CONSENSYS
Diligence

ENS Permanent Registrar Audit

Date	March 2019
Auditors	Shayan Eskandari, Steve Marx
Blog Post	Ethereum Name Service Audited

1 Update [2019-05-29]

In addition to the results below, ConsenSys Diligence also audited an updated `Root` contract with the SHA1 hash `42d807dc438b978b7ddfd7a0c030cf6140bd49d6`.

No new issues were found.

2 Summary

ConsenSys Diligence conducted a security audit on two new components of the Ethereum Name Service (ENS): the `.eth` Permanent Registrar and the new ENS `Root`.

- `.eth` **Permanent Registrar** is a new rent-based registrar for the `.eth` top-level domain.
- **New ENS Root** is to allow for certain `onlyOwner` functions to be disintermediated from the ENS root key holders. The main new functionality is the ability for anyone to register a new TLD based on DNSSEC records.

2.1 Audit Goals

The focus of the audit was to verify that the smart contract system is secure,

resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security: Identifying security related issues within each contract and within the system of contracts.

Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Improving scalability
- Quantity and quality of test coverage

2.2 System Overview

Documentation

The following documentation was available to the audit team:

- This Documentation with description about the mechanism of the `.eth Permanent Registrar` and `ETHRegistrarController`.

Scope

The audit focus was on the smart contract files, and test suites found in the following repositories of the ensdomains GitHub organization:

Directory	Commit hash	Commi date

Directory	Commit hash	Commit date
ensdomains/ethregistrar	e52abfc2799ac36[illegible]fc20f9175a29fd	29th January 2019
ensdomains/root	cc8775499398887274 68f635af94643deeaa10b3	13th February 2019

Specifically, the following files were audited:

- BaseRegistrar.sol
- BaseRegistrarImplementation.sol
- ETHRegistrarController.sol
- Ownable.sol
- PriceOracle.sol
- Root.sol
- SafeMath.sol
- SimplePriceOracle.sol
- StablePriceOracle.sol
- StringUtils.sol

There are other components of the pending ENS upgrade that were *not* part of this audit. Notably, the DNSSEC oracle and DNSSEC-based registrar were out of scope and not considered during this audit.

Architecture

ENS has two principal components: the registry, and resolvers. The system is broadly analogous to DNS.

The ENS registry is a simple smart contract that tracks for each registered name: who owns it, where the name can be resolved, and how long a client may cache name resolution. *Resolvers* (EIP-137) are responsible for actually translating names into addresses.

A *registrar* is a contract responsible for allocating subdomains and updating records in the ENS registry. Registrars can be configured at any level of ENS, and they are pointed to by the owner field of the registry.

There exists an auction-based *interim* `.eth` *registrar* which will be replaced by the new *permanent* `.eth` *registrar* (`BaseRegistrarImplementation`). The new registrar implements an ERC721-compatible non-fungible token. Users can

registrar implements an ERC721 compatible non-fungible token. Users can interact directly with the non-fungible token to transfer ownership. To obtain an available subdomain or renew subdomain ownership, users interact with any number of *controllers* connected to the registrar. At the time of this audit, only one controller exists: the `ETHRegistrarController`. This controller handles domain names that are at least 7 characters long and uses a simple rent-based model similar to `.com` domain ownership in DNS. It uses a price oracle (`StablePriceOracle`) to control rent prices in US dollars and convert those prices to ETH.

A new `Root` contract is being introduced that maps TLDs to registrars. It points the `.eth` TLD at the `BaseRegistrarImplementation`, and it points all others at a registrar specified in DNSSEC records for the TLD or a generic DNSSEC-based registrar as a fallback. This allows DNS owners to dictate how ENS names are resolved for their DNS.

The ENS system itself is controlled by a multisig wallet with key stakeholders from the Ethereum community. They have broad control over the system, up to and including replacing the entire registrar implementation. As such, ENS is only trusted to the extent that these key stakeholders are trusted.

2.3 Key Observations/Recommendations

- The system is well designed. The old registrar does proper checks to be disabled when the new system is deployed.
- The code is well written and considers most of the security best practices.
- **Avoid inline assembly**: A great deal of complexity is added by using libraries with inline assembly code. Wherever possible, we recommend avoiding inline assembly in favor of more straightforward Solidity-based implementations.
- **Insufficient test coverage**: Test *code coverage* is likely high, but there's a lot of room for improvement. See section 6 for more details.

3 Issues

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.

judgment as to whether to address such issues.

- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

3.1 Memory corruption in `Buffer` Critical ✓ Fixed

Resolution
Issue has been closed in ensdomains/buffer#3

Description

Although out of scope for this audit, the audit team noticed a memory corruption issue in the `Buffer` library. The `init` function is as follows:

contracts/Buffer.sol:L22-L41

```
/**
* @dev Initializes a buffer with an initial capacity.
* @param buf The buffer to initialize.
* @param capacity The number of bytes of space to allocate the buffer
* @return The buffer, for chaining.
*/
function init(buffer memory buf, uint capacity) internal pure returns(
    if (capacity % 32 != 0) {
        capacity += 32 - (capacity % 32);
    }
    // Allocate space for the buffer data
    buf.capacity = capacity;
    assembly {
        let ptr := mload(0x40)
        mstore(buf, ptr)
        mstore(ptr, 0)
```

```
            mstore(0x40, add(32, add(ptr, capacity)))
        }
        return buf;
    }
```

Note that memory is reserved only for `capacity` bytes, but the `bytes` actually requires `capacity + 32` bytes to account for the prefixed array length. Other functions in `Buffer` assume correct allocation and therefore corrupt nearby memory.

Although we didn't immediately spot an ENS exploit for this vulnerability, we consider any memory corruption issue to be important to address.

Example

A simple test shows the memory corruption issue:

```
contract Test {
    using Buffer for Buffer.buffer;

    function test() external pure {
        Buffer.buffer memory buffer;
        buffer.init(1);

        // foo immediately follows buffer.buf in memory
        bytes memory foo = new bytes(0);

        assert(foo.length == 0);

        buffer.append("A");

        // "A" == 65, gets written to the high order byte of foo.leng
        assert(foo.length == 65 * 256**31);
    }
}
```

Remediation

Allocate an additional 32 bytes as follows, to account for storing the `uint256`

size of the `bytes` array:

```
mstore(0x40, add(ptr, add(capacity, 32)))
```

3.2 `SimplePriceOracle.price` is susceptible to integer overflow Critical ✓ Fixed

Resolution
Issue has been closed in ensdomains/ethregistrar#17 by using `SafeMath`.

Description

`SimplePriceOracle.price` is as follows:

ethregistrar/contracts/SimplePriceOracle.sol:L26-L28

```
function price(string calldata /*name*/, uint /*expires*/, uint durat
    return duration * rentPrice;
}
```

This is susceptible to a simple overflow attack, e.g. setting the duration to `2**256/rentPrice` to give yourself a price of 0.

Severity note: It's unclear whether the `SimplePriceOracle` is expected to be used in practice, but the severity is set here under the assumption that the code may be used somewhere.

Remediation

Use `SafeMath` or explicitly check for the overflow.

3.3 `ETHRegistrarController.register` is vulnerable to front running Critical ✓ Fixed

Resolution
Issue has been closed in ensdomains/ethregistrar#18

Description

`commit()` and then `register()` appears to serve the purpose of preventing front running. However, because the commitment is not tied to a specific owner, it serves equally well as a commitment for a front-running attacker.

Example

1. Alice calls `commit(makeCommitment("mydomain", <secret>))`.
2. 10 minutes later, Alice submits a transaction to `register("mydomain", Alice, ..., <secret>)`.
3. Eve observes this transaction in the transaction pool.
4. Eve submits `register("mydomain", Eve, ..., <secret>)` with a higher gas price and wins the race.

Remediation

Commitments should commit to `owner`s in addition to `name`s. This way an attacker can't repurpose a previous commitment. (They would have to buy on behalf of the original committer.)

As an alternative, if it's undesirable to pin down `owner`, the commitment could include `msg.sender` instead (only allowing the original committer to call `register`).

E.g. the following (and corresponding changes to callers):

```
function makeCommitment(
    string memory name,
    address owner, /* or perhaps committer/sender */
    bytes32 secret

)
    pure
    public
```

```
    returns(bytes32)
{
    bytes32 label = keccak256(bytes(name));
    return keccak256(abi.encodePacked(label, owner, secret));
}
```

3.4 SOA record check on the wrong domain Major ✓ Fixed

Resolution
During the audit, this issue was discovered by the client development team and already fixed in ensdomains/root#25.

Description

The SOA record check in `Root.getAddress` is meant to happen on the root TLD, but in the version of the code audited, it is performed instead on `_ens.nic.<tld>`.

3.5 Work towards a trustless model for ENS Medium Acknowledged

Resolution
Acknowledged by client team. As stated, this is a long-term issue for which there is no immediate fix, but work is already in progress.

Description

The ENS registry itself is owned by a multisig wallet owned by a number of reputable Ethereum community members. That multisig wallet can do just about

anything, up to and including directly taking over any existing or future registered names.

It's important to note that even if we as a community trust the current owners of the multisig wallet, we also need to consider the possibility of their Ethereum private keys being compromised by malicious actors.

Remediation

This centralized control is by design, and the multisig owners have been chosen carefully. However, we do recommend—as is already the plan—that the multisig wallet's power be reduced in future updates to the system. Changes made by that wallet are already quite transparent to the community, but future enhancements might include requiring a waiting period for any changes or disallowing certain types of changes altogether.

In the meantime, wherever possible, the trust model should be made clear so that users understand what guarantees they do and do not have when interacting with ENS.

3.6 Consider replacing the `Buffer` implementation Medium ✓ Fixed

Resolution
There will be no immediate fix for this, but the client team is working on collaborating to get a better audited `Buffer` library in place.

Description

The audit team uncovered two bugs in the `Buffer` library, one each in the only two functions that were looked at. (The library was in general *not* in scope for this audit.) One bug was a critical memory corruption bug. This calls into question how safe this library is to use in general.

Remediation

Consider using a different library, ideally one that has been fully tested and audited and that minimizes the use of inline assembly, particularly around memory allocation.

memory allocation.

3.7 Overzealous resizing in `Buffer` Medium ✓ Fixed

Resolution
Issue has been closed in ensdomains/buffer#4

Description

In the following code, the buffer is resized even when sufficient `capacity` is available to perform the write. The `buf.buf.length` term is unnecessary and leads to unnecessary resizing:

contracts/Buffer.sol:L91-L95

```
function write(buffer memory buf, uint off, bytes memory data, uint le
    require(len <= data.length);

    if (off + len > buf.capacity) {
        resize(buf, max(buf.capacity, len + off) * 2);
```

Contrast with the calculation in a similar function:

contracts/Buffer.sol:L206-L209

```
function write(buffer memory buf, uint off, bytes32 data, uint len) pr
    if (len + off > buf.capacity) {
        resize(buf, (len + off) * 2);
    }
```

Remediation

Check just the condition `if (off + len > buf.capacity)` when deciding whether to resize the buffer. This will be a significant gas savings in the common case of reserving exactly the right capacity and then performing two `append` operations.

3.8 Pending auctions in the legacy registrar don't result in proper ownership in ENS Medium ✓ Fixed

Resolution
Addressed in ensdomains/ethregistrar#23 by reducing the waiting period to 28 days.

Description

If an auction has yet to be finalized in the legacy `HashRegistrar` at the time that the new, permanent `.eth` registrar is put in place, the auction winner doesn't get actual ownership of the ENS entry.

The sequence of events would look like:

1. Auction is started in the `HashRegistrar` for the name `something.eth`
2. The new `BaseRegistrarImplementation` becomes the owner of the `.eth` root node in ENS.
3. The auction is won.
4. The auction winner calls `finalizeAuction`, which calls `trySetSubnodeOwner`, which fails to actually set subnode ownership (as the `HashRegistrar` no longer has ownership of the `.eth` root node).

At this point, there's an owner of the deed for the name `something.eth` in the `HashRegistrar`, but the ENS subnode is unowned. It can't be transferred to the new registrar for 183 days, and the name can't be registered in the new registrar.

The owner *can* get themselves out of this situation by calling `releaseDeed` in the `HashRegistrar`. If they want to avoid potentially losing their domain in the process, they can transfer the deed to a smart contract which can then release the deed and rent the same name in the new registrar atomically.

Remediation

Here are a few ideas of improvements to help in this situation:

1. Discourage (or prevent, if possible) new auctions very close to the launch of

the new registrar.

2. Allow domains to be transferred before the 183-day waiting period but require rent payment in those cases. (Perhaps just use the existing grace period to have people renew?)
3. Document the process for rescuing names that get stuck in this state, or better yet provide a tool for doing so.

3.9 `BaseRegistrarImplementation.acceptRegistrarTransfer` should probably use the `live` modifier Medium ✓Fixed

Resolution
Issue has been closed in ensdomains/ethregistrar#19.

Description

Most `external` functions in `BaseRegistrarImplementation` have the `live` modifier, which ensures that they can only be called on the current ENS owner of the registrar's base address. The `acceptRegistrarTransfer` function does *not* have this modifier, which means names can be transferred to the new registrar even if it's not the proper registry owner.

It's hard to think of a real-world example of why this is problematic, especially because the interim registrar appears to protect against this by only transferring to the `ens.owner`, but it seems safer to include the `live` modifier unless there's a specific reason not to.

Remediation

Add the `live` modifier to `acceptRegistrarTransfer`.

3.10 Reconsider use of inline assembly in `BytesUtils.sol` Minor ✓Fixed

Resolution

Issue has been closed in ensdomains/dnssec-oracle#55

Description

`Root.sol` imports and uses `@ensdomains/dnssec-oracle/contracts/BytesUtils.sol` for byte operations.

`BytesUtils.sol` is mainly written in assembly. In general, inline assembly is concerning from a security perspective because it bypasses compiler checks and inhibits human code reasoning.

e.g. `readUint8()`:

```
function readUint8(bytes memory self, uint idx) internal pure returns
    require(idx + 1 <= self.length);
    assembly {
        ret := and(mload(add(add(self, 1), idx)), 0xFF)
    }
}
```

Remediation

Some of the functions in `BytesUtil.sol` can be written in Solidity without affecting the gas costs.

`readUint8()` can be written as following Solidity code which functions the same:

```
function readUint8(bytes memory self, uint idx) internal pure returns
    return uint8(self[idx]);
}
```

3.11 `BaseRegistrarImplementation.acceptRegistrarTransfer` does not check for invalid names Minor ✓ Fixed

Resolution

Short names will be manually canceled in the old registrar during the migration period. Note that this is still feasible with the reduced 28-day lock-up period.

Description

`BaseRegistrarImplementation.acceptRegistrarTransfer` does not explicitly check for invalid names.

In the old registrar it is possible to register domain names with length less than 7 characters. However anyone can call `HashRegistrar.invalidateName()` to invalidate the registration and get half of the deed amount as an incentive.

Assume that an invalid domain is registered in the old registrar and no one invalidates the registration (within the 183 days between the `registrationDate` and the transfer `ETHRegistrarController.acceptRegistrarTransfer`), it is possible to transfer the invalid domain to the new ENS registrar.

Remediation

Given that it is easy to check for invalid domains using a rainbow table for all possible <7 character domains, anyone can invalidate them before the new registrar goes live. Note that for the auctions starting right before the new registrar goes live, there will be a 183 days window in which anyone can call `HashRegistrar.invalidateName()` to invalidate the domain names.

3.12 Sanity check around `transferPeriodEnds` Minor ✓ Fixed

Resolution

Issue has been closed in ensdomains/ethregistrar#23

Description

`BaseRegistrarImplementation.acceptRegistrarTransfer` has a hardcoded limit such that only domains registered 183 days ago can be transferred in.

This imposes an implicit constraint on the `transferPeriodEnds` state variable. If the transfer period ends too soon after the new registrar is put in place, names that were just registered won't be transferrable during the transfer period (and will thus become available to be rented by another user).

Remediation

A sanity check in the constructor would help here, e.g.:

```
require(_transferPeriodEnds > now + 183 days);
```

Note that the true requirement is something more like "The time between when this registrar becomes the ENS node owner of the .eth domain and the time of `transferPeriodEnds` must be at least 183 days plus a sufficient time window for late registrants to have a chance to perform the transfer." But it's hard to see a way to encode this precisely. A broad sanity check will at least avoid simple timing mistakes.

3.13 `StablePriceOracle.price` has an unimportant integer underflow Minor ✓ Fixed

Resolution
Issue has been closed in ensdomains/ethregistrar#20

Description

ethregistrar/contracts/StablePriceOracle.sol:L57-L63

```
function price(string calldata name, uint /*expires*/, uint duration)
    uint len = name.strlen();
    require(len > 0);
    if(len > rentPrices.length) {
        len = rentPrices.length;
    }
    uint priceUSD = rentPrices[len - 1].mul(duration);
```

If the length of the `rentPrices` array is 0, then the last line above attempts to access `rentPrices[2**256-1]`. This will `assert`, but it might be more friendly (from a gas perspective) to `revert` in this case.

Remediation

A simple fix would be to move the `require(len > 0)` down until just before the array access.

3.14 `ETHRegistrarController.register` should `revert` rather than silently fail Minor ✓ Fixed

Resolution
Issue has been closed in ensdomains/ethregistrar#22

Description

When called with an invalid commitment or unavailable domain, `ETHRegistrarController.register` refunds the sent ether and silently fails rather than `revert`ing:

ethregistrar/contracts/ETHRegistrarController.sol:L56-L64

```
function register(string calldata name, address owner, uint duration,
    // Require a valid commitment
    bytes32 commitment = makeCommitment(name, secret);
    require(commitments[commitment] + MIN_COMMITMENT_AGE <= now);

    // If the commitment is too old, or the name is registered, stop
    if(commitments[commitment] + MAX_COMMITMENT_AGE < now || !availabl
        msg.sender.transfer(msg.value);
        return;
```

`register` also has no return value, so it's difficult for a caller to know whether the `register` action succeeded or failed.

Remediation

It's probably better to use `require(...)` to handle these invalid cases. This is roughly equivalent because no state changes have been made before this early `return`, but it seems less error prone and clearer to callers about what happened.

3.15 `StringUtils.strlen` could be rewritten without assembly Minor ✓ Fixed

Resolution
Issue has been closed in ensdomains/ethregistrar#21

Description

`StringUtils.strlen` uses inline assembly to walk through a UTF-8 string and count its character length. In general, inline assembly is concerning from a security perspective because it bypasses compiler checks and inhibits human code reasoning.

Remediation

Consider rewriting in Solidity, something similar to the following:

```
function strlen(string memory s) internal pure returns (uint256) {
    uint256 i = 0;
    uint256 len;
    for (len = 0; i < bytes(s).length; len++) {
        byte b = bytes(s)[i];
        if (b < 0x80) {
            i += 1;
        } else if (b < 0xE0) {
            i += 2;
        }

        ...
    }
```

```
    return len;
}
```

4 Threat Model

The creation of a threat model is beneficial when building smart contract systems as it helps to understand the potential security threats, assess risk, and identify appropriate mitigation strategies. This is especially useful during the design and development of a contract system as it allows to create a more resilient design which is more difficult to change post-development.

A threat model was created during the audit process in order to analyze the attack surface of the contract system and to focus review and testing efforts on key areas that a malicious actor would likely also attack. It consists of two parts: a high-level analysis that help to understand the attack surface and a list of threats that exist for the contract system.

4.1 Overview

The following assets are managed by contracts and likely targets for an attacker:

- Registered domain names (e.g. `foo.eth`)
- Ether, in the form of rent paid to the `ETHRegistrarController`

The following actors have access to the system to perform an attack:

- System owners (ENS itself, registrars, controllers, price oracles)
- DNS domain/subdomain owners, who can update DNSSEC records
- Users who are registering, renewing, and transferring domains

The following describes the surface area available to attackers:

- DNSSEC records
- Registrars and controllers
- `Root` contract

- Ethereum private keys

Because they were out of scope for this audit, we did *not* consider some

interesting targets such as the DNSSEC oracle, DNSSEC-based registrar, the interim `.eth` registrar, or the multisig wallet used for ENS ownership.

4.2 Threat Analysis

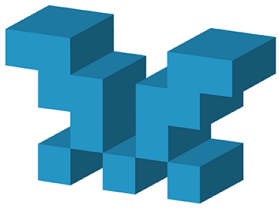
The following table contains a list of identified threats, along with their mitigations:

Threat	Attack Strategy	Mitigation
user may try to register/renew a domain for less than the expected price	overflow on the rent price / manipulate the price oracle	SafeMath mitigates some potential math errors
user may try to mount denial-of-service attacks on other users (e.g. censor their purchases/renewals)	network DoS	long purchase windows and grace periods
user may try to snipe a domain	front-running	a commit/reveal scheme attempts to prevent this but is ineffective (see section 3), a generous grace period prevents race conditions on expiration
user may try to register `.eth` TLD	update DNSSEC records	`Root` disallows changes to that node
`Root` owner may steal domains, manipulate prices, etc.	ENS root swaps the controller/registrar with malicious code	such manipulation would be transparent today, and future updates may limit the root owners' powers
domain owners may take over already-owned subdomains	change DNSSEC to replace registrar for a domain	this is allowed by design

5 Tool-based analysis

The issues from the tool based analysis have been reviewed and the relevant issues have been listed in chapter 3 - Issues.

5.1 MythX



MythX is a security analysis API for Ethereum smart contracts. It performs multiple types of analysis, including fuzzing and symbolic execution, to detect many common vulnerability types. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on MythX can be found at mythx.io.

Where possible, we ran the full MythX analysis. MythX is still in beta, and where analysis failed, we fell back to running Mythril Classic, a large subset of the functionality of MythX.

Below is the raw output of MythX and Mythril Classic vulnerability scans:

In order to run MythX, `Root.sol` contract was flattened. flat_root.sol line numbers reflect on the output of `truffle-flattener contracts/Root.sol`.

```
Title: Floating Pragma
Head: A floating pragma is set.
Description: It is recommended to make a conscious choice on what vers
Source code:

flat_root.sol 1:0
-----------------------------------------------
pragma solidity ^0.4.24;
-----------------------------------------------


================================================
 Title: Shadowing State Variables
Head: State variable shadows another state variable.
Description: The state variable "CLASS_INET" in contract "DNSClaimChec
Source code:

flat_root.sol 869:4
-----------------------------------------------
uint16 constant CLASS_INET = 1
-----------------------------------------------
```

```
==================================================
 Title: Shadowing State Variables
Head: State variable shadows another state variable.
Description: The state variable "TYPE_TXT" in contract "DNSClaimChecke
Source code:

flat_root.sol 870:4
--------------------------------------------------
uint16 constant TYPE_TXT = 16
--------------------------------------------------

==================================================
 Title: Shadowing State Variables
Head: Local variable shadows a state variable.
Description: The local variable "owner" in contract "ENS" shadows the
Source code:

flat_root.sol 20:58
--------------------------------------------------
address owner
--------------------------------------------------
flat_root.sol 22:36
--------------------------------------------------
address owner
--------------------------------------------------

==================================================
 Title: Shadowing State Variables
Head: Local variable shadows a state variable.
Description: The local variable "owner" in contract "Root" shadows the
Source code:

flat_root.sol 1012:44
--------------------------------------------------
address owner
--------------------------------------------------
flat_root.sol 1037:36
--------------------------------------------------
address owner
--------------------------------------------------
```

```
==================================================
 Title: Shadowing State Variables
Head: Local variable shadows a state variable.
Description: The local variable "oracle" in contract "DNSClaimChecker"
Source code:

flat_root.sol 881:29
--------------------------------------------------
DNSSEC oracle
--------------------------------------------------

==================================================
```

BaseRegistrarImplementation

Mythril Classic results for `BaseRegistrarImplementation` are as follows. flat_BaseRegistrarImplementation.sol line numbers reflect on the output of `truffle-flattener contracts/BaseRegistrarImplementation.sol`

```
SWC ID: 110
Severity: Low
Contract: BaseRegistrarImplementation
Function name: [available(uint256), available(uint256)] (ambiguous)
PC address: 4722
Estimated Gas Usage: 3414 - 38591
A reachable exception has been detected.
It is possible to trigger an exception (opcode 0xfe). Exceptions can be caus
--------------------
In file: flat_BaseRegistrarImplementation.sol:1443

previousRegistrar.state(bytes32(id)) == Registrar.Mode.Open

--------------------
```

ETHRegistrarController

Mythril Classic results for `ETHRegistrarController` are as follows. flat_ETHRegistrarController.sol line numbers reflect on the output of `truffle-flattener contracts/ETHRegistrarController.sol`

`flattener contracts/ETHRegistrarController.sol`.

```
==== Multiple Calls in a Single Transaction ====
SWC ID: 113
Severity: Medium
Contract: ETHRegistrarController
Function name: rentPrice(string,uint256)
PC address: 996
Estimated Gas Usage: 5179 - 79151
Multiple sends are executed in one transaction.
Consecutive calls are executed at the following bytecode offsets:
Offset: 2947
Offset: 3202
Try to isolate each external call into its own transaction, as externa

--------------------
In file: flat_ETHRegistrarController.sol:1467

function rentPrice(string memory name, uint duration) view public retu
        bytes32 hash = keccak256(bytes(name));
        return prices.price(name, base.nameExpires(uint256(hash)), du
    }

--------------------

==== Dependence on predictable environment variable ====
SWC ID: 116
Severity: Low
Contract: ETHRegistrarController
Function name: register(string,address,uint256,bytes32)
PC address: 3552
Estimated Gas Usage: 2056 - 6247
Sending of Ether depends on a predictable variable.
The contract sends Ether depending on the values of the following vari
- block.timestamp
- block.timestamp
- block.timestamp

Note that the values of variables like coinbase, gaslimit, block numbe
--------------------
In file: flat_ETHRegistrarController.sol:1498
```

```
msg.sender.transfer(msg.value)

--------------------

==== Integer Overflow ====
SWC ID: 101
Severity: High
Contract: ETHRegistrarController
Function name: register(string,address,uint256,bytes32)
PC address: 5332
Estimated Gas Usage: 2333 - 9254
The binary addition can overflow.
The operands of the addition operation are not sufficiently constraine
--------------------
In file: flat_ETHRegistrarController.sol:1411

add(mload(s), ptr)

--------------------
```

5.2 Ethlint

Ethlint is an open source project for linting Solidity code. Only security-related issues were reviewed by the audit team.

Below is the raw output of the Ethlint vulnerability scan:

ethregistrar

```
contracts/BaseRegistrarImplementation.sol
  36:36     warning    Avoid using 'now' (alias to 'block.timestamp')
  60:42     warning    Avoid using 'now' (alias to 'block.timestamp')
  65:15     warning    Avoid using 'now' (alias to 'block.timestamp')
  73:16     warning    Avoid using 'now' (alias to 'block.timestamp')
  73:48     warning    Avoid using 'now' (alias to 'block.timestamp')
  75:23     warning    Avoid using 'now' (alias to 'block.timestamp')
  83:39     warning    Avoid using 'now' (alias to 'block.timestamp')
```

```
  85:15     warning    Avoid using 'now' (alias to 'block.timestamp')
  89:47     warning    Avoid using 'now' (alias to 'block.timestamp')
  114:37    warning    Avoid using 'now' (alias to 'block.timestamp')
  118:35    warning    Avoid using 'now' (alias to 'block.timestamp')

contracts/ETHRegistrarController.sol
  53:63    warning    Avoid using 'now' (alias to 'block.timestamp').
  54:34    warning    Avoid using 'now' (alias to 'block.timestamp').
  61:64    warning    Avoid using 'now' (alias to 'block.timestamp').
  64:58    warning    Avoid using 'now' (alias to 'block.timestamp').

contracts/StringUtils.sol
  15:8     error    Avoid using Inline Assembly.    security/no-inline
  22:12    error    Avoid using Inline Assembly.    security/no-inline

✖ 2 errors, 15 warnings found.
```

root

```
No issues found.
```

5.3 Surya

Surya is an utility tool for smart contract systems. It provides a number of visual outputs and information about structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Below is a complete list of functions with their visibility and modifiers:

Files Description Table

File Name	SHA-1 Has
root/contracts/Migrations.sol	eac3bb098bace681296263c0
root/contracts/Ownable.sol	b596da7ad9b5c92a119268e0

File Name	SHA-1 Has
root/contracts/Root.sol	94c5fd45635c6d78[illegible]
ethregistrar/contracts/BaseRegistrar.sol	dfadfc8a35024069ff66cbc4a
ethregistrar/contracts/BaseRegistrarImplementation.sol	a1e04ce66a9588063155591b
ethregistrar/contracts/DummyOracle.sol	e1dab33211d55e02874ae251
ethregistrar/contracts/ETHRegistrarController.sol	7cb180a1d5102efd2acc04b0
ethregistrar/contracts/Migrations.sol	b6732a145e4cb6841945488f
ethregistrar/contracts/PriceOracle.sol	3257acda730f294f19984163
ethregistrar/contracts/SafeMath.sol	5effc6db2209b2bf2d49abe4a
ethregistrar/contracts/SimplePriceOracle.sol	fc11bff8c93e8471b8d8478f1
ethregistrar/contracts/StablePriceOracle.sol	892333542a757ba6089c5c3c
ethregistrar/contracts/StringUtils.sol	4d784bb26b409cfd8ed841f43
ethregistrar/contracts/_TestDeps.sol	2077d541fedbd889d2f814c5

Contracts Description Table

Contract	Type	Bases	
└	**Function Name**	**Visibility**	**Mutabilit**
Migrations	Implementation		
└	<Constructor>	Public ❗	🛑
└	setCompleted	Public ❗	🛑
└	upgrade	Public ❗	🛑
Ownable	Implementation		
└	<Constructor>	Public ❗	🛑
└	transferOwnership	Public ❗	🛑
└	isOwner	Public ❗	
Root	Implementation	Ownable	
└	<Constructor>	Public ❗	🛑

Contract	Type	Bases	
└	proveAndClaimRegisterTLD	External ❗	🛑
└	setSubnodeOwner	External ❗	🛑
└	setRegistrar	External ❗	🛑
└	registerTLD	Public ❗	🛑
└	setResolver	Public ❗	🛑
└	setOwner	Public ❗	🛑
└	setTTL	Public ❗	🛑
└	getLabel	Internal 🔒	
└	getAddress	Internal 🔒	
└	getSOAHash	Internal 🔒	
BaseRegistrar	Implementation	ERC721, Ownable	
└	addController	External ❗	🛑
└	removeController	External ❗	🛑
└	nameExpires	External ❗	
└	available	Public ❗	
└	register	External ❗	🛑
└	renew	External ❗	🛑
└	reclaim	External ❗	🛑
└	acceptRegistrarTransfer	External ❗	🛑
BaseRegistrarImplementation	Implementation	BaseRegistrar	
└	<Constructor>	Public ❗	🛑
└	ownerOf	Public ❗	
└	addController	External ❗	🛑
└	removeController	External ❗	🛑

└	nameExpires	External ❗	

Contract	Type	Bases	
└	available	Public ❗	
└	register	External ❗	🛑
└	renew	External ❗	🛑
└	reclaim	External ❗	🛑
└	acceptRegistrarTransfer	External ❗	🛑
DummyOracle	Implementation		
└	<Constructor>	Public ❗	🛑
└	set	Public ❗	🛑
└	read	External ❗	
ETHRegistrarController	Implementation	Ownable	
└	<Constructor>	Public ❗	🛑
└	rentPrice	Public ❗	
└	valid	Public ❗	
└	available	Public ❗	
└	makeCommitment	Public ❗	
└	commit	Public ❗	🛑
└	register	External ❗	💵
└	renew	External ❗	💵
└	setPriceOracle	Public ❗	🛑
└	withdraw	Public ❗	🛑
Migrations	Implementation		
└	<Constructor>	Public ❗	🛑
└	setCompleted	Public ❗	🛑
└	upgrade	Public ❗	🛑

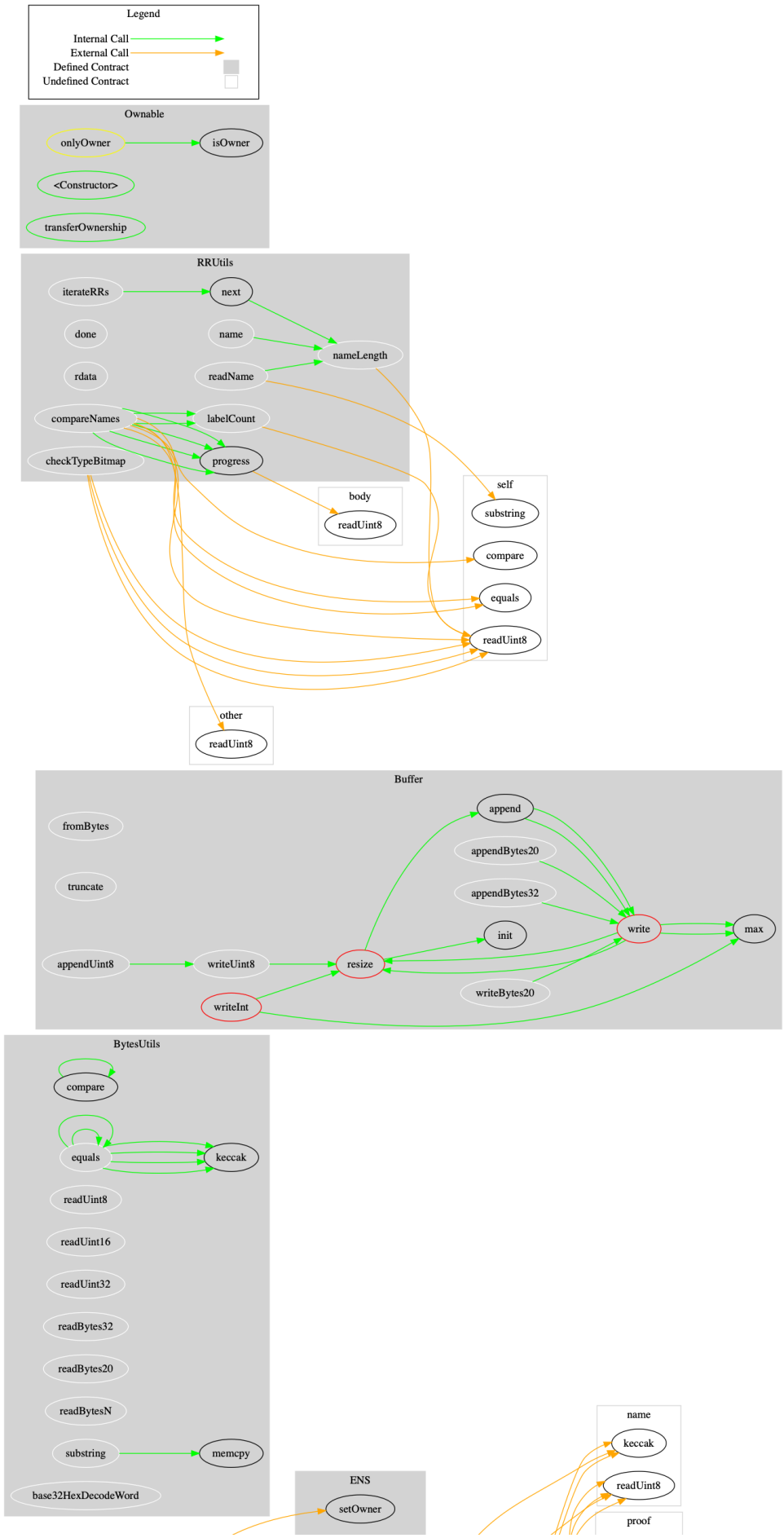
Contract	Type	Bases	
PriceOracle	Interface		
└	price	External ❗	
SafeMath	Library		
└	mul	Internal 🔒	
└	div	Internal 🔒	
└	sub	Internal 🔒	
└	add	Internal 🔒	
└	mod	Internal 🔒	
SimplePriceOracle	Implementation	Ownable, PriceOracle	
└	<Constructor>	Public ❗	🛑
└	setPrice	Public ❗	🛑
└	price	External ❗	
DSValue	Interface		
└	read	External ❗	
StablePriceOracle	Implementation	Ownable, PriceOracle	
└	<Constructor>	Public ❗	🛑
└	setOracle	Public ❗	🛑
└	setPrices	Public ❗	🛑
└	price	External ❗	
StringUtils	Library		
└	strlen	Internal 🔒	

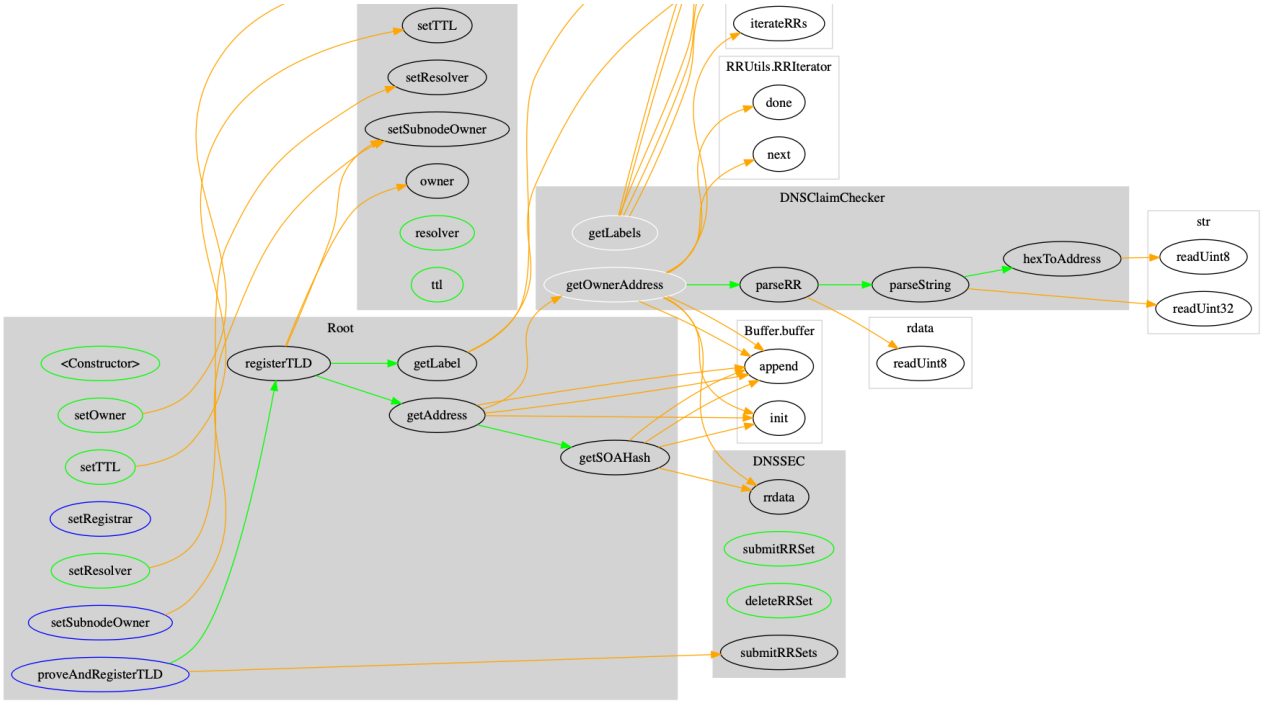
Legend

Symbol	Meaning
🛑	Function can modify state

Symbol	Meaning
$	Function is payable

Root Control Flow





6 Test Coverage Measurement

Testing is implemented using Truffle. 12 tests are included for the `Root` contract, and they all pass. 30 tests are included for the `.eth` permanent registrar, and they all pass.

We were unable to obtain code coverage numbers for the tests, but the audit team's overall impression is that testing covers a high percentage of code branches. That said, the testing is weak, in particular regarding negative test cases and edge cases. As a specific example, changing the following in `ETHRegistrarController.renew` causes no test failures, which shows a serious lack of coverage:

```
// OLD: require(msg.value >= cost);
// NEW:
require(msg.value > 0);
```

Appendix 1 - File Hashes

The SHA1 hashes of the source code files in scope of the audit are listed in the table below.

File Name	SHA-1 Has
root/contracts/Ownable.sol	b596da7ad9b5c92a119268e(

File Name	SHA-1 Has
root/contracts/Root.sol	94c5fd45635c6d78[illegible]
ethregistrar/contracts/BaseRegistrar.sol	dfadfc8a35024069ff66cbc4a
ethregistrar/contracts/BaseRegistrarImplementation.sol	a1e04ce66a9588063155591b
ethregistrar/contracts/ETHRegistrarController.sol	7cb180a1d5102efd2acc04b0
ethregistrar/contracts/PriceOracle.sol	3257acda730f294f19984163
ethregistrar/contracts/SafeMath.sol	5effc6db2209b2bf2d49abe4a
ethregistrar/contracts/SimplePriceOracle.sol	fc11bff8c93e8471b8d8478f1
ethregistrar/contracts/StablePriceOracle.sol	892333542a757ba6089c5c3c
ethregistrar/contracts/StringUtils.sol	4d784bb26b409cfd8ed841f4

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the

absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.