PUBLIC

# Code Assessment

## of the NameWrapper Smart Contracts

September 17, 2021

Produced for


by

# Contents

# 1 Executive Summary

Dear ENS team,

First and foremost we would like to thank you for giving us the opportunity to assess the current state of your NameWrapper system. This document outlines the findings, limitations, and methodology of our assessment.

We hope that this assessment provided valuable findings. All identified findings have been resolved. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

## 1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 2 |
| • Code Corrected | 1 |
| • Specification Changed | 1 |
| Low-Severity Findings | 4 |
| • Code Corrected | 3 |
| • Specification Changed | 1 |

# 2 Assessment Overview

In this section we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

## 2.1 Scope

The assessment was performed on the source code files inside the NameWrapper repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | June 01 2021 | 429d8c4b86f93c10827fcef2dd16a744db4f3fb6 | Initial Version |
| 2 | June 21 2021 | c624466bb8566c705bc76724e3267d09f52f0e49 | Second Version |
| 3 | June 30 2021 | 0ec2823a4b54a79ba0e9e736c8b92d451d5b1bc6 | Third Version |
| 4 | August 11 2021 | 2aad2c92150b7ed56e3247f2218b7a6babb78b3e | Final Version |

For the solidity smart contracts, the compiler version `0.8.4` was chosen.

### 2.1.1 Excluded from scope

The existing contracts of ENS, including but not limited to the registry and the registrar, are out of the scope of this engagement.

### 2.1.2 Excluded from report

Independently, a vulnerability was discovered by samczsun which is hence not part of this report. For more information please visit: https://samczsun.com/the-dangers-of-surprising-code/

## 2.2 System Overview

This system overview describes the *last* received version (Version 3) of the contracts as defined in the Assessment Overview. Furthermore, in the findings section we have added a version icon to each of the findings to increase the readability of the report.

The provided contracts implement a wrapper which allows for ENS names to be wrapped as `ERC1155` tokens. Moreover, it introduces a set of fuses for each domain name which facilitate permission control. The exposed interface allows for the following actions:

`wrap`:

These functions wrap a non-`.eth` second-level domain or any other subdomain. During wrapping, the `NameWrapper` directly interacts with the ENS Registry and not with a registrar. The ENS Registry owner entry for the domain is set to be the `NameWrapper` contract. Then, the initial state of the fuses is set and an `ERC1155` token is emitted.

`unwrap`:

The `wrap` functionality is being reverted meaning that the `ERC1155` token is burned and that the `owner` entry inside the ENS Registry is reset.

`wrapETH2LD`:

This function wraps a second-level `.eth` domain into an `ERC1155` token. These domains are currently represented by an `ERC721` token. The `ERC721` token is transferred and the ENS Registry entry is changed so that the `NameWrapper` contract becomes the owner. The value of the fuses can then freely be set. The case of `.eth` domains is handled differently due to the fact that the domain is wrapped by a registrar.

The `NameWrapper` also implements the `onERC721Received` hook. This means that an `.eth` domain, represented as the `ERC721` token, can be sent to the `NameWrapper` contract to trigger the wrapping. Appropriate metadata needs to be supplied for the data parameter of the safeTransfer.

`unwrapETH2LD`:

At the unwrapping the inverse procedure takes place. Hence the `ERC721` token is released while the `ERC1155` token is burned. Additionally, the ENS Registry entry is updated accordingly.

The `NameWrapper` exposes wrapped actions for the ENS. These include `setTTL`, `setResolver`, `setRecord`, `setSubnodeOwner`, and `setSubnodeRecord`. The aforementioned actions can be executed only if the appropriate fuses are not burned.

## 2.2.1 Fuses

The permissions for each domain is handled by a set of fuses. Fuses are represented by a `uint96` where each bit represents a different permission. Owners of a domains can burn fuses (by calling `burnFuses`) to define permissions. If any of the fuses are burned then `CANNOT_UNWRAP` must also be burned and `CANNOT_REPLACE_SUBDOMAIN` must be burned on the parent node. The available fuses to burn from the `NameWrapper` interface are the following:

- `CANNOT_UNWRAP = 1`: If burned, the name cannot be unwrapped. This fuse must be burned if any other fuse has been burned.
- `CANNOT_BURN_FUSES = 2`: If burned, no further fuses can be burned.
- `CANNOT_TRANSFER = 4`: If burned, the name cannot be transferred. It leads to not being able to unwrap.
- `CANNOT_SET_RESOLVER = 8`: If burned, the resolver cannot be changed. Calls to `setResolver` and `setRecord` will fail.
- `CANNOT_SET_TTL = 16`: If burned, the TTL cannot be changed. Calls to `setTTL` and `setRecord` will fail.
- `CANNOT_CREATE_SUBDOMAIN = 32`: If burned, no new subdomains can be created under the current node. Calls to `setSubnodeOwner`, `setSubnodeRecord`, `setSubnodeOwnerAndWrap` and `setSubnodeRecordAndWrap` will fail if they reference a name that does not already exist.
- `CANNOT_REPLACE_SUBDOMAIN = 64`: If burned, existing subdomains cannot be replaced. Calls to `setSubnodeOwner`, `setSubnodeRecord`, `setSubnodeOwnerAndWrap` and `setSubnodeRecordAndWrap` will fail if they reference a name that already exists. Please note that a previously (before the last expiry) wrapped instance of a subdomain counts as existing and cannot be replaced.

### 2.2.1.1 Checking Fuses

When querying a fuse value, the domain hierarchy also needs to be considered. This is due to the fact that domains might be re-wrapped after expiry. For more information see our Resolved Findings regarding earlier versions below.

Hence, given a (sub)domain the function `getFuses` returns the raw fuse values along with a vulnerability type. A domain or a subdomain can be "vulnerable" under certain conditions. For a vulnerable domain the returned values cannot be trusted as they could be manipulated through re-wrapping. In case that `getFuses` declares that a (sub)domain is safe, however, it is sufficient to query the fuse values from this domain without the need for further hierarchical checks. Please note that the safety of a domain is only given until it or any parent domain expires next. Hence, it needs to be rechecked in due time.

### 2.2.2 Controller Functions

The `NameWrapper` contract has certain privileged functions that can only be executed by addresses with the `Controller` role:

- In order to streamline the registration process for wrapped domains, the `NameWrapper` has the capability to directly register and wrap an `.eth` domain through the `registerAndWrapETH2LD` function. The successful execution of this function requires the correct configuration of the Registrar contract.
- The `NameWrapper` contract can trigger the renewal of domains. This also requires the `NameWrapper` contract to have the necessary permissions inside the Registrar contract.

### 2.2.3 Trust Model

In the current scope the existing ENS contracts, including Registry and Registrar, are considered to be trusted and to work correctly.

The `NameWrapper` has a privileged owner role, however, the owner can only do two things. First, the owner can do set the metadata URI as this might need adjustment in the future. Second, the owner can set addresses that should receive the `Controller` role. All addresses that have the `Controller` role can call the Controller Functions described above.

In summary:

- Controllers can register and renew `.eth` domains without need for payment. Note that they can also renew non-wrapped domains.
- The owner controls the controllers and thereby has all their powers. Additionally, the owner can set the metadata service.

# 3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

# 4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

| Likelihood | Impact | | |
|---|---|---|---|
| | High | Medium | Low |
| High | Critical | High | Medium |
| Medium | High | Medium | Low |
| Low | Medium | Low | Low |

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

# 5 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 0 |

# 6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 2 |

- Expired Domains Look Valid for the Subdomains Code Corrected
- Old State From Expired Domains Can Block Legitimate Actions Specification Changed

| | |
|---|---|
| Low-Severity Findings | 4 |

- Variable node Assigned but Never Used in wrapETH2LD Code Corrected
- Incorrect Specification for Unwrapping Functions Specification Changed
- Repetitive Code Code Corrected
- Specification Unclear for setSubnode* Functions Code Corrected

## 6.1 Expired Domains Look Valid for the Subdomains

Correctness Medium Version 1 Code Corrected

The desired invariant that fuses can be inspected individually and care only needs to be taken when domains expire, can be broken. This is because an expired domain can be rewrapped with new fuses and wrapped subdomains are not aware of the expiry of higher-level domains. Please consider the following scenario:

1. User `U` controls `example.eth`, wraps it and burns the `CANNOT_REPLACE_SUBDOMAIN` fuse.
2. The domain expires and user `V` takes control of it and makes sure it will not expire any time soon.
3. User `V` assigns control to `W` over `sub.example.org`.
4. User `W` wraps `sub.example.org`:
    1. `W` also decides to burn the `CANNOT_UNWRAP` fuse.
    2. During the execution the parent node (`example.eth`) is checked where the `CANNOT_REPLACE_SUBDOMAIN` fuse has been burnt.
    3. Hence, the wrapping succeeds.
5. Now third parties check the wrapped state of `sub.example.org`: According to the invariant it cannot be unwrapped as it will not expire any time soon and as the `CANNOT_UNWRAP` fuse has been burnt.
6. User `V` can freely reassign `sub.example.org` (independently of the `NameWrapper`). Hence, the permission system has been bypassed as a non-wrapped and a wrapped version exists for `sub.example.org`.

Hence, fuses are indifferent to the expiry of domains and they enforce the corresponding permissions only for never-expired domains.

---

**Code corrected:**

The code was rewritten so that it checks the hierarchy of a name for safety.

## 6.2 Old State From Expired Domains Can Block Legitimate Actions

Correctness Medium Version 1 Specification Changed

In case a domain is wrapped, then expires and is later controlled by another user and wrapped again, the following problem can arise, which blocks the new legitimate owner from performing an action they would have been allowed to.

Please consider the following sequence:

1. User `U` controls `example.eth` and wraps it.
2. User `U` creates a subnode `sub.example.org` and sets themselves as owner.
3. The domain expires and user `V` takes control of it.
4. User `V` wraps `example.eth` again and burns the `CANNOT_REPLACE_SUBDOMAIN` fuse.
5. Now, user `V` tries to create `sub.example.org`:
    1. The function `canCallSetSubnodeOwner` is evaluated, it should return `true` as `V` has the permission to create new subdomains.
    2. The `owner` of `sub.example.org` is queried and it returns `U`.
    3. As the `owner` is non-zero, the `CANNOT_REPLACE_SUBDOMAIN` fuse is checked.
    4. Finally, `canCallSetSubnodeOwner` returns `false` and hence the legitimate creation of the subnode fails.

---

**Specification corrected:**

The specification has been made more explicit so that it covers the case above.

## 6.3 Variable `node` Assigned but Never Used in `wrapETH2LD`

Design Low Version 2 Code Corrected

In the `wrapETH2LD` functions, the new node is calculated using `_makeNode(ETH_NODE, labelhash)`. However, this value is never used.

---

**Code corrected**:

The redundant variable was removed.

## 6.4 Incorrect Specification for Unwrapping Functions

Correctness Low Version 1 Specification Changed

The README says about unwrapping:

> Wrapped names can be unwrapped by calling either *unwrapETH2LD(label, newRegistrant, newController)* or *unwrap(parentNode, label, newController)* as appropriate. *label* and *parentNode* have meanings as described under "Wrapping a name"

Furthermore, the docstring says:

- @param label label as a string of the .eth domain to wrap e.g. vitalik.xyz would be 'vitalik'

However, the implementation works differently. Instead of passing a label, a labelhash should be passed to the unwrapping functions, as seen for `unwrapETH2LD` below:

```
function unwrapETH2LD(
    bytes32 label,
    address newRegistrant,
    address newController
) public override onlyTokenOwner(_makeNode(ETH_NODE, label)) {
    _unwrap(_makeNode(ETH_NODE, label), newController);
```

---

**Specification changed:**

The documentation has been updated.

## 6.5 Repetitive Code

Design Low Version 1 Code Corrected

There are multiple instances of repetitive code that could be avoided. These instances include:

- Within the function `wrapETH2LD`, two calls are made to `registrar.ownerOf(tokenId)`.
- Within the functions `unwrapETH2LD`, `unwrap`, and `burnFuses`, two calls are made to `_makeNode(parentNode, labelhash)`.
- Within the function `burnFuses`, `getData` is called multiple times in different spots.

The cost impact of these repetitions has been lowered by the recently introduced EIP-2929, however gas optimizations remain possible.

---

**Code corrected:**

The superfluous call to `registrar.ownerOf(tokenId)` has been removed as well as the duplicate calle to `getData` in `burnFuses`.

## 6.6 Specification Unclear for `setSubnode*` Functions

Correctness | Low | Version 1 | Code Corrected

The docstring for the `setSubnodeRecord` function says:

- @notice Sets records for the subdomain in the ENS Registry
- @param node namehash of the name

However, the `node` parameter should contain the namehash for the parent node. This is not entirely clear from the description. Especially, in comparison with the `setSubnodeRecordAndWrap` function, where the docstring says:

- @notice Sets the subdomain owner in the registry with records and then wraps the subdomain
- @param parentNode parent namehash of the subdomain

A consistent naming of `node` versus `parentNode` for these very similar functions would be beneficial to avoid confusion. This also extends to the `setSubnodeOwner` and `setSubnodeOwnerAndWrap` functions. Furthermore, the `label` parameter is missing from the `setSubnodeRecord` description.

---

**Code corrected:**

The parameter names were changed to reflect their status.

# 7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

## 7.1 Dirty Bits in Return Value of `getData`

**Note** **Version 1**

The `getData` function is one of the most important functions of the `NameWrapper` as it retrieves information about the different nodes.

```
function getData(uint256 tokenId)
    public
    view
    returns (address owner, uint96 fuses)
{
    uint256 t = _tokens[tokenId];
    owner = address(uint160(t));
    fuses = uint96(t >> 160);
}
```

Functions calling `getData` need to be aware that the `owner` return value will contain "dirty bits". This is dangerous if `assembly` is being used, because `assembly` will access the raw data. Writing normal solidity code should be fine.

Hence, we recommend to avoid `assembly` in connection with `getData`. We have attached an example file for this behaviour.

## 7.2 Note to Integrators: `onERC1155Received` Hook

**Note** **Version 1**

This note is meant for any developers wanting to build upon the `NameWrapper`. Similarly to `ERC223`, `ERC721`, `ERC777`, and others the implementation of `ERC1155` invokes the `onERC1155Received` hook at the end of `safeTransferFrom`. Developers building services which interact with the `NameWrapper` should be aware of that and implement the hook, as these hooks have historically led to reentrancy attacks.

## 7.3 Restrictions on Custom Permissions

**Note** **Version 1**

The `README` documents that additional fuses might be designated to additional permissions. While this generally can be implemented with the current contract, not all types of permissions will be feasible this way.

Any permissions, requiring checks "up the chain" of custody would not work without modifications to the contract or without breaking the invariant that fuses can be inspected individually. As a somewhat

contrived example, a permission enforcing that TTL values of subnodes must be strictly larger than TTL values of parent nodes, currently could not be enforced.

## 7.4 The Transitive Permission Structure

Note Version 1

Users should be aware of the transitive permission structure of the system. This permission structure involves the `NameWrapper`, `Registry` and `Registrar`. Given a typical user setup, setting an operator `O` using `NameWrapper.setApprovalForAll` does not only pass control over all wrapped domains but `O` also controls all non-wrapped domains. Furthermore, domains that are acquired in the future, can be controlled by `O`.

In short becoming an operator for a particular account on the `NameWrapper` is more powerful than becoming an operator for the same account on the `Registrar` or the `Registry`. Hence, operator permissions should be given out with great care.