, addr)`
- `gt(i, 0)` with `i`

Also underscores can be added to `ALPHABET` to make it more readable.

Note: The current implementation copies the same implementation from `ens-contracts`.

Recommendation: Apply the following patch to save around `40` gas:

```
diff --git a/src/lib/Sha3.sol b/src/lib/Sha3.sol
index 12cec00..5877a36 100644
--- a/src/lib/Sha3.sol
+++ b/src/lib/Sha3.sol
@@ -3,7 +3,7 @@ pragma solidity ^0.8.23;

 library Sha3 {
     // Hex encoding of "0123456789abcdef"
-    bytes32 constant ALPHABET = 0x3031323334353637383961626364656600000000000000000000000000000000;
+    bytes32 constant ALPHABET =
↪   0x30_31_32_33_34_35_36_37_38_39_61_62_63_64_65_66_00000000000000000000000000000000;

     /**
      * @dev An optimised function to compute the sha3 of the lower-case
@@ -14,13 +14,13 @@ library Sha3 {
      */
     function hexAddress(address addr) internal pure returns (bytes32 ret) {
         assembly {
-            for { let i := 40 } gt(i, 0) {} {
+            for { let i := 40 } i {} {
                 i := sub(i, 1)
                 mstore8(i, byte(and(addr, 0xf), ALPHABET))
-                addr := div(addr, 0x10)
+                addr := shr(4, addr)
                 i := sub(i, 1)
                 mstore8(i, byte(and(addr, 0xf), ALPHABET))
-                addr := div(addr, 0x10)
+                addr := shr(4, addr)
             }

             ret := keccak256(0, 40)
```

`forge snapshot --diff .gas-review`

```
test_allowsOwnerOfContract_toSetName_forOwnedContractAddress() (gas: -40 (-0.012%))
test_allowsOwnerOfContract_toClaimForAddr_forOwnedContractAddress() (gas: -40 (-0.014%))
test_allowsOperator_toSetName_forUserAddress() (gas: -40 (-0.023%))
test_allowsUser_toSetName_forUserAddress() (gas: -40 (-0.027%))
test_setsName() (gas: -40 (-0.027%))
test_allowsUser_toClaim() (gas: -40 (-0.036%))
test_allowsOperator_toClaimForAddr_forUserAddress() (gas: -40 (-0.036%))
test_allowsUser_toClaimForAddr_forUserAddress() (gas: -40 (-0.048%))
test_allowsUser_toClaimWithResolver() (gas: -40 (-0.048%))
test_constructor_setsTheRootNodeOwner() (gas: -27 (-0.211%))
test_returnsExpectedNode(address) (gas: -40 (-0.273%))
test_constructor() (gas: -8061 (-29.447%))
test_constructor() (gas: 1149708 (4199.847%))
Overall gas change: 1141220 (4.932%)
```

Coinbase: Addressed in PR 57.

Cantina Managed: Fixed.

3.4 Informational

3.4.1 Integration tests are missing

Severity: Informational

Context: Global scope

Description: Currently integration tests are missing and only some unit tests are presents

Recommendation: It would be best to develop a comprehensive cross-chain integration testing suite. A simple base test suite setup would look like this:

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.23;

import {Test, console} from "forge-std/Test.sol";

import {L1Resolver} from "src/L1/L1Resolver.sol";

import {L2Resolver} from "src/L2/L2Resolver.sol";
import {BaseRegistrar} from "src/L2/BaseRegistrar.sol";
import {ExponentialPremiumPriceOracle} from "src/L2/ExponentialPremiumPriceOracle.sol";
import {RegistrarController} from "src/L2/RegistrarController.sol";
import {Registry} from "src/L2/Registry.sol";
import {ReverseRegistrar} from "src/L2/ReverseRegistrar.sol";

import {AttestationValidator} from "src/L2/discounts/AttestationValidator.sol";
import {CBIdDiscountValidator} from "src/L2/discounts/CBIdDiscountValidator.sol";
import {ERC1155DiscountValidator} from "src/L2/discounts/ERC1155DiscountValidator.sol";

import {IBaseRegistrar} from "src/L2/interface/IBaseRegistrar.sol";
import {IDiscountValidator} from "src/L2/interface/IDiscountValidator.sol";
import {IPriceOracle} from "src/L2/interface/IPriceOracle.sol";
import {IReverseRegistrar} from "src/L2/interface/IReverseRegistrar.sol";

import {
    ETH_NODE,
    BASE_ETH_NODE,
    REVERSE_NODE,
    ADDR_REVERSE_NODE,
    GRACE_PERIOD,
    BASE_ETH_NAME
} from "src/util/Constants.sol";

contract IntegrationTest is Test {
    address owner;
    address signer;
    address alice;

    L1Resolver l1Resolver;

    Registry registry;
    BaseRegistrar baseRegistrar;
    RegistrarController registrarController;
    L2Resolver defaultL2Resolver;
    ReverseRegistrar reverseRegistrar;
    ExponentialPremiumPriceOracle exponentialPremiumPriceOracle;

    AttestationValidator attestationValidator;
    CBIdDiscountValidator cBIdDiscountValidator;
    ERC1155DiscountValidator eRC1155DiscountValidator;

    bytes32 constant ROOT_NODE = bytes32(0);
    bytes32 constant ETH_LABEL = 0x4f5b812789fc606be1b3b16908db13fc7a9adf7ca72641f84d75b47069d3d7f0;
    bytes32 constant BASE_LABEL = 0xf1f3eb40f5bc1ad1344716ced8b8a0431d840b5783aea1fd01786bc26f35ac0f;
    bytes32 constant REVERSE_LABEL = 0xdec08c9dbbdd0890e300eb5062089b2d4b1c40e3673bbccb5423f7b37dcf9a9c;
    bytes32 constant ADDR_LABEL = 0xe5e14487b78f85faa6e1808e89246cf57dd34831548ff2e6097380d98db2504a;


    function setUp() public {
        owner = makeAddr("owner");
        signer = makeAddr("signer");
        alice = makeAddr("alice");

        registry = new Registry(owner);
        reverseRegistrar = new ReverseRegistrar(registry, owner);

        uint256[] memory rentPrices = new uint256[](6);
        rentPrices[0] = 1e18;
        rentPrices[1] = 2e18;
        rentPrices[2] = 3e18;
        rentPrices[3] = 4e18;
        rentPrices[4] = 5e18;
        rentPrices[5] = 6e18;

        exponentialPremiumPriceOracle = new ExponentialPremiumPriceOracle(rentPrices, 1e18, 21);
        baseRegistrar = new BaseRegistrar(registry, owner, BASE_ETH_NODE);

        {
            // setting up the nodes
            vm.startPrank(owner);
            registry.setSubnodeOwner(ROOT_NODE, ETH_LABEL, owner);
            registry.setSubnodeOwner(ETH_NODE, BASE_LABEL, address(baseRegistrar));

            registry.setSubnodeOwner(ROOT_NODE, REVERSE_LABEL, owner);
            registry.setSubnodeOwner(REVERSE_NODE, ADDR_LABEL, address(reverseRegistrar));
            vm.stopPrank();
        }

        registrarController = new RegistrarController(
            baseRegistrar,
            exponentialPremiumPriceOracle,
            IReverseRegistrar(address(reverseRegistrar)),
            owner,
            BASE_ETH_NODE,
            "base.eth"
        );

        vm.prank(owner);
        baseRegistrar.addController(address(registrarController));

        defaultL2Resolver = new L2Resolver(
            registry,
            address(registrarController),
            address(reverseRegistrar),
            owner
        );

        vm.prank(owner);
        reverseRegistrar.setDefaultResolver(address(defaultL2Resolver));

        vm.prank(owner);
        reverseRegistrar.setName("rootOwner");

        vm.warp(GRACE_PERIOD * 10);
    }

    function test_integration_setup() public {}

    function test_integration_register() public {
        vm.stopPrank();
        vm.startPrank(alice);


        string memory name = "alice";
        uint256 duration = 90 days;

        uint256 registerPrice = registrarController.registerPrice(name, duration);
        vm.deal(alice, registerPrice);

        RegistrarController.RegisterRequest memory request = RegistrarController.RegisterRequest({
            name: name,
            owner: alice,
            duration: duration,
            resolver: address(defaultL2Resolver),
            data: new bytes[](0),
            reverseRecord: true
        });

        registrarController.register{value: registerPrice}(request);
    }
```

```
}
```

Coinbase: Addressed in PR 56.

Cantina Managed: Fixed.

3.4.2 `RegistrarController` needs to be approved before it can set reverse records

Severity: Informational

Context: RegistrarController.sol#L532, ReverseRegistrar.sol#L71

Description: For `RegistrarController` to be able to call `reverseRegistrar` using the current implementations, the `msg.sender` must have approved `RegistrarController` in `Registry` so that the following would return true:

```
// msg.sender to `RegistrarController`
registry.isApprovedForAll(msg.sender, RegistrarController)
```

Recommendation: To avoid having all the users to set the above permission/approval, one can do either of the following:

- Have the `RegistrarController` to be automatically approved for all users in `Registry` or...
- Have the `RegistrarController` to be an authorised entity for all users in `ReverseRegistrar`.

The second option might be preferred since it limits how much privilege the `RegistrarController` would have in the whole ecosystem.

Coinbase: Addressed in PR 43.

Cantina Managed: Fixed.

3.4.3 Protocol is reliant on the blockchain's private mempool design

Severity: Informational

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

Description: The `usernames` protocol does not have any front-running protocol inbuilt in its codebase. This is likely due to the fact that the protocol in intended to be deployed on Base L2 which currently has a private mempool (upcoming transactions are only revealed to trusted parties).

In case the Base L2 opts to have a public mempool in the future then the `usernames` protocol may face front-running related issues.

Coinbase: Acknowledged with comment *"For UX reasons, we eliminated this commit/reveal pattern which saves users a transaction. Our mempool is currently private and is expected to remain so for the foreseeable future. If at some point we did decide to add a public mempool, we could always deploy a new controller which does implement a commit/reveal pattern."*

Cantina Managed: Verified.

3.4.4 Ownership and resolution stay active after expiration

Severity: Informational

Context: Registry.sol#L178-L180

Description: Resolution of the ENS protocol starts with finding the resolver for a name in the registry. If the name ownership has expired `resolver` will still return the resolver address set by the old owner. Although the public ENS resolver acts in the same way they have been faced with the problem of owners not being aware their names expiring until someone else registers their name after the grace period has passed. This could be mitigated by not resolving the name immediately after expiration so that owners at least have an indication their name has expired and have the full grace period to renew their new if they choose to.

The same applies for registry changes where the ownership is only checked in the `Registry` without regard for the expiration. The previous owner can still make changes to the registry up until a new owner has registered the name and reset the ownership in the `Registry`.

Recommendation: In the `resolver`, `owner` functions and the `authorized`modifier make a call to the `RegistrarController` to check the expiration of the name and revert when the name has expired.

Coinbase: Acknowledged, won't fix (yet).

This is a known weakness with the ENS architecture. They've proposed a significant architectural overhaul which they've dubbed ENS v2. What we are deploying is well-aligned with ENS's existing v1 architecture and we plan to migrate alongside them when ENS v2 is production-ready.

Cantina Managed: Acknowledged.

3.4.5 `CHAIN_ID` should be included while creating signature hashes

Severity: Informational

Context: SignatureVerifier.sol#L19, SybilResistanceVerifier.sol#L29

Description: In case the protocol contracts get deployed on different EVM chains on same address or if the chain faces hard-fork, then a signer's signature created for one contract one one chain can be replayed on the other chain as well. This may result in unintended signatures getting successfully validated.

Recommendation: Consider including `CHAIN_ID` in signature hash generation.

```
function makeSignatureHash(/*...*/)
    internal
    pure
    returns (bytes32)
{
    return keccak256(abi.encodePacked(/*...*/, block.chainid, /*...*/));
}
```

EIP-712 can also be leveraged for data hashing and signing.

Coinbase: Acknowledged. Justifications for the two instances below:

- `SignatureVerifier`: The paired offchain signing service for `SignatureVerifier` is used in other places and so changing this is out of scope for this project.
- `SybilResistanceVerifier`: We will make `SybilResistanceVerifier` EIP-191 compliant to avoid cross-contract replayability. This version of the service and the contract will only be used on Base so cross-chain replayability isn't a concern. We can revisit whether we want to use EIP-712 if/when we productize a Sybil Resistance Service and expect that it will be used across multiple chains/products.

Cantina Managed: Acknowledged.

3.4.6 The holders of ERC1155 token can claim discount repeatedly

Severity: Informational

Context: ERC1155DiscountValidator.sol#L37-L39

Description: The protocol intends to allow a name buyer to claim discount only once. However when the discount is decided by `ERC1155DiscountValidator`, a token holder can rotate his wallets by transferring the token to different accounts and hence can claim discount multiple times.

Recommendation: It should be made sure that the ERC1155 token is a non-transferable token.

Coinbase: Acknowledged by enhancing documentation in PR 55. The expected non-transferable behaviour from the ERC1155 token has now been documented in `ERC1155DiscountValidator`.

Cantina Managed: Acknowledged.

3.4.7 Provide more info in the NatSpec as why only the `price.base` is used in `renew`

Severity: Informational

Context: RegistrarController.sol#L454-L456

Description: This assumes that the `premium` is `0` which relies on two facts:

1. Currently you can only `renew` a token on `BaseRegistrar` before the grace period ends.
2. The current implementation of the `IPriceOracle` uses the elapsed time after the grace period ends and thus it returns `0` for `premium` before that.

Recommendation: The above should be documented more in the NatSpec. Currently it is only mentioned that:

```
///     The price for renewal never incorporates pricing `premium`. Use the `base` price returned by the
↪  `rentPrice`
///     tuple to determine the price for calling this method.
```

Coinbase: Addressed in PR 54. I think we want to categorically ensure that a renewer never pays a premium. It seems more desirable to strictly only charge the `.base` price.

Cantina Managed: Fixed.

3.4.8 Better value needs to be chosen with correct units for `totalDays` in `ExponentialPremiumOracleBase.t.sol`

Severity: Informational

Context: ExponentialPremiumOracleBase.t.sol#L19, ExponentialPremiumPriceOracle.sol#L17

Description: We have:

$$p_s \max(0, 0.5^{\Delta t/D_1} - 0.5^{D_2})$$

parameter	description
p_s	`startPremium`
Δt	`elapsed`
D_1	`1 days`
D_2	`totalDays`

where $p_s \cdot 0.5^{D_2}$ is the `endValue`. Note that how `endValue` calculated in the `ExponentialPremiumPriceOracle.constructor`:

```
endValue = startPremium >> totalDays;
```

`totalDays` is supposed to half `startPremium` for every day in includes and thus its unit needs to be in days and not seconds.

Recommendation: Based on above perhaps in `test/ExponentialPremiumPriceOracle/ExponentialPremiumOracleBa`
one meant to use:

```
uint256 totalDays = 365;
```

or even a smaller more realistic value, since for any number in `uint256` like x, $x \gg 365$ would be `0`.

For a comparison `lib/ens-contracts/test/ethregistrar/TestExponentialPremiumPriceOracle.js` uses `21` for this value:

```
const START_PRICE = 100000000
const DAY = 86400
const LAST_DAY = 21
const LAST_VALUE = START_PRICE * 0.5 ** LAST_DAY
function exponentialReduceFloatingPoint(startPrice, days) {
  const premium = startPrice * 0.5 ** days
  if (premium >= LAST_VALUE) {
    return premium - LAST_VALUE
  }
  return 0
}
```

Coinbase: Addressed in PR 58.

Cantina Managed: Fixed.

3.4.9 Typos, Comments, NatSpec, Unused Imports, Unused Variables,

Severity: Informational

Context: BaseRegistrar.sol#L369, Constants.sol#L15, Constants.sol#L4, Constants.sol#L8, DeployRegistrarController.s.sol#L16, DeployTestnetDiscountValidator.s.sol#L7, EDAPrice.sol#L4, ExponentialPremiumPriceOracle.sol#L13, ExponentialPremiumPriceOracle.sol#L8

Description/Recommendation:

- Constants.sol#L4: it should be `The node hash of "eth"`. The `.` before `eth` is extra.
- Constants.sol#L8: it should be `The node hash of "reverse"`. The `.` before `reverse` is extra.

  A simple Python program to check the correctness of node hashes is shown below:

  ```
  from web3 import Web3
  from hexbytes import HexBytes

  ROOT_NODE = HexBytes(b'\0' * 32)

  def sha3(s):
      return Web3.keccak(s)

  def namehash(name: str) -> bytes:
      if name == '':
          return ROOT_NODE
      else:
          label, _, remainder = name.partition('.')
          return sha3(namehash(remainder) + sha3(label.encode('utf-8')))

  if __name__ == '__main__':
      s: str = input()
      print(namehash(s).hex())
  ```

- Constants.sol#L15: It would be best to represent `BASE_ETH_NAME` as `hex"04_62617365_03_657468_00"` with the underscores as the `dnsName` encoding follows the following pattern:

  ```
                w0    .              w1   .  ... .              wN     ->
  len(w0) | hex(w0) | len(w1) | hex(w1) | ... | len(wN) | hex(wN) | 00
  ```

  and `dnsEncodeName` only works for words of the form `w0 . w1`.

- EDAPrice.sol#L4, ExponentialPremiumPriceOracle.sol#L8: The following imports should be removed:

  ```
  import {Test, console} from "forge-std/Test.sol";
  ```

- ExponentialPremiumPriceOracle.sol#L13: `PRECISION` is unused.
- BaseRegistrar.sol#L369: misleading comment about taking grace period into consideration should be removed.
- DeployRegistrarController.s.sol#L16: typo, it should be `DeployRegistrarController`.
- DeployTestnetDiscountValidator.s.sol#L7: typo, it should be `DeployTestnetDiscountValidator`.

Coinbase: All suggestions are implemented (see PR 49).

Cantina Managed: Fixed.