ENS Contest
Findings & Analysis Report

2023-08-31

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the ENS smart contract system written in Solidity. The audit took place between April 14—April 28 2023.

Wardens

65 Wardens contributed reports to the ENS audit:

1. 0x73696d616f
2. 0xAgro
3. 0xSmartContract
4. 0xTheC0der
5. ABA
6. AkshaySrivastav

7. ArbitraryExecution (cr0, arbitrary-wanaks, tridearm, CodeBeholder, WGMIApe, pbwaffles, and yowl)
8. Aymen0909
9. BRONZEDISC
10. Bauchibred
11. Dyear
12. Eurovickk
13. Holmgren
14. IceBear
15. J4de
16. JCN
17. Jerry0x
18. Jorgect
19. Josiah
20. Kek
21. Lilyjjo
22. MalfurionWhitehat
23. MohammedRizwan
24. Parad0x
25. RaymondFam
26. Recep
27. Rickard
28. SAAJ
29. SaharAP
30. Sathish9098
31. Shubham
32. Shyn
33. Trust
34. Udsen

35. auditor0517
36. bin2chen
37. brgltd
38. catellatech
39. chaduke
40. codeslide
41. d3e4
42. descharre
43. eierina
44. favelanky
45. hassan-truscova
46. hihen
47. j4ld1na
48. lukris02
49. matrix_0wl
50. naman1778
51. niser93
52. nobody2018
53. openwide
54. pontifex
55. rvierdiiev
56. saneryee
57. schrodinger
58. tnevler
59. urataps

This audit was judged by LSDan.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 7 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 7 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 45 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 8 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the **C4 ENS audit repository**, and is composed of 18 smart contracts written in the Solidity programming language and includes 2,022 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on **the C4 website**, specifically our section on **Severity Categorization**.

Medium Risk Findings (7)

[M-01] `HexUtils.hexStringToBytes32()` and `HexUtils.hexToAddress()` may return incorrect results

Submitted by hihen*, also found by* chaduke*,* eierina*,* AkshaySrivastav*,* Kek*, and* chaduke

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/utils/HexUtils.sol#L11
https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/utils/HexUtils.sol#L68

The `HexUtils.hexStringToBytes32()` and `HexUtils.hexToAddress()` may return incorrect results if the input data provided is not in a standard format.

This could cause the contract to behave abnormally in some scenarios or be exploited for malicious purposes.

Proof of Concept

The function `HexUtils.hexStringToBytes32(bytes memory str, uint256 idx, uint256 lastIdx)` is used to convert a hexadecimal string `str[idx...lastIndx]` to a `bytes32`.

However, the function lacks some critical checks on the input data, resulting in the following situations:

1. If the length `lastIdx - idx` is odd, it will not revert, but will read an additional out-of-range byte `str[lastIdx]` and return it.
2. If the length `lstIdx - idx > 32`, it will not revert, but will discard the excess data at the beginning and return the last 32 bytes.
3. If the length `lstIdx - idx < 32`, it will not revert, but will pad the data with zeros at the beginning.

The following test code verifies these situations:

```
diff --git a/test/utils/HexUtils.js b/test/utils/HexUtils.js
index 296eadf..e12e11c 100644
--- a/test/utils/HexUtils.js
+++ b/test/utils/HexUtils.js
@@ -16,6 +16,44 @@ describe('HexUtils', () => {
     HexUtils = await HexUtilsFactory.deploy()
   })

+  describe.only('Special cases for hexStringToBytes32()', () =>
+    const hex32Bytes = '5cee339e13375638553bdf5a6e36ba80fb9f6a4
+    it('odd length 1', async () => {
+      let [bytes32, valid] = await HexUtils.hexStringToBytes32(
+        toUtf8Bytes(hex32Bytes), 0, 63,
+      )
+      expect(valid).to.equal(true)
+      // the last 4 bits (half byte) of hex32Bytes is out of ra
+      expect(bytes32).to.equal('0x' + hex32Bytes)
+    })
+    it('odd length 2', async () => {
+      let [bytes32, valid] = await HexUtils.hexStringToBytes32(
+        toUtf8Bytes(hex32Bytes + '00'), 1, 64,
+      )
+      expect(valid).to.equal(true)
+      // the first half byte of '00' is out of range but read
+      expect(bytes32).to.equal('0x' + hex32Bytes.substring(1) +
+    })
+    it('not enough length', async () => {
+      let [bytes32, valid] = await HexUtils.hexStringToBytes32(
+        toUtf8Bytes(hex32Bytes), 0, 2,
+      )
+      expect(valid).to.equal(true)
+      // only one byte is read, but it is expanded to 32 bytes
+      expect(bytes32).to.equal(
+        '0x0000000000000000000000000000000000000000000000000000
+      )
+    })
+    it('exceed length', async () => {
+      let [bytes32, valid] = await HexUtils.hexStringToBytes32(
+        toUtf8Bytes(hex32Bytes + "1234"), 0, 64 + 4,
+      )
+      expect(valid).to.equal(true)
+      // 34 bytes is read, and returns the last 32 bytes
+      expect(bytes32).to.equal('0x' + hex32Bytes.substring(4) +
+    })
+  })
```

```
+
    describe('hexStringToBytes32()', () => {
      it('Converts a hex string to bytes32', async () => {
        let [bytes32, valid] = await HexUtils.hexStringToBytes32(
```

Test code outputs:

```
  HexUtils
    Special cases for hexStringToBytes32()
      ✓ odd length 1
      ✓ odd length 2
      ✓ not enough length
      ✓ exceed length
```

Since HexUtils.hexToAddress() is implemented by directly calling `HexUtils.hexStringToBytes32()`, it also has similar issues.

Tools Used

VS Code

Recommended Mitigation Steps

Should revert the function if the input length `lastIdx - idx` is odd.

For cases where the length is greater than or less than 32 (or 20)

- if the current implementation meets the requirements, the design should be detailed in a comment
- otherwise the function should revert if the length is not 32 (or 20)

Arachnid (ENS) confirmed

[M-02] Invalid addresses will be accepted as resolvers, possibly bricking assets

Submitted by Trust

The `hexToAddress` utility parses a string into an address type.

```
    function hexToAddress(
        bytes memory str,
        uint256 idx,
        uint256 lastIdx
    ) internal pure returns (address, bool) {
        if (lastIdx - idx < 40) return (address(0x0), false);
        (bytes32 r, bool valid) = hexStringToBytes32(str, idx, ]
        return (address(uint160(uint256(r))), valid);
    }
```

The issue is that in the return statement, the `bytes32` type is reduced from 256 bits to 160 bits, without checking that the truncated bytes are zero. When the upper bits are not zero, the `bytes32` is not a valid address and the function must return `false` in the second parameter. However, it instead returns an incorrect trimmed number.

The utility is used in the flow below:

```
proveAndClaim
        _claim
                getOwnerAddress
                        parseRR
                                parseString
                                        hexToAdress
```

The incorrect address would be returned from `_claim` and then used as the owner address for `ens.setSubnodeOwner`. This would brick the node.

```
function proveAndClaim(
    bytes memory name,
    DNSSEC.RRSetWithSignature[] memory input
) public override {
    (bytes32 rootNode, bytes32 labelHash, address addr) = _clain
        name,
        input
    );
```

```
        ens.setSubnodeOwner(rootNode, labelHash, addr);
    }
```

Impact

Lack of validation can lead to ENS addresses becoming permanently bricked.

Recommended Mitigation Steps

Add a check for the upper 96 bits in the highlighted return statement.

Note for judge

Historically, the judge has awarded medium severity to various issues which rely on some user error, are easy to check/fix and present material risk. I respect this line of thought and for the sake of consistency I believe this submission should be judged similarly.

Arachnid (ENS) disputed and commented:

> The upshot of this is that if you provide an invalid address in DNS, it will be set to a (possibly different) invalid address in ENS. This can be fixed by correcting the address in DNS and calling proveAndClaim again. I would argue this should be considered minor at best.

[M-03] Offchain name resolution would fail despite the located DNS resolver being fully functional

Submitted by Trust

In OffchainDNSResolver, `resolveCallback` parses resource records received off-chain and extracts the DNS resolver address:

```
                // Look for a valid ENS-DNS TXT record
                (address dnsresolver, bytes memory context) = pa
                        iter.data,
                        iter.rdataOffset,
                        iter.nextOffset
```

```
        );
```

The contract supports three methods of resolution through the resolver:

1. IExtendedDNSResolver.resolve
2. IExtendedResolver.resolve
3. Arbitrary call with the `query` parameter originating in `resolve()`

Code is below:

```
// If we found a valid record, try to resolve it
if (dnsresolver != address(0)) {
    if (
        IERC165(dnsresolver).supportsInterface(
            IExtendedDNSResolver.resolve.selector
        )
    ) {
        return
            IExtendedDNSResolver(dnsresolver).resolv
                name,
                query,
                context
            );
    } else if (
        IERC165(dnsresolver).supportsInterface(
            IExtendedResolver.resolve.selector
        )
    ) {
        return IExtendedResolver(dnsresolver).resolv
    } else {
        (bool ok, bytes memory ret) = address(dnsres
            .staticcall(query);
        if (ok) {
            return ret;
        } else {
            revert CouldNotResolve(name);
        }
    }
}
```

The issue is that a resolver could support option (3), but execution would revert prior to the `query` call. The function uses `supportsInterface` in an unsafe way. It should first check that the contract implements ERC165, which will guarantee the call won't revert. Dynamic resolvers are not likely in practice to implement ERC165 as there's no specific signature to support ahead of time.

Impact

Resolution with custom DNS resolvers are likely to fail.

Recommended Mitigation Steps

Use the OZ `ERC165Checker.sol` library, which addresses the issue:

```
function supportsInterface(address account, bytes4 interface
    // query support of both ERC165 as per the spec and supp
    return supportsERC165(account) && supportsERC165Interfac
}
```

Arachnid (ENS) disputed and commented:

> ERC165 support is required in order to be a valid resolver. Any resolver that doesn't support it is incorrectly implemented.

LSDan (judge) commented:

> This is easy to protect against. Issue stands.

Arachnid (ENS) commented:

> There's no point building a protection for this; either way the result is a failed resolution.

LSDan (judge) commented:

> The OZ implementation would guarantee that the else clause gets triggered and the error returned is understandable / sane. In this case, a very simple fix will

> significantly enhance the composability of the protocol and improve the experience of dev users.

Arachnid (ENS) commented:

> I continue to disagree this is an issue. ERC165 support is a baseline requirement for a resolver; checking it's supported is a pointless waste of gas.

lllllll (warden) commented:

> https://github.com/code-423n4/2023-04-ens/blob/83836eff1975fb47dae6b0982afd0b00294165cf/contracts/utils/UniversalResolver.sol#L498-L510 this code shows that at least in other areas, the possibility failure is acknowledged and handled.

[M-04] Incorrect implementation of `RecordParser.readKeyValue()`

Submitted by hihen, *also found by* chaduke, eierina, Parad0x, rvierdiiev, nobody2018, bin2chen, *and* chaduke

`RecordParser.readKeyValue()` returns a wrong `value` if the terminator not found.
This is a fundamental library and any contract using it may experience unexpected errors and problems due to this bug.

Proof of Concept

The implementation logic of `RecordParser.readKeyValue(bytes memory input, uint256 offset, uint256 len)` is roughly as follows:

1. Find the character `=` in the range `offset..(offset+len)` of input and record its position in `separator`.
2. Find the space character in the range `(separator+1)...(offset+len)` of input, and record its position in `terminator`.
3. Return `key: input[offset..separator]`, `value: input[(separator+1)..terminator]`, `nextOffset: terminator+1`

The problem is that if the space is not found in step 2, terminator will be set to `input.length` - [RecordParser.sol#L34]():

```
        if (terminator == type(uint256).max) {
            terminator = input.length;
        }
```

This is incorrect because the parameters passed require data to be read within range `offset..(offset+len)` , it should not return a value beyond `offset+len` .

For example, suppose we have: `input = "...;key1=val1 key2=val2;..."` and `offset` is the start position of `key2` .

If we call `readKeyValue(input, offset, 9)` , the function will return:

```
key: "key2"
value: "val2;..."
nextOffset: input.length+1
```

The returned `value` is wrong due to the incorrect implementation of RecordParser.

The correct return should be:

```
key: "key2"
value: "val2"
nextOffset: offset+9
```

Test code for PoC:

```
diff --git a/contracts/dnsregistrar/mocks/DummyParser.sol b/cont
new file mode 100644
index 0000000..538e652
--- /dev/null
+++ b/contracts/dnsregistrar/mocks/DummyParser.sol
@@ -0,0 +1,34 @@
+pragma solidity ^0.8.4;
```

```
+
+import "../../dnssec-oracle/BytesUtils.sol";
+import "../RecordParser.sol";
+
+contract DummyParser {
+    using BytesUtils for bytes;
+
+    // parse data in format: name;key1=value1 key2=value2;url
+    function parseData(
+        bytes memory data,
+        uint256 kvCount
+    ) external pure returns (string memory name, string[] memor
+        uint256 len = data.length;
+        // retrieve name
+        uint256 sep1 = data.find(0, len, ";");
+        name = string(data.substring(0, sep1));
+
+        // retrieve url
+        uint256 sep2 = data.find(sep1 + 1, len - sep1, ";");
+        url = string(data.substring(sep2 + 1, len - sep2 - 1));
+
+        keys = new string[](kvCount);
+        values = new string[](kvCount);
+        // retrieve keys and values
+        uint256 offset = sep1 + 1;
+        for (uint256 i; i < kvCount && offset < len; i++) {
+            (bytes memory key, bytes memory val, uint256 nextOf
+            keys[i] = string(key);
+            values[i] = string(val);
+            offset = nextOffset;
+        }
+    }
+}
diff --git a/test/DummyParser.test.js b/test/DummyParser.test.js
new file mode 100644
index 0000000..396557d
--- /dev/null
+++ b/test/DummyParser.test.js
@@ -0,0 +1,27 @@
+const { expect } = require('chai')
+const { ethers } = require('hardhat')
+const { toUtf8Bytes } = require('ethers/lib/utils')
+
+describe('DummyParser', () => {
+  let parser
+
```

```
+  before(async () => {
+    const factory = await ethers.getContractFactory('DummyParse
+    parser = await factory.deploy()
+  })
+
+  it('parse data', async () => {
+    const data = "usdt;issuer=tether decimals=18;https://tether
+    const [name, keys, values, url] = await parser.parseData(tc
+    // correct name
+    expect(name).to.eq('usdt')
+    // correct keys and values
+    expect(keys[0]).to.eq('issuer')
+    expect(values[0]).to.eq('tether')
+    expect(keys[1]).to.eq('decimals')
+    // incorrect last value
+    expect(values[1]).to.eq('18;https://tether.to')
+    // correct url
+    expect(url).to.eq('https://tether.to')
+  })
+})
\ No newline at end of file
```

Tools Used

VS Code

Recommended Mitigation Steps

We should change the assignment of `terminator` so that it cannot exceed the query range:

```
diff --git a/contracts/dnsregistrar/RecordParser.sol b/contracts
index 339f213..876a87f 100644
--- a/contracts/dnsregistrar/RecordParser.sol
+++ b/contracts/dnsregistrar/RecordParser.sol
@@ -31,11 +31,12 @@ library RecordParser {
                " "
          );
          if (terminator == type(uint256).max) {
-             terminator = input.length;
+             terminator = len + offset;
+             nextOffset = terminator;
          } else {
```

```
+             nextOffset = terminator + 1;
          }
-
          key = input.substring(offset, separator - offset);
          value = input.substring(separator + 1, terminator - sep
-         nextOffset = terminator + 1;
      }
  }
```

Arachnid (ENS) acknowledged

[M-05] Unintentionally register a non-relevant DNS name owner

Submitted by SaharAP, *also found by* chaduke, nobody2018, *and* Lilyjjo

If a user proves and claims a DNS name using a wrong address format, it executes successfully without getting any error and the DNS name owner will be changed to a new unknown address.

I considered this as medium severity, as it is high impact finding with low likelihood. Cause the person who owns the new address can take control of the ENS name and transfer its ownership to another account. But because if a person finds out, she can immediately replace the correct address, the probability of such an event is low.

Proof of Concept

In the following scenario, I provided a value called `arbitrarybytes` which is 22 bytes and set it as the 'foo.test' DNS owner address. `proveAndClaim()` function will execute successfully. Finally, the owner of the ENS name would be the value set as `newOwner` which is the last 20 bytes (from the right) of the provided value in `arbitrarybytes`.

```
const arbitrarybytes= '0x9fD6E51AaD88f6f4CE6aB8827279CFFFb92
const newOwner= '0xe51Aad88f6F4CE6aB8827279cFFFB92266332265'
const proof = [
  hexEncodeSignedSet(rootKeys(expiration, inception)),
  hexEncodeSignedSet(testRrset('foo.test', arbitrarybytes)),
```

```
    ]

    await registrar.proveAndClaim(utils.hexEncodeName('foo.test'
      from: accounts[0],
    })

    assert.equal(await ens.owner(namehash.hash('foo.test')), new
```

Tools Used

vscode

Recommended Mitigation Steps

To check the validity of the owner address, the code first checks for the prefix which must be `a=0x`, and second for the length of the address which should not be less than 20 bytes or 40 characters through `hexToAddress()` function. The length of the address can also be checked to not be larger than 40 characters.

```
function hexToAddress(
        bytes memory str,
        uint256 idx,
        uint256 lastIdx
    ) internal pure returns (address, bool) {
        if (lastIdx - idx < 40) return (address(0x0), false);
        (bytes32 r, bool valid) = hexStringToBytes32(str, idx, l
        return (address(uint160(uint256(r))), valid);
    }
```

Arachnid (ENS) confirmed, but disagreed with severity and commented:

> I think this warrants a severity of 'note', as it requires the user to already supply an invalid-length address. It should revert rather than silently truncating.

LSDan (judge) commented:

> Medium risk is appropriate in this case. It is very easy to guard against the user making the mistake described.

> ref: https://github.com/code-423n4/org/issues/53#issuecomment-1340685618

[M-06] `validateSignature(...)` in `EllipticCurve` mixes up Jacobian and projective coordinates

Submitted by Holmgren*, also found by* auditor0517

Currently not exploitable because this bug is cancelled out by another issue (see my Gas report). If the other issue is fixed `validateSignature` will return completely incorrect values.

Details

In https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/EllipticCurve.sol#L415 `validateSignature` converts to affine coordinates from Jacobian coordinates, i.e. $X_a = X_j \cdot (Z_j^{-1})^2$.

However, the inputs from the previous computation https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/EllipticCurve.sol#L408 are actually projective coordinates and the correct conversion formula is $X_a = X_p \cdot Z_p^{-1}$.

This has been working so far only because the `EllipticCurve` performs a redundant chain of immediate conversions projective->affine->projective->affine and so during that last conversion $Z = 1$. Should the chain of redundant conversions be fixed, `validateSignature` will no longer work correctly.

Recommended Mitigation Steps

To just fix this bug:

```
diff --git a/contracts/dnssec-oracle/algorithms/EllipticCurve.sc
index 6861264..ea7e865 100644
--- a/contracts/dnssec-oracle/algorithms/EllipticCurve.sol
+++ b/contracts/dnssec-oracle/algorithms/EllipticCurve.sol
@@ -412,7 +412,7 @@ contract EllipticCurve {
         }
```

```
            uint256 Px = inverseMod(P[2], p);
-           Px = mulmod(P[0], mulmod(Px, Px, p), p);
+           Px = mulmod(P[0], Px, p);

            return Px % n == rs[0];
        }
```

Or to fix this bug and optimize out the redundant conversions chain:

```
diff --git a/contracts/dnssec-oracle/algorithms/EllipticCurve.sc
index 6861264..8568be2 100644
--- a/contracts/dnssec-oracle/algorithms/EllipticCurve.sol
+++ b/contracts/dnssec-oracle/algorithms/EllipticCurve.sol
@@ -405,14 +405,13 @@ contract EllipticCurve {
            uint256 sInv = inverseMod(rs[1], n);
            (x1, y1) = multiplyScalar(gx, gy, mulmod(uint256(messag
            (x2, y2) = multiplyScalar(Q[0], Q[1], mulmod(rs[0], sIn
-           uint256[3] memory P = addAndReturnProjectivePoint(x1, y
+           (uint256 Px,, uint256 Pz) = addProj(x1, y1, 1, x2, y2,

-           if (P[2] == 0) {
+           if (Pz == 0) {
                return false;
            }

-           uint256 Px = inverseMod(P[2], p);
-           Px = mulmod(P[0], mulmod(Px, Px, p), p);
+           Px = mulmod(Px, inverseMod(Pz, p), p);

            return Px % n == rs[0];
        }
```

Arachnid (ENS) confirmed

d3e4 (warden) commented:

> I agree that EllipticCurve.sol is somewhat of a bodge. It's correct what the warden says in that " `EllipticCurve` mixes up" something. But actually it adds affine points and then trivially converts them to jacobian/projective coordinates. Since they are trivial they are the same in jacobian as in projective, so one could say that it's as much a misnamed function as a confused computation.

> But it's also correct what the warden himself said that it is "currently not exploitable". In fact, I'm quite sure it's not exploitable even if "the other issue is fixed". Then it will just invalidate every signature.

> Both recommendations here are therefore just gas savings and refactoring. `mulmod(Px, Px, p)` is indeed a redundant computation, because `Px == 1` always here. But so is `inverseMod(P[2], p)` redundant. The first recommendation can be simplified even further to just

```
- uint256 Px = inverseMod(P[2], p);
- Px = mulmod(P[0], mulmod(Px, Px, p), p);

- return Px % n == rs[0];
+ return P[0] % n == rs[0];
```

> instead, since `P[2] == 1`.

> The second recommendation is a better optimisation but still does redundant conversions in `multiplyScalar`, which also should be corrected then. The whole point of using projective coordinates is to do the modular inverse (i.e. convert to affine) only at the end.

> In any case, there is nothing exploitable here, no funds or functionality are at risk. The code does what it's supposed to do as it is, just not in the prettiest way. And the way in which the hypothetical issue is proposed to arise is by fixing the very same thing that the hypothetical issue is itself based on, i.e. it is said that fixing the needless conversion between coordinate representations causes an error due to coordinate conversions. One should assume that reworking the coordinate handling would fix all coordinate issues. And if the bug described here were to arise in the suggested manner, it would be immediately noticed in testing.

> *This is a bug that is not yet a bug but could be a bug that is impossible to miss if someone were to create this bug so it's never going to actually be a bug.*

> I think this is a good catch, but the severity is only QA/Gas.

LSDan (judge) commented:

> I agree with @d3e4 that the bug is highly unlikely to make it to production without being caught. In this case, however, if it were to make it into production, the "function of the protocol or availability could be impacted". That makes this a valid medium in my view. I agree with the warden and sponsor.

[M-07] Missing recursive calls handling in `OffchainDNSResolver` CCIP-aware contract

Submitted by MalfurionWhitehat

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/OffchainDNSResolver.sol#L109-L113
https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/OffchainDNSResolver.sol#L119

The `resolveCallback` function from `OffchainDNSResolver` is used as part of the EIP-3668 standard to properly resolve DNS names using an off-chain gateway and validating RRsets against the DNSSEC oracle.

The issue is that the function lacks proper error handling, specifically, a try/catch block to properly bubble up `OffchainLookup` error from the `dnsresolver` extracted from the RRset. As the EIP specifies,

> *When a CCIP-aware contract wishes to make a call to another contract, and the possibility exists that the callee may implement CCIP read, the calling contract MUST catch all `OffchainLookup` errors thrown by the callee, and revert with a different error if the `sender` field of the error does not match the callee address. [...]*
> *Where the possibility exists that a callee implements CCIP read, a CCIP-aware contract MUST NOT allow the default solidity behaviour of bubbling up reverts from nested calls.*

Impact

As per the EIP, the result would be an OffchainLookup that looks valid to the client, as the sender field matches the address of the contract that was called, but does not execute correctly.

Proof of Concept

1. Client calls `OffchainDNSResolver.resolve`, which reverts with `OffchainLookup`, and prompts the client to execute `resolveCallback` after having fetched the necessary data from the `gatewayURL`
2. The RRset returned by the gateway contains a `dnsresolver` that is a CCIP-aware contract, and also supports the `IExtendedDNSResolver.resolve.selector` interface
3. Calling `IExtendedDNSResolver(dnsresolver).resolve(name,query,context);` could trigger another `OffchainLookup` error, but with a `sender` that does not match the `dnsresolver`, which would be just returned to the client without any modifications
4. As a result, the `sender` field would be incorrect as per the EIP

Recommended Mitigation Steps

Use the recommended example from the EIP in order to support nested lookups.

Arachnid (ENS) confirmed

Low Risk and Non-Critical Issues

For this audit, 39 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **Sathish9098** received the top score from the judge.

The following wardens also submitted reports: brgltd, Shubham, Udsen, Josiah, pontifex, lukris02, tnevler, eierina, SAAJ, ArbitraryExecution, urataps, favelanky, auditor0517, Dyear, 0xAgro, Aymen0909, schrodinger, BRONZEDISC, Jerry0x, Recep, IceBear, j4ld1na, naman1778, catellatech, 0x73696d616f, 0xTheC0der, MohammedRizwan, ABA, MalfurionWhitehat, matrix_0wl, RaymondFam, codeslide, Bauchibred, Rickard, 0xSmartContract, Jorgect, chaduke, *and* Eurovickk.

LOW FINDINGS

COUNT	ISSUES	INSTANCES
[L-01]	Use abi.encode to convert safest way from uint values to bytes	3
[L-02]	Loss of precision due to rounding	1
[L-03]	Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows when casting from uint256	11
[L-04]	Lack of Sanity/Threshold/Limit Checks	2
[L-05]	Function Calls in Loop Could Lead to Denial of Service	10
[L-06]	Project Upgrade and Stop Scenario should be	-
[L-07]	Front running attacks by the onlyOwner	3
[L-08]	Use BytesLib.sol library to safely covert bytes to uint256	2
[L-09]	In the constructor, there is no return of incorrect address identification	6
[L-10]	Even with the onlyOwner or `owner_only` modifier, it is best practice to use the re-entrancy pattern	3

NON-CRITICAL FINDINGS

COUNT	ISSUES	INSTANCES
[N-01]	Avoid infinite loops whenever possible	2
[N-02]	immutable should be uppercase	4
[N-03]	For functions, follow Solidity standard naming conventions (internal function style rule)	28
[N-04]	Need Fuzzing test for unchecked	5
[N-05]	Remove commented out code	-
[N-06]	Inconsistent method of specifying a floating pragma	8

COUNT	ISSUES	INSTANCES
[N-07]	NO SAME VALUE INPUT CONTROL	1
[N-08]	Constant redefined elsewhere	-
[N-09]	According to the syntax rules, use => mapping (instead of => mapping(using spaces as keyword	3
[N-10]	Use SMTChecker	-
[N-11]	Assembly Codes Specific — Should Have Comments	18
[N-12]	Take advantage of Custom Error's return value property	4
[N-13]	Use constants instead of using numbers directly without explanations	6
[N-14]	Shorthand way to write if / else statement	4
[N-15]	Don't use named return variables its confusing	8
[N-16]	Constants should be in uppercase	7
[N-17]	TYPOS	4
[N-18]	Use named parameters for mapping type declarations	3
[N-19]	File does not contain an SPDX Identifier	4
[N-20]	Declaration shadows an existing declaration	-
[N-21]	Event is missing indexed fields	2

[L-01] Use `abi.encode` to convert safest way from uint values to bytes

DRAWBACKS

Now the protocol uses direct conversion way this is not safe. bytes(value) to convert a uint to bytes is not considered a safe way because it creates an uninitialized byte array of length. This means that the contents of the byte array are undefined and may contain sensitive data from previous memory usage, which could result in security vulnerabilities.

BENIFITS of `abi.encode`

In Solidity, the safest way to convert a uint to bytes is to use the abi.encode function. This function will encode the uint as a byte array using the ABI encoding rules, which ensures that the output is a deterministic and standardized representation of the uint value.

```
FILE: 2023-04-ens/contracts/utils/NameEncoder.sol

27:  dnsName[i + 1] = bytes1(labelLength);
49:  dnsName[0] = bytes1(labelLength);
```

NameEncoder.sol#L27

```
FILE: 2023-04-ens/contracts/dnssec-oracle/BytesUtils.sol

376: return bytes32(ret << (256 - bitlen));
```

BytesUtils.sol#L376

[L-02] Loss of precision due to rounding

```
FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/EllipticCur

52: q = r1 / r2;
```

EllipticCurve.sol#L52

[L-03] Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows when casting from uint256

Using the SafeCast library can help prevent unexpected errors in your Solidity code and make your contracts more secure

```
FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol

160: if (!RRUtils.serialNumberGte(rrset.expiration, uint32(now))
```

```
161: revert SignatureExpired(rrset.expiration, uint32(now));
166: if (!RRUtils.serialNumberGte(uint32(now), rrset.inception))
167: revert SignatureNotValidYet(rrset.inception, uint32(now));
```

[DNSSECImpl.sol#L160]

```
FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol

430: return uint16(ac1);
```

[RRUtils.sol#L430]

> When ever we convert int256 to uint256 or uint256 to int256 we should use OpenZeppelin's safecast to avoid unexpected errors

```
FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/EllipticCur

56: if (t1 < 0) return (m - uint256(-t1));
58: return uint256(t1);
```

[EllipticCurve.sol#L56-L58]

```
FILE: 2023-04-ens/contracts/dnssec-oracle/BytesUtils.sol

92: int256 diff = int256(a & mask) - int256(b & mask);
99: return int256(len) - int256(otherlen);
183: return uint8(self[idx]);
344: decoded = uint8(base32HexTable[uint256(uint8(char)) - 0x30]
```

[BytesUtils.sol#L92]

Recommended Mitigation Steps:

Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows when casting from uint256.

[L-04] Lack of Sanity/Threshold/Limit Checks

Devoid of sanity/threshold/limit checks, critical parameters can be configured to invalid values, causing a variety of issues and breaking expected interactions within/between contracts. Consider adding proper uint256 validation as well as zero address checks for critical changes. A worst case scenario would render the contract needing to be re-deployed in the event of human/accidental errors that involve value assignments to immutable variables. If the validation procedure is unclear or too complex to implement on-chain, document the potential issues that could produce invalid values

```
FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol

64: function setAlgorithm(uint8 id, Algorithm algo) public owner
        algorithms[id] = algo;
        emit AlgorithmUpdated(id, address(algo));
    }

75: function setDigest(uint8 id, Digest digest) public owner_onl
        digests[id] = digest;
        emit DigestUpdated(id, address(digest));
    }
```

[DNSSECImpl.sol#L64-L78]

[L-05] Function Calls in Loop Could Lead to Denial of Service

Function calls made in unbounded loop are error-prone with potential resource exhaustion as it can trap the contract due to the gas limitations or failed transactions

```
FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol

while (counts > othercounts) {
            prevoff = off;
            off = progress(self, off);
            counts--;
        }

while (othercounts > counts) {
            otherprevoff = otheroff;
            otheroff = progress(other, otheroff);
```

```
            othercounts--;
        }

        // Compare the last nonequal labels to each other
        while (counts > 0 && !self.equals(off, other, otheroff)) {
            prevoff = off;
            off = progress(self, off);
            otherprevoff = otheroff;
            otheroff = progress(other, otheroff);
            counts -= 1;
        }
```

[RRUtils.sol#L291-L295]

```
FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol

118: for (uint256 i = 0; i < input.length; i++) {
            RRUtils.SignedSet memory rrset = validateSignedSet(
                input[i],
                proof,
                now
            );
            proof = rrset.data;
            inception = rrset.inception;
        }

260: for (; !proof.done(); proof.next()) {
            bytes memory proofName = proof.name();
            if (!proofName.equals(rrset.signerName)) {
                revert ProofNameMismatch(rrset.signerName, proof
            }

            bytes memory keyrdata = proof.rdata();
            RRUtils.DNSKEY memory dnskey = keyrdata.readDNSKEY(
                0,
                keyrdata.length
            );
            if (verifySignatureWithKey(dnskey, keyrdata, rrset,
                return;
            }
```

[DNSSECImpl.sol#L118-L126]

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/DNSSECImpl.sol#L336-L360

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/DNSSECImpl.sol#L380-L404

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/EllipticCurve.sol#L325-L327

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/EllipticCurve.sol#L361-L362

[L-06] Project Upgrade and Stop Scenario should be

At the start of the project, the system may need to be stopped or upgraded, I suggest you have a script beforehand and add it to the documentation. This can also be called an " EMERGENCY STOP (CIRCUIT BREAKER) PATTERN ".

https://github.com/maxwoe/solidity_patterns/blob/master/security/EmergencyStop.sol

[L-07] Front running attacks by the onlyOwner

```
FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

80: function setPublicSuffixList(PublicSuffixList _suffixes) pub
        suffixes = _suffixes;
        emit NewPublicSuffixList(address(suffixes));
    }
```

DNSRegistrar.sol#L80-L83

```
FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol
```

```
64: function setAlgorithm(uint8 id, Algorithm algo) public owner
        algorithms[id] = algo;
        emit AlgorithmUpdated(id, address(algo));
    }

75: function setDigest(uint8 id, Digest digest) public owner_onl
        digests[id] = digest;
        emit DigestUpdated(id, address(digest));
    }
```

DNSSECImpl.sol#L64-L67

[L-08] Use BytesLib.sol library to safely covert bytes to uint256

Use toUint256() safely convert bytes to uint256 instead of plain way of conversion

```
FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/P256SHA256A

34: return [uint256(data.readBytes32(0)), uint256(data.readBytes
41: return [uint256(data.readBytes32(4)), uint256(data.readBytes
```

P256SHA256Algorithm.sol#L34

[L-09] In the constructor, there is no return of incorrect address identification

In case of incorrect address definition in the constructor , there is no way to fix it because of the variables are immutable.

```
FILE: 2023-04-ens/contracts/dnsregistrar/OffchainDNSResolver.sol

44: ens = _ens;
45: oracle = _oracle;
```

OffchainDNSResolver.sol#L44-L45

```
FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

62: previousRegistrar = _previousRegistrar;
63: resolver = _resolver;
64: oracle = _dnssec;
67: ens = _ens;
```

DNSRegistrar.sol#L63-L65

Recommended Mitigations:

```
require(_ens!=address(0), " zero address");
```

[L-10] Even with the `onlyOwner` or `owner_only` modifier, it is best practice to use the re-entrancy pattern

It's still good practice to apply the reentry model as a precautionary measure in case the code is changed in the future to remove the onlyOwner modifier or the contract is used as a base contract for other contracts.

Using the reentry modifier provides an additional layer of security and ensures that your code is protected from potential reentry attacks regardless of any other security measures you take.

So even if you followed the "check-effects-interactions" pattern correctly, it's still considered good practice to use the reentry modifier

```
FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

80: function setPublicSuffixList(PublicSuffixList _suffixes) pub
        suffixes = _suffixes;
        emit NewPublicSuffixList(address(suffixes));
    }
```

DNSRegistrar.sol#L80-L83

```
FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol
```

```
64: function setAlgorithm(uint8 id, Algorithm algo) public owner
        algorithms[id] = algo;
        emit AlgorithmUpdated(id, address(algo));
    }

75: function setDigest(uint8 id, Digest digest) public owner_onl
        digests[id] = digest;
        emit DigestUpdated(id, address(digest));
    }
```

[DNSSECImpl.sol#L64-L67]

Recommended Mitigation

```
  modifier noReentrant() {
        require(!locked, "Reentrant call");
        locked = true;
        _;
        locked = false;
    }

function setPublicSuffixList(PublicSuffixList _suffixes) public
```

[N-01] Avoid infinite loops whenever possible

```
FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/EllipticCur

> This r2 != 0 condition can be always true cause unbounded loop

51: while (r2 != 0) {
                q = r1 / r2;
                (t1, t2, r1, r2) = (t2, t1 - int256(q) * t2, r2,
            }
```

[EllipticCurve.sol#L51-L54]

```
FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol
```

```
    while (true) {
                assert(idx < self.length);
                uint256 labelLen = self.readUint8(idx);
                idx += labelLen + 1;
                if (labelLen == 0) {
                    break;
                }
            }
```

RRUtils.sol#L24-L31

[N-02] immutable should be uppercase

```
FILE : 2023-04-ens/contracts/dnsregistrar/OffchainDNSResolver.sc

37: ENS public immutable ens;
38: DNSSEC public immutable oracle;
```

OffchainDNSResolver.sol#L37-L38

Recommended Mitigation

```
- 38: DNSSEC public immutable oracle;
+ 38: DNSSEC public immutable ORACLE;
```

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnsregistrar/DNSRegistrar.sol#L26-L27

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnsregistrar/DNSRegistrar.sol#L29-L30

[N-03] For functions, follow Solidity standard naming conventions (internal function style rule)

Description

The above codes don't follow Solidity's standard naming convention,

internal and private functions : the mixedCase format starting with an underscore (`_mixedCase` starting with an underscore)

```
File: 2023-04-ens/contracts/dnsregistrar/OffchainDNSResolver.sol

136: function parseRR(
         bytes memory data,
         uint256 idx,
         uint256 lastIdx
     ) internal view returns (address, bytes memory) {

162: function readTXT(
         bytes memory data,
         uint256 startIdx,
         uint256 lastIdx
     ) internal pure returns (bytes memory) {

173: function parseAndResolve(
         bytes memory nameOrAddress,
         uint256 idx,
         uint256 lastIdx
     ) internal view returns (address) {

190: function resolveName(
         bytes memory name,
         uint256 idx,
         uint256 lastIdx
     ) internal view returns (address) {

209: function textNamehash(
         bytes memory name,
         uint256 idx,
         uint256 lastIdx
     ) internal view returns (bytes32) {
```

[OffchainDNSResolver.sol#L136-L140]

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/RSAVerify.sol#L14-L18

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/ModexpPrecompile.sol#L7-L11

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/P256SHA256Algorithm.sol#L30-L32

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/P256SHA256Algorithm.sol#L37-L39

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/utils/NameEncoder.sol#L9-L11

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/utils/HexUtils.sol#L11-L15

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/utils/HexUtils.sol#L68-L72

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnsregistrar/DNSClaimChecker.sol#L19-L22

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnsregistrar/DNSClaimChecker.sol#L46-L50

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnsregistrar/DNSClaimChecker.sol#L66-L70

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L13-L17

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L32-L35

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L52-L59

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L111-L117

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L129-L134

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L148-L152

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L164-L167

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L179-L182

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L332-L336

[N-04] Need Fuzzing test for unchecked

```
FILE: 2023-04-ens/contracts/utils/NameEncoder.sol

24:  unchecked {

FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol

380:   unchecked {
336:   unchecked {

FILE: 2023-04-ens/contracts/dnssec-oracle/BytesUtils.sol

284:  unchecked {

FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/EllipticCur

41:  unchecked {
```

[N-05] Remove commented out code

```
FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol

358: *     function computeKeytag(bytes memory data) internal pu
359: *         uint ac;
360: *         for (uint i = 0; i < data.length; i++) {
361: *             ac += i & 1 == 0 ? uint16(data.readUint8(i))
362: *         }
363: *         return uint16(ac + (ac >> 16));
364: *     }
```

[RRUtils.sol#L358-L364]

[N-06] Inconsistent method of specifying a floating pragma

Some files use `>=`, some use `^`. The instances below are examples of the method that has the fewest instances for a specific version. Note that using `>=` without also specifying `<=` will lead to failures to compile, or external project incompatability, when the major version changes and there are breaking-changes, so `^` should be preferred regardless of the instance counts

```
FILE: 2023-04-ens/contracts/dnsregistrar/OffchainDNSResolver.sol

2: pragma solidity ^0.8.4;

FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol

1: pragma solidity ^0.8.4;

FILE: 2023-04-ens/contracts/dnssec-oracle/SHA1.sol

1: pragma solidity >=0.8.4;

FILE: 2023-04-ens/contracts/dnsregistrar/DNSClaimChecker.sol

2: pragma solidity ^0.8.4;

FILE: 2023-04-ens/contracts/dnsregistrar/RecordParser.sol

2: pragma solidity ^0.8.11;

FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/ModexpPrecc

1: pragma solidity ^0.8.4;

FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/RSAVerify.s

1: pragma solidity ^0.8.4;
```

[N-07] NO SAME VALUE INPUT CONTROL

```
FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

80: function setPublicSuffixList(PublicSuffixList _suffixes) pub
        suffixes = _suffixes;
        emit NewPublicSuffixList(address(suffixes));
    }
```

DNSRegistrar.sol#L80-L83

[N-08] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated.

A cheap way to store constants in a single location is to create an internal constant in a library. If the variable is a local cache of another contract's value, consider making the cache variable internal or private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

```
FILE: 2023-04-ens/contracts/dnsregistrar/OffchainDNSResolver.sol

29: uint16 constant CLASS_INET = 1;
30: uint16 constant TYPE_TXT = 16;

FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol

72: uint256 constant RRSIG_TYPE = 0;
73: uint256 constant RRSIG_ALGORITHM = 2;
74: uint256 constant RRSIG_LABELS = 3;
75: uint256 constant RRSIG_TTL = 4;
76: uint256 constant RRSIG_EXPIRATION = 8;
77: uint256 constant RRSIG_INCEPTION = 12;
78: uint256 constant RRSIG_KEY_TAG = 16;
79: uint256 constant RRSIG_SIGNER_NAME = 18;
210: uint256 constant DNSKEY_FLAGS = 0;
211: uint256 constant DNSKEY_PROTOCOL = 2;
212: uint256 constant DNSKEY_ALGORITHM = 3;
213: uint256 constant DNSKEY_PUBKEY = 4;

236: uint256 constant DS_KEY_TAG = 0;
237: uint256 constant DS_ALGORITHM = 2;
238: uint256 constant DS_DIGEST_TYPE = 3;
239: uint256 constant DS_DIGEST = 4;

FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/EllipticCur

21: uint256 constant a =
        0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFF
23: uint256 constant b =
        0x5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C
25: uint256 constant gx =
        0x6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A139
27: uint256 constant gy =
        0x4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB640
29: uint256 constant p =
```

```
        0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFF
30: uint256 constant n =
        0xFFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CA

32: uint256 constant lowSmax =
        0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F
```

FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol

```
27: uint16 constant DNSCLASS_IN = 1;
29: uint16 constant DNSTYPE_DS = 43;
30: uint16 constant DNSTYPE_DNSKEY = 48;
32: uint256 constant DNSKEY_FLAG_ZONEKEY = 0x100;
```

[N-09] According to the syntax rules, use => mapping (instead of => mapping(using spaces as keyword

FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol

```
45: mapping(uint8 => Algorithm) public algorithms;
46: mapping(uint8 => Digest) public digests;
```

DNSSECImpl.sol#L45-L46

FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

```
32: mapping(bytes32 => uint32) public inceptions;
```

DNSRegistrar.sol#L32

[N-10] Use SMTChecker

The highest tier of smart contract behavior assurance is formal mathematical verification. All assertions that are made are guaranteed to be true across all inputs → The quality of your asserts is the quality of your verification

https://twitter.com/0xOwenThurm/status/1614359896350425088?t=dbG9gHFigBX85Rv29lOjlQ&s=19

[N-11] Assembly Codes Specific — Should Have Comments

Since this is a low level language that is more difficult to parse by readers, include extensive documentation, comments on the rationale behind its use, clearly explaining what each assembly instruction does.

This will make it easier for users to trust the code, for reviewers to validate the code, and for developers to build on or update the code.

Note that using Assembly removes several important security features of Solidity, which can make the code more insecure and more error-prone.

```
FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/ModexpPrecc

23: assembly {
```

ModexpPrecompile.sol#L23

```
FILE: 2023-04-ens/contracts/utils/HexUtils.sol

17:  assembly {
```

HexUtils.sol#L17

```
FILE: FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol

386: assembly {
```

RRUtils.sol#L386

```
FILE: 2023-04-ens/contracts/dnssec-oracle/SHA1.sol

7: assembly {
```

SHA1.sol#L7

```
FILE: 2023-04-ens/contracts/dnssec-oracle/BytesUtils.sol

19:  assembly {
73:  assembly {
80:  assembly {
197: assembly {
213: assembly {
229: assembly {
245: assembly {
267: assembly {
276: assembly {
286: assembly {
311: assembly {
```

[BytesUtils.sol#L19]

[N-12] Take advantage of Custom Error's return value property

An important feature of Custom Error is that values such as address, tokenID, msg.value can be written inside the () sign, this kind of approach provides a serious advantage in debugging and examining the revert details of dapps such as tenderly

```
2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

34: error NoOwnerRecordFound();
35: error PreconditionNotMet();
36: error StaleProof();
```

[DNSRegistrar.sol#L36-L37]

```
FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol

38:  error InvalidRRSet();
```

[DNSSECImpl.sol#L38]

[N-13] Use constants instead of using numbers directly without explanations

```
FILE: 2023-04-ens/contracts/dnsregistrar/OffchainDNSResolver.sol

144: if (txt.length < 5 || !txt.equals(0, "ENS1 ", 0, 5)) {
149: uint256 lastTxtIdx = txt.find(5, txt.length - 5, " ");
151: address dnsResolver = parseAndResolve(txt, 5, txt.length);
154: address dnsResolver = parseAndResolve(txt, 5, lastTxtIdx);
```

OffchainDNSResolver.sol#L144

```
FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/P256SHA256A

33: require(data.length == 64, "Invalid p256 signature length");
40: require(data.length == 68, "Invalid p256 key length");
```

P256SHA256Algorithm.sol#L33

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/utils/HexUtils.sol#L25-L36

[N-14] Shorthand way to write if / else statement

The normal if / else statement can be refactored in a shorthand way to write it:

Increases readability
Shortens the overall SLOC

```
FILE: 2023-04-ens/contracts/dnsregistrar/OffchainDNSResolver.sol

if (separator < lastIdx) {
            parentNode = textNamehash(name, separator + 1, lastI
        } else {
            separator = lastIdx;
        }
```

[OffchainDNSResolver.sol#L216-L220]

```
FILE : 2023-04-ens/contracts/dnssec-oracle/BytesUtils.sol

if (shortest - idx >= 32) {
                    mask = type(uint256).max;
                } else {
                    mask = ~(2 ** (8 * (idx + 32 - shortest)) -
                }
```

[BytesUtils.sol#L87-L91]

```
FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/EllipticCur

221: if (isZeroCurve(x0, y0)) {
            return (x1, y1, z1);
        } else if (isZeroCurve(x1, y1)) {
            return (x0, y0, z0);
        }

234: if (t0 == t1) {
                return twiceProj(x0, y0, z0);
            } else {
                return zeroProj();
            }
```

[EllipticCurve.sol#L221-L225]

Recommended Mitigation

```
if (separator < lastIdx) {
            parentNode = textNamehash(name, separator + 1, lastI
        } else {
            separator = lastIdx;
        }


separator < lastIdx ? parentNode = textNamehash(name, separator
```

[N-15] Don't use named return variables, it's confusing

```
FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

113: function _claim(
        bytes memory name,
        DNSSEC.RRSetWithSignature[] memory input
    ) internal returns (bytes32 parentNode, bytes32 labelHash, a

166: function enableNode(bytes memory domain) public returns (by

174: function _enableNode(
        bytes memory domain,
        uint256 offset
    ) internal returns (bytes32 node) {
```

DNSRegistrar.sol#L133-L136

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnsregistrar/RecordParser.sol#L14-L21

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/ModexpPrecompile.sol#L7-L11

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/utils/HexUtils.sol#L11-L15

[N-16] Constants should be in uppercase

```
FILE: 2023-04-ens/contracts/dnssec-oracle/algorithms/EllipticCur

21: uint256 constant a =
        0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFF
23: uint256 constant b =
        0x5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C
```

```
25: uint256 constant gx =
        0x6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13
27: uint256 constant gy =
        0x4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB64
29: uint256 constant p =
        0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFF
30: uint256 constant n =
        0xFFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CA

32: uint256 constant lowSmax =
        0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F
```

EllipticCurve.sol#L21-L35

[N-17] TYPOS

```
FILE: 2023-04-ens/contracts/dnssec-oracle/BytesUtils.sol

/// @audit codepoints => code points

- 43: *      on unicode codepoints.
+ 43: *      on unicode code points.

FILE: 2023-04-ens/contracts/dnssec-oracle/RRUtils.sol

/// @audit bitshifting=> bit shifting
/// @audit Naive => Native

- 356: * from the input string, with some mild bitshifting. Here
- 356: * from the input string, with some mild bit shifting. Her

FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol

/// @audit canonicalised => MEANING LESS WORD

135: *        data, followed by a series of canonicalised RR rec
```

[N-18] Use named parameters for mapping type declarations

Consider using named parameters in mappings (e.g. `mapping(address account => uint256 balance)`) to improve readability. This feature is present since Solidity

0.8.18.

```
FILE: 2023-04-ens/contracts/dnssec-oracle/DNSSECImpl.sol

45: mapping(uint8 => Algorithm) public algorithms;
46: mapping(uint8 => Digest) public digests;
```

DNSSECImpl.sol#L45-L46

```
FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

32: mapping(bytes32 => uint32) public inceptions;
```

DNSRegistrar.sol#L32

[N-19] File does not contain an SPDX Identifier

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/SHA1.sol#L1-L3

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/BytesUtils.sol#L1-L3

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/RRUtils.sol#L1-L9

https://github.com/code-423n4/2023-04-ens/blob/45ea10bacb2a398e14d711fe28d1738271cd7640/contracts/dnssec-oracle/algorithms/EllipticCurve.sol#L1-L19

[N-20] Declaration shadows an existing declaration

```
FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol
```

```
30:   address public immutable resolver;
104:  address resolver,
159:  function resolver(

192: address owner,
143: function owner(
```

[N-21] Event is missing indexed fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (threefields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

```
FILE: 2023-04-ens/contracts/dnsregistrar/DNSRegistrar.sol

47: event Claim(
        bytes32 indexed node,
        address indexed owner,
        bytes dnsname,
        uint32 inception
    );
53: event NewPublicSuffixList(address suffixes);
```

DNSRegistrar.sol#L47-L53

Gas Optimizations

For this audit, 8 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **JCN** received the top score from the judge.

The following wardens also submitted reports: d3e4, niser93, descharre, openwide, naman1778, Sathish9098, *and* saneryee.

Summary

A majority of the optimizations were benchmarked via the protocol's tests, i.e. using the following config: `solc version 0.8.17`, `optimizer on`, and `1300 runs`. Optimizations that were not benchmarked are explained via EVM gas costs and opcodes.

Below are the overall average gas savings for the following tested functions, with all the optimizations applied: | Function | Before | After | Avg Gas Savings | | ------ | ------ | ------- | ------- | | DNSRegistrar.proveAndClaim | 269704 | 240737 | 28967 | | DNSRegistrar.proveAndClaimWithResolver | 320432 | 291386 | 29046 |

Total gas saved across all listed functions: 58013

Notes:

- The Gas report output, after all optimizations have been applied, can be found at the end of the report.
- The final diffs for each contract, with all the optimizations applied, can be found here.

Table of Contents

Number	Issue	Instances
[G-10]	Include check in assembly block	3
[G-11]	Write a more gas efficient assembly loop	1
[G-12]	Use a `do while` loop instead of a `for` loop	8
[G-13]	Don't cache value if it is only used once	1
[G-14]	Refactor code to avoid instantiating memory struct within loop	1
[G-15]	`If` statements that use `&` can be refactored into nested `if` statements	1
[G-16]	Refactor modifier to avoid two external calls when calling `setPublicSuffixList`	1

[G-01] Refactor code to avoid unnecessary memory expansion and data check within loops

In the instances below, the proof name is being read from memory and then stored into a new section of memory. After this is done an `if` statement is used to check a condition. Both the proof name and condition in the `if` statement stay the same for each iteration, and therefore those lines of code can be moved outside of the loop to avoid doing those computations on each iteration.

Total Instances: `2`

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L254-L264

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 3270 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	298194	336129	317162	2

```
File: contracts/dnsssec-oracle/DNSSECImpl.sol
254:     function verifyWithKnownKey(
255:         RRUtils.SignedSet memory rrset,
```

```
256:         RRSetWithSignature memory data,
257:         RRUtils.RRIterator memory proof
258:     ) internal view {
259:         // Check the DNSKEY's owner name matches the signer
260:         for (; !proof.done(); proof.next()) {
261:             bytes memory proofName = proof.name();
262:             if (!proofName.equals(rrset.signerName)) {
263:                 revert ProofNameMismatch(rrset.signerName, p
264:             }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..f8adb3c 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -257,11 +257,11 @@ contract DNSSECImpl is DNSSEC, Owned {
         RRUtils.RRIterator memory proof
     ) internal view {
         // Check the DNSKEY's owner name matches the signer nam
+        bytes memory proofName = proof.name();
+        if (!proofName.equals(rrset.signerName)) {
+            revert ProofNameMismatch(rrset.signerName, proofNam
+        }
         for (; !proof.done(); proof.next()) {
-            bytes memory proofName = proof.name();
-            if (!proofName.equals(rrset.signerName)) {
-                revert ProofNameMismatch(rrset.signerName, proc
-            }

             bytes memory keyrdata = proof.rdata();
             RRUtils.DNSKEY memory dnskey = keyrdata.readDNSKEY(
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L373-L384

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 3223 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	298241	336176	317209	2

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
373:     function verifyKeyWithDS(
374:         bytes memory keyname,
375:         RRUtils.RRIterator memory dsrrs,
376:         RRUtils.DNSKEY memory dnskey,
377:         bytes memory keyrdata
378:     ) internal view returns (bool) {
379:         uint16 keytag = keyrdata.computeKeytag();
380:         for (; !dsrrs.done(); dsrrs.next()) {
381:             bytes memory proofName = dsrrs.name();
382:             if (!proofName.equals(keyname)) {
383:                 revert ProofNameMismatch(keyname, proofName)
384:             }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..ae9ba6a 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -377,11 +377,11 @@ contract DNSSECImpl is DNSSEC, Owned {
         bytes memory keyrdata
     ) internal view returns (bool) {
         uint16 keytag = keyrdata.computeKeytag();
+        bytes memory proofName = dsrrs.name();
+        if (!proofName.equals(keyname)) {
+            revert ProofNameMismatch(keyname, proofName);
+        }
         for (; !dsrrs.done(); dsrrs.next()) {
-            bytes memory proofName = dsrrs.name();
-            if (!proofName.equals(keyname)) {
-                revert ProofNameMismatch(keyname, proofName);
-            }

             RRUtils.DS memory ds = dsrrs.data.readDS(
                 dsrrs.rdataOffset,
```

[G-02] State variables can be cached instead of re-reading them from storage

Caching of a state variable replaces each `Gwarmaccess (100 gas)` with a much cheaper stack read.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L415-L425

Gas Savings for `DNSRegistrar.proveAndClaim`*, obtained via protocol's tests: Avg 182 gas*

	Min	Max	Avg	# calls
Before	198110	308834	269704	7
After	197928	308652	269522	7

Cache `digests[digesttype]` to save 1 SLOAD

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
415:     function verifyDSHash(
416:         uint8 digesttype,
417:         bytes memory data,
418:         bytes memory digest
419:     ) internal view returns (bool) {
420:         if (address(digests[digesttype]) == address(0)) {
421:             return false;
422:         }
423:         return digests[digesttype].verify(data, digest);
424:     }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..df0bbf7 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -417,9 +417,11 @@ contract DNSSECImpl is DNSSEC, Owned {
         bytes memory data,
         bytes memory digest
     ) internal view returns (bool) {
-        if (address(digests[digesttype]) == address(0)) {
+        Digest _digest = digests[digesttype];
+        if (address(_digest) == address(0)) {
             return false;
         }
-        return digests[digesttype].verify(data, digest);
+        return _digest.verify(data, digest);
```

```
        }
    }
```

[G-03] Create immutable variable to avoid an external call

Instead of performing an external call to get the `root` address each time `_enableNode` is invoked, we can perform this external call once in the constructor and store the `root` as an immutable variable. Doing this will save 1 external call each time `_enableNode` is invoked.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSRegistrar.sol#L187-L192

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 1011 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	300453	338388	319421	2

```
File: contracts/dnsregistrar/DNSRegistrar.sol
187:         if (owner == address(0) || owner == previousRegistra
188:             if (parentNode == bytes32(0)) {
189:                 Root root = Root(ens.owner(bytes32(0)));
190:                 root.setSubnodeOwner(label, address(this));
191:                 ens.setResolver(node, resolver);
192:             } else {
```

```
diff --git a/contracts/dnsregistrar/DNSRegistrar.sol b/contracts
index 953a9a3..fda3ebc 100644
--- a/contracts/dnsregistrar/DNSRegistrar.sol
+++ b/contracts/dnsregistrar/DNSRegistrar.sol
@@ -28,6 +28,7 @@ contract DNSRegistrar is IDNSRegistrar, IERC16
     PublicSuffixList public suffixes;
     address public immutable previousRegistrar;
     address public immutable resolver;
+    Root private immutable root;
```

```
     // A mapping of the most recent signatures seen for each cl
     mapping(bytes32 => uint32) public inceptions;

@@ -65,6 +66,7 @@ contract DNSRegistrar is IDNSRegistrar, IERC16
         suffixes = _suffixes;
         emit NewPublicSuffixList(address(suffixes));
         ens = _ens;
+        root = Root(_ens.owner(bytes32(0)));
     }

     /**
@@ -186,7 +188,6 @@ contract DNSRegistrar is IDNSRegistrar, IERC
         address owner = ens.owner(node);
         if (owner == address(0) || owner == previousRegistrar)
             if (parentNode == bytes32(0)) {
-                Root root = Root(ens.owner(bytes32(0)));
                 root.setSubnodeOwner(label, address(this));
                 ens.setResolver(node, resolver);
             } else {
```

[G-04] Avoid emitting storage values

In the instance below, we can emit the calldata value instead of emitting a storage value. This will result in using a cheap `CALLDATALOAD` instead of an expensive `SLOAD`.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSRegistrar.sol#L80-L83

Emit `_suffixes` instead of reading from storage

```
File: contracts/dnsregistrar/DNSRegistrar.sol
80:     function setPublicSuffixList(PublicSuffixList _suffixes)
81:         suffixes = _suffixes;
82:         emit NewPublicSuffixList(address(suffixes));
83:     }
```

```
diff --git a/contracts/dnsregistrar/DNSRegistrar.sol b/contracts
index 953a9a3..64e758f 100644
--- a/contracts/dnsregistrar/DNSRegistrar.sol
```

```
+++ b/contracts/dnsregistrar/DNSRegistrar.sol
@@ -79,7 +79,7 @@ contract DNSRegistrar is IDNSRegistrar, IERC16

     function setPublicSuffixList(PublicSuffixList _suffixes) pu
         suffixes = _suffixes;
-        emit NewPublicSuffixList(address(suffixes));
+        emit NewPublicSuffixList(address(_suffixes));
     }
```

[G-05] Refactor code with assembly to check zero address, hash values, and perform external call

In the instance below, we can check for the zero address using assembly. In addition, we can reuse memory to hash values and perform an external call.

Note: In order to do this optimization safely we will cache and restore the free memory pointer.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSRegistrar.sol#L115-L122

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 169 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301449	339077	320263	2

```
File: contracts/dnsregistrar/DNSRegistrar.sol
115:         if (addr != address(0)) {
116:             if (resolver == address(0)) {
117:                 revert PreconditionNotMet();
118:             }
119:             bytes32 node = keccak256(abi.encodePacked(rootNo
120:             // Set the resolver record
121:             AddrResolver(resolver).setAddr(node, addr);
122:         }
```

```
diff --git a/contracts/dnsregistrar/DNSRegistrar.sol b/contracts
index 953a9a3..a2802fd 100644
--- a/contracts/dnsregistrar/DNSRegistrar.sol
+++ b/contracts/dnsregistrar/DNSRegistrar.sol
@@ -112,13 +112,25 @@ contract DNSRegistrar is IDNSRegistrar, IE
             revert PermissionDenied(msg.sender, owner);
         }
         ens.setSubnodeRecord(rootNode, labelHash, owner, resolv
-        if (addr != address(0)) {
-            if (resolver == address(0)) {
-                revert PreconditionNotMet();
+        assembly {
+            if iszero(iszero(calldataload(0x64))) {
+                if iszero(calldataload(0x44)) {
+                    mstore(0x00, 0xf1613c4c)
+                    revert(0x1c, 0x04)
+                }
+                let memptr := mload(0x40)
+                mstore(0x00, rootNode)
+                mstore(0x20, labelHash)
+                let node := keccak256(0x00, 0x40)
+                mstore(0x00, 0xd5fa2b00)
+                mstore(0x20, node)
+                mstore(0x40, calldataload(0x64))
+                let success := call(gas(), calldataload(0x44),
+                if iszero(success) {
+                    revert(0, 0)
+                }
+                mstore(0x40, memptr)
             }
-            bytes32 node = keccak256(abi.encodePacked(rootNode,
-            // Set the resolver record
-            AddrResolver(resolver).setAddr(node, addr);
         }
     }
```

[G-06] Use assembly to hash values more efficiently

In the instances below, we can hash values more efficiently by using the least amount of opcodes possible via assembly.

Total Instances: 2

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSRegistrar.sol#L151

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 116 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301348	339283	320316	2

```
File: contracts/dnsregistrar/DNSRegistrar.sol
151:         bytes32 node = keccak256(abi.encodePacked(parentNode
```

```
diff --git a/contracts/dnsregistrar/DNSRegistrar.sol b/contracts
index 953a9a3..8fc7456 100644
--- a/contracts/dnsregistrar/DNSRegistrar.sol
+++ b/contracts/dnsregistrar/DNSRegistrar.sol
@@ -147,8 +147,13 @@ contract DNSRegistrar is IDNSRegistrar, IER

         // Make sure the parent name is enabled
         parentNode = enableNode(parentName);
-
-        bytes32 node = keccak256(abi.encodePacked(parentNode, l
+
+        bytes32 node;
+        assembly {
+            mstore(0x00, parentNode)
+            mstore(0x20, labelHash)
+            node := keccak256(0x00, 0x40)
+        }
         if (!RRUtils.serialNumberGte(inception, inceptions[node
             revert StaleProof();
         }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSRegistrar.sol#L185

Gas Savings for `DNSRegistrar.proveAndClaim` *, obtained via protocol's tests: Avg 97 gas*

	Min	Max	Avg	# calls
Before	198110	308834	269704	7
After	198025	308665	269607	7

```
File: contracts/dnsregistrar/DNSRegistrar.sol
185:         node = keccak256(abi.encodePacked(parentNode, label))
```

```
diff --git a/contracts/dnsregistrar/DNSRegistrar.sol b/contracts
index 953a9a3..8fc7456 100644
--- a/contracts/dnsregistrar/DNSRegistrar.sol
+++ b/contracts/dnsregistrar/DNSRegistrar.sol
@@ -182,7 +182,11 @@ contract DNSRegistrar is IDNSRegistrar, IER

         bytes32 parentNode = _enableNode(domain, offset + len +
         bytes32 label = domain.keccak(offset + 1, len);
-        node = keccak256(abi.encodePacked(parentNode, label));
+        assembly {
+            mstore(0x00, parentNode)
+            mstore(0x20, label)
+            node := keccak256(0x00, 0x40)
+        }
         address owner = ens.owner(node);
         if (owner == address(0) || owner == previousRegistrar)
             if (parentNode == bytes32(0)) {
```

[G-07] Use assembly to make more efficient back-to-back calls

In the instance below, two external calls, both of which take two function parameters, are performed. We can potentially avoid memory expansion costs by using assembly to utilize `scratch space` + `free memory pointer` memory offsets for the function calls. We will use the same memory space for both function calls.

Note: In order to do this optimization safely we will cache the free memory pointer value and restore it once we are done with our function calls.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSRegistrar.sol#L190-L191

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 380 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301084	339019	320052	2

```
File: contracts/dnsregistrar/DNSRegistrar.sol
190:                 root.setSubnodeOwner(label, address(this));
191:                 ens.setResolver(node, resolver);
```

```
diff --git a/contracts/dnsregistrar/DNSRegistrar.sol b/contracts
index 953a9a3..07f8d99 100644
--- a/contracts/dnsregistrar/DNSRegistrar.sol
+++ b/contracts/dnsregistrar/DNSRegistrar.sol
@@ -187,8 +187,26 @@ contract DNSRegistrar is IDNSRegistrar, IER
         if (owner == address(0) || owner == previousRegistrar)
             if (parentNode == bytes32(0)) {
                 Root root = Root(ens.owner(bytes32(0)));
-                root.setSubnodeOwner(label, address(this));
-                ens.setResolver(node, resolver);
+                address _resolver = resolver;
+                ENS _ens = ens;
+                assembly {
+                    let memptr := mload(0x40)
+                    mstore(0x00, 0x8cb8ecec)
+                    mstore(0x20, label)
+                    mstore(0x40, address())
+                    let success1 := call(gas(), root, 0x00, 0x1
+                    if iszero(success1) {
+                        revert(0, 0)
+                    }
+                    mstore(0x00, 0x1896f70a)
+                    mstore(0x20, node)
+                    mstore(0x40, _resolver)
+                    let success2 := call(gas(), _ens, 0x00, 0x1
+                    if iszero(success2) {
```

```
+                        revert(0, 0)
+                    }
+                    mstore(0x40, memptr)
+                }
            } else {
                ens.setSubnodeRecord(
                    parentNode,
```

[G-08] Use assembly for loops

We can use assembly to write a more gas efficient loop. See the final diffs for comments regarding the assembly code.

Total Instances: `3`

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/RRUtils.sol#L19-L31

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 9009 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	292455	330390	311423	2

```
File: contracts/dnssec-oracle/RRUtils.sol
19:     function nameLength(
20:         bytes memory self,
21:         uint256 offset
22:     ) internal pure returns (uint256) {
23:         uint256 idx = offset;
24:         while (true) {
25:             assert(idx < self.length);
26:             uint256 labelLen = self.readUint8(idx);
27:             idx += labelLen + 1;
28:             if (labelLen == 0) {
29:                 break;
30:             }
31:         }
```

```
diff --git a/contracts/dnssec-oracle/RRUtils.sol b/contracts/dns
index 20fbf15..2579945 100644
--- a/contracts/dnssec-oracle/RRUtils.sol
+++ b/contracts/dnssec-oracle/RRUtils.sol
@@ -21,12 +21,14 @@ library RRUtils {
         uint256 offset
     ) internal pure returns (uint256) {
         uint256 idx = offset;
-        while (true) {
-            assert(idx < self.length);
-            uint256 labelLen = self.readUint8(idx);
-            idx += labelLen + 1;
-            if (labelLen == 0) {
-                break;
+        assembly {
+            for {} 1 {} {
+                if iszero(lt(idx, mload(self))) {
+                    revert(0, 0)
+                }
+                let labelLen := shr(248, mload(add(add(self, 0x
+                idx := add(idx, add(labelLen, 1))
+                if iszero(labelLen) { break }
             }
         }
         return idx - offset;
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/RRUtils.sol#L55-L68

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 3216 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	298248	336183	317216	2

```
File: contracts/dnssec-oracle/RRUtils.sol
55:     function labelCount(
56:         bytes memory self,
57:         uint256 offset
```

```
58:     ) internal pure returns (uint256) {
59:         uint256 count = 0;
60:         while (true) {
61:             assert(offset < self.length);
62:             uint256 labelLen = self.readUint8(offset);
63:             offset += labelLen + 1;
64:             if (labelLen == 0) {
65:                 break;
66:             }
67:             count += 1;
68:         }
```

```
diff --git a/contracts/dnssec-oracle/RRUtils.sol b/contracts/dns
index 20fbf15..0cfe57a 100644
--- a/contracts/dnssec-oracle/RRUtils.sol
+++ b/contracts/dnssec-oracle/RRUtils.sol
@@ -57,14 +57,16 @@ library RRUtils {
         uint256 offset
     ) internal pure returns (uint256) {
         uint256 count = 0;
-        while (true) {
-            assert(offset < self.length);
-            uint256 labelLen = self.readUint8(offset);
-            offset += labelLen + 1;
-            if (labelLen == 0) {
-                break;
+        assembly {
+            for {} 1 {} {
+                if iszero(lt(offset, mload(self))) {
+                    revert(0, 0)
+                }
+                let labelLen := shr(248, mload(add(add(self, 0x
+                offset := add(offset, add(labelLen, 1))
+                if iszero(labelLen) { break }
+                count := add(count, 1)
             }
-            count += 1;
         }
         return count;
     }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/RRUtils.sol#L259-L270

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 894 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	300570	338505	319538	2

```
File: contracts/dnssec-oracle/RRUtils.sol
259:     function isSubdomainOf(
260:         bytes memory self,
261:         bytes memory other
262:     ) internal pure returns (bool) {
263:         uint256 off = 0;
264:         uint256 counts = labelCount(self, 0);
265:         uint256 othercounts = labelCount(other, 0);
266:
267:         while (counts > othercounts) {
268:             off = progress(self, off);
269:             counts--;
270:         }
```

```
diff --git a/contracts/dnssec-oracle/RRUtils.sol b/contracts/dns
index 20fbf15..8f098f0 100644
--- a/contracts/dnssec-oracle/RRUtils.sol
+++ b/contracts/dnssec-oracle/RRUtils.sol
@@ -263,10 +263,18 @@ library RRUtils {
         uint256 off = 0;
         uint256 counts = labelCount(self, 0);
         uint256 othercounts = labelCount(other, 0);
-
-        while (counts > othercounts) {
-            off = progress(self, off);
-            counts--;
+
+        assembly {
+            for {} 1 {} {
+                if iszero(gt(counts, othercounts)) {
+                    break
+                }
+                if iszero(lt(off, mload(self))) {
+                    revert(0, 0)
```

```
+                 }
+                 off := add(add(off, 1), shr(248, mload(add(add(
+                 counts := sub(counts, 1)
+             }
          }

          return self.equals(off, other, 0);
```

[G-09] Use assembly to grab and cast value in byte array

Like various similar functions, i.e. readUint16 and readUint32, assembly can be used to grab the value at a specified index of a byte array and cast that value to a specific uint type. In the diff below, a check is done before this operation to ensure that the offset is not out of bounds.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/BytesUtils.sol#L179-L184

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 1280 gas*

	Min	Max	Avg	# calls	
Before	301464	339399	320432	2	
After	300184	338119	319152	2	

```
File: contracts/dnssec-oracle/BytesUtils.sol
179:     function readUint8(
180:         bytes memory self,
181:         uint256 idx
182:     ) internal pure returns (uint8 ret) {
183:         return uint8(self[idx]);
184:     }
```

```
diff --git a/contracts/dnssec-oracle/BytesUtils.sol b/contracts/
index 96344ce..8b6335d 100644
--- a/contracts/dnssec-oracle/BytesUtils.sol
+++ b/contracts/dnssec-oracle/BytesUtils.sol
@@ -180,7 +180,12 @@ library BytesUtils {
```

```
        bytes memory self,
        uint256 idx
    ) internal pure returns (uint8 ret) {
-       return uint8(self[idx]);
+       assembly {
+           if iszero(lt(idx, mload(self))) {
+               revert(0, 0)
+           }
+           ret := shr(248, mload(add(add(self, 0x20), idx)))
+       }
    }
```

[G-10] Include check in assembly block

In the instances below, we can include the checks in the `require` statements inside the following assembly blocks.

Total Instances: `3`

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/BytesUtils.sol#L13-L22

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 1656 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	299808	337743	318776	2

```
File: contracts/dnssec-oracle/BytesUtils.sol
13:     function keccak(
14:         bytes memory self,
15:         uint256 offset,
16:         uint256 len
17:     ) internal pure returns (bytes32 ret) {
18:         require(offset + len <= self.length);
19:         assembly {
20:             ret := keccak256(add(add(self, 32), offset), len)
21:         }
```

```
22:     }
```

```
diff --git a/contracts/dnssec-oracle/BytesUtils.sol b/contracts/
index 96344ce..472c224 100644
--- a/contracts/dnssec-oracle/BytesUtils.sol
+++ b/contracts/dnssec-oracle/BytesUtils.sol
@@ -15,8 +15,10 @@ library BytesUtils {
         uint256 offset,
         uint256 len
     ) internal pure returns (bytes32 ret) {
-        require(offset + len <= self.length);
         assembly {
+            if gt(add(offset, len), mload(self)) {
+                revert(0, 0)
+            }
             ret := keccak256(add(add(self, 32), offset), len)
         }
     }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/BytesUtils.sol#L192-L200

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 3795 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	297669	335604	316637	2

```
File: contracts/dnssec-oracle/BytesUtils.sol
192:     function readUint16(
193:         bytes memory self,
194:         uint256 idx
195:     ) internal pure returns (uint16 ret) {
196:         require(idx + 2 <= self.length);
197:         assembly {
198:             ret := and(mload(add(add(self, 2), idx)), 0xFFFF
199:         }
200:     }
```

```
diff --git a/contracts/dnssec-oracle/BytesUtils.sol b/contracts/
index 96344ce..cd94ece 100644
--- a/contracts/dnssec-oracle/BytesUtils.sol
+++ b/contracts/dnssec-oracle/BytesUtils.sol
@@ -193,8 +193,10 @@ library BytesUtils {
         bytes memory self,
         uint256 idx
     ) internal pure returns (uint16 ret) {
-        require(idx + 2 <= self.length);
         assembly {
+            if gt(add(idx, 2), mload(self)) {
+                revert(0, 0)
+            }
             ret := and(mload(add(add(self, 2), idx)), 0xFFFF)
         }
     }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/BytesUtils.sol#L208-L216

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 1449 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	300015	337950	318983	2

```
File: contracts/dnssec-oracle/BytesUtils.sol
208:     function readUint32(
209:         bytes memory self,
210:         uint256 idx
211:     ) internal pure returns (uint32 ret) {
212:         require(idx + 4 <= self.length);
213:         assembly {
214:             ret := and(mload(add(add(self, 4), idx)), 0xFFFF
215:         }
216:     }
```

```
diff --git a/contracts/dnssec-oracle/BytesUtils.sol b/contracts/
```

```
index 96344ce..e5d3190 100644
--- a/contracts/dnssec-oracle/BytesUtils.sol
+++ b/contracts/dnssec-oracle/BytesUtils.sol
@@ -209,8 +209,10 @@ library BytesUtils {
         bytes memory self,
         uint256 idx
     ) internal pure returns (uint32 ret) {
-        require(idx + 4 <= self.length);
         assembly {
+            if gt(add(idx, 4), mload(self)) {
+                revert(0, 0)
+            }
             ret := and(mload(add(add(self, 4), idx)), 0xFFFFFFF
         }
     }
```

This optimization can also be done for the instances below. However, they are not included in the final Diffs as they do not result in gas savings when the tests are run:

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/BytesUtils.sol#L224-L232

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/BytesUtils.sol#L240-L251

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/BytesUtils.sol#L260-L271

[G-11] Write a more gas efficient assembly loop

In the instance below, we can rewrite the assembly loop using a more gas efficient infinite loop, which performs the condition check at the end of the loop. See this for more information.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/utils/HexUtils.sol#L44-L58

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 306 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301158	339093	320126	2

```
File: contracts/utils/HexUtils.sol
44:             for {
45:                 let i := idx
46:             } lt(i, lastIdx) {
47:                 i := add(i, 2)
48:             } {
49:                 let byte1 := getHex(byte(0, mload(add(ptr, i)
50:                 let byte2 := getHex(byte(0, mload(add(ptr, ac
51:                 // if either byte is invalid, set invalid and
52:                 if or(eq(byte1, 0xff), eq(byte2, 0xff)) {
53:                     valid := false
54:                     break
55:                 }
56:                 let combined := or(shl(4, byte1), byte2)
57:                 r := or(shl(8, r), combined)
58:             }
```

```
diff --git a/contracts/utils/HexUtils.sol b/contracts/utils/HexU
index 3508390..b8579b7 100644
--- a/contracts/utils/HexUtils.sol
+++ b/contracts/utils/HexUtils.sol
@@ -41,11 +41,8 @@ library HexUtils {
             }

             let ptr := add(str, 32)
-            for {
-                let i := idx
-            } lt(i, lastIdx) {
-                i := add(i, 2)
-            } {
+            let i := idx
+            for {} 1 {} {
                 let byte1 := getHex(byte(0, mload(add(ptr, i))))
                 let byte2 := getHex(byte(0, mload(add(ptr, add(
                 // if either byte is invalid, set invalid and b
@@ -55,6 +52,8 @@ library HexUtils {
                 }
                 let combined := or(shl(4, byte1), byte2)
```

```
                    r := or(shl(8, r), combined)
+                   i := add(i, 2)
+                   if iszero(lt(i, lastIdx)) { break }
                }
            }
        }
```

[G-12] Use a `do while` loop instead of a `for` loop

A `do while` loop will cost less gas since the condition is not being checked for the first iteration.

Note: Other optimizations, such as caching length, precrementing, and using unchecked blocks are not included in the Diff since they are included in the automated findings report.

Total Instances: `8`

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L118-L126

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 60 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301404	339339	320372	2

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
118:         for (uint256 i = 0; i < input.length; i++) {
119:             RRUtils.SignedSet memory rrset = validateSigned
120:                 input[i],
121:                 proof,
122:                 now
123:             );
124:             proof = rrset.data;
125:             inception = rrset.inception;
126:         }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..c30d8ba 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -115,7 +115,8 @@ contract DNSSECImpl is DNSSEC, Owned {
         returns (bytes memory rrs, uint32 inception)
     {
         bytes memory proof = anchors;
-        for (uint256 i = 0; i < input.length; i++) {
+        uint256 i;
+        do {
             RRUtils.SignedSet memory rrset = validateSignedSet(
                 input[i],
                 proof,
@@ -123,7 +124,8 @@ contract DNSSECImpl is DNSSEC, Owned {
             );
             proof = rrset.data;
             inception = rrset.inception;
-        }
+            i++;
+        } while (i < input.length);
         return (proof, inception);
     }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L260-L274

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 62 gas*

	Min	Max	Avg	# calls	
Before	301464	339399	320432	2	
After	301402	339337	320370	2	

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
260:         for (; !proof.done(); proof.next()) {
261:             bytes memory proofName = proof.name();
262:             if (!proofName.equals(rrset.signerName)) {
263:                 revert ProofNameMismatch(rrset.signerName, p
264:             }
```

```
265:
266:             bytes memory keyrdata = proof.rdata();
267:             RRUtils.DNSKEY memory dnskey = keyrdata.readDNSK
268:                 0,
269:                 keyrdata.length
270:             );
271:             if (verifySignatureWithKey(dnskey, keyrdata, rrs
272:                 return;
273:             }
274:         }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..a357e70 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -257,7 +257,7 @@ contract DNSSECImpl is DNSSEC, Owned {
         RRUtils.RRIterator memory proof
     ) internal view {
         // Check the DNSKEY's owner name matches the signer nam
-        for (; !proof.done(); proof.next()) {
+        do {
             bytes memory proofName = proof.name();
             if (!proofName.equals(rrset.signerName)) {
                 revert ProofNameMismatch(rrset.signerName, proo
@@ -271,7 +271,9 @@ contract DNSSECImpl is DNSSEC, Owned {
             if (verifySignatureWithKey(dnskey, keyrdata, rrset,
                 return;
             }
-        }
+            proof.next();
+        } while (!proof.done());
+
         revert NoMatchingProof(rrset.signerName);
     }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L181-L213

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 140 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301324	339259	320292	2

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
181:     function validateRRs(
182:         RRUtils.SignedSet memory rrset,
183:         uint16 typecovered
184:     ) internal pure returns (bytes memory name) {
185:         // Iterate over all the RRs
186:         for (
187:             RRUtils.RRIterator memory iter = rrset.rrs();
188:             !iter.done();
189:             iter.next()
190:         ) {
191:             // We only support class IN (Internet)
192:             if (iter.class != DNSCLASS_IN) {
193:                 revert InvalidClass(iter.class);
194:             }
195:
196:             if (name.length == 0) {
197:                 name = iter.name();
198:             } else {
199:                 // Name must be the same on all RRs. We do t
200:                 // repeatedly.
201:                 if (
202:                     name.length != iter.data.nameLength(iter
203:                     !name.equals(0, iter.data, iter.offset,
204:                 ) {
205:                     revert InvalidRRSet();
206:                 }
207:             }
208:
209:             // o  The RRSIG RR's Type Covered field MUST equ
210:             if (iter.dnstype != typecovered) {
211:                 revert SignatureTypeMismatch(iter.dnstype, t
212:             }
213:         }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..f09438a 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
```

```
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -183,11 +183,8 @@ contract DNSSECImpl is DNSSEC, Owned {
         uint16 typecovered
     ) internal pure returns (bytes memory name) {
         // Iterate over all the RRs
-        for (
-            RRUtils.RRIterator memory iter = rrset.rrs();
-            !iter.done();
-            iter.next()
-        ) {
+        RRUtils.RRIterator memory iter = rrset.rrs();
+        do {
             // We only support class IN (Internet)
             if (iter.class != DNSCLASS_IN) {
                 revert InvalidClass(iter.class);
@@ -210,7 +207,8 @@ contract DNSSECImpl is DNSSEC, Owned {
             if (iter.dnstype != typecovered) {
                 revert SignatureTypeMismatch(iter.dnstype, type
             }
-        }
+            iter.next();
+        } while (!iter.done());
     }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L330-L361

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 68 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301396	339331	320364	2

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
330:     function verifyWithDS(
331:         RRUtils.SignedSet memory rrset,
332:         RRSetWithSignature memory data,
333:         RRUtils.RRIterator memory proof
334:     ) internal view {
335:         uint256 proofOffset = proof.offset;
```

```
336:         for (
337:             RRUtils.RRIterator memory iter = rrset.rrs();
338:             !iter.done();
339:             iter.next()
340:         ) {
341:             if (iter.dnstype != DNSTYPE_DNSKEY) {
342:                 revert InvalidProofType(iter.dnstype);
343:             }
344:
345:             bytes memory keyrdata = iter.rdata();
346:             RRUtils.DNSKEY memory dnskey = keyrdata.readDNSK
347:                 0,
348:                 keyrdata.length
349:             );
350:             if (verifySignatureWithKey(dnskey, keyrdata, rrs
351:                 // It's self-signed - look for a DS record t
352:                 if (
353:                     verifyKeyWithDS(rrset.signerName, proof,
354:                 ) {
355:                     return;
356:                 }
357:                 // Rewind proof iterator to the start for th
358:                 proof.nextOffset = proofOffset;
359:                 proof.next();
360:             }
361:         }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..66cc74e 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -333,11 +333,8 @@ contract DNSSECImpl is DNSSEC, Owned {
         RRUtils.RRIterator memory proof
     ) internal view {
         uint256 proofOffset = proof.offset;
-        for (
-            RRUtils.RRIterator memory iter = rrset.rrs();
-            !iter.done();
-            iter.next()
-        ) {
+        RRUtils.RRIterator memory iter = rrset.rrs();
+        do {
             if (iter.dnstype != DNSTYPE_DNSKEY) {
                 revert InvalidProofType(iter.dnstype);
```

```
                }
@@ -358,7 +355,8 @@ contract DNSSECImpl is DNSSEC, Owned {
                    proof.nextOffset = proofOffset;
                    proof.next();
                }
-           }
+               iter.next();
+           } while (!iter.done());
            revert NoMatchingProof(rrset.signerName);
        }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L373-L404

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 68 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301396	339331	320364	2

```
File: contracts/dnsssec-oracle/DNSSECImpl.sol
373:     function verifyKeyWithDS(
374:         bytes memory keyname,
375:         RRUtils.RRIterator memory dsrrs,
376:         RRUtils.DNSKEY memory dnskey,
377:         bytes memory keyrdata
378:     ) internal view returns (bool) {
379:         uint16 keytag = keyrdata.computeKeytag();
380:         for (; !dsrrs.done(); dsrrs.next()) {
381:             bytes memory proofName = dsrrs.name();
382:             if (!proofName.equals(keyname)) {
383:                 revert ProofNameMismatch(keyname, proofName)
384:             }
385:
386:             RRUtils.DS memory ds = dsrrs.data.readDS(
387:                 dsrrs.rdataOffset,
388:                 dsrrs.nextOffset - dsrrs.rdataOffset
389:             );
390:             if (ds.keytag != keytag) {
391:                 continue;
```

```
392:             }
393:             if (ds.algorithm != dnskey.algorithm) {
394:                 continue;
395:             }
396:
397:             Buffer.buffer memory buf;
398:             buf.init(keyname.length + keyrdata.length);
399:             buf.append(keyname);
400:             buf.append(keyrdata);
401:             if (verifyDSHash(ds.digestType, buf.buf, ds.dige
402:                 return true;
403:             }
404:         }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..ed1c137 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -377,7 +377,7 @@ contract DNSSECImpl is DNSSEC, Owned {
         bytes memory keyrdata
     ) internal view returns (bool) {
         uint16 keytag = keyrdata.computeKeytag();
-        for (; !dsrrs.done(); dsrrs.next()) {
+        do {
             bytes memory proofName = dsrrs.name();
             if (!proofName.equals(keyname)) {
                 revert ProofNameMismatch(keyname, proofName);
@@ -388,9 +388,11 @@ contract DNSSECImpl is DNSSEC, Owned {
                 dsrrs.nextOffset - dsrrs.rdataOffset
             );
             if (ds.keytag != keytag) {
+                dsrrs.next();
                 continue;
             }
             if (ds.algorithm != dnskey.algorithm) {
+                dsrrs.next();
                 continue;
             }

@@ -401,7 +403,8 @@ contract DNSSECImpl is DNSSEC, Owned {
             if (verifyDSHash(ds.digestType, buf.buf, ds.digest)
                 return true;
             }
-        }
+        } while (dsrrs.next());
+
         return false;
     }
```

```
+             dsrrs.next();
+         } while (!dsrrs.done());
          return false;
      }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSClaimChecker.sol#L29-L40

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 42 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301422	339357	320390	2

```
File: contracts/dnsregistrar/DNSClaimChecker.sol
29:         for (
30:             RRUtils.RRIterator memory iter = data.iterateRRs(
31:             !iter.done();
32:             iter.next()
33:         ) {
34:             if (iter.name().compareNames(buf.buf) != 0) conti
35:             bool found;
36:             address addr;
37:             (addr, found) = parseRR(data, iter.rdataOffset, i
38:             if (found) {
39:                 return (addr, true);
40:             }
```

```
diff --git a/contracts/dnsregistrar/DNSClaimChecker.sol b/contra
index 54950d1..7848671 100644
--- a/contracts/dnsregistrar/DNSClaimChecker.sol
+++ b/contracts/dnsregistrar/DNSClaimChecker.sol
@@ -26,19 +26,20 @@ library DNSClaimChecker {
          buf.append("\x04_ens");
          buf.append(name);

-         for (
-             RRUtils.RRIterator memory iter = data.iterateRRs(0)
```

```
-             !iter.done();
-             iter.next()
-         ) {
-             if (iter.name().compareNames(buf.buf) != 0) continu
+         RRUtils.RRIterator memory iter = data.iterateRRs(0);
+         do {
+             if (iter.name().compareNames(buf.buf) != 0) {
+                 iter.next();
+                 continue;
+             }
              bool found;
              address addr;
              (addr, found) = parseRR(data, iter.rdataOffset, ite
              if (found) {
                  return (addr, true);
              }
-         }
+             iter.next();
+         } while (!iter.done());

          return (address(0x0), false);
      }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSClaimChecker.sol#L46-L61

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 36 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301428	339363	320396	2

```
File: contracts/dnsregistrar/DNSClaimChecker.sol
46:     function parseRR(
47:         bytes memory rdata,
48:         uint256 idx,
49:         uint256 endIdx
50:     ) internal pure returns (address, bool) {
51:         while (idx < endIdx) {
52:             uint256 len = rdata.readUint8(idx);
```

```
53:             idx += 1;
54:
55:             bool found;
56:             address addr;
57:             (addr, found) = parseString(rdata, idx, len);
58:
59:             if (found) return (addr, true);
60:             idx += len;
61:         }
```

```
diff --git a/contracts/dnsregistrar/DNSClaimChecker.sol b/contra
index 54950d1..55bb5d2 100644
--- a/contracts/dnsregistrar/DNSClaimChecker.sol
+++ b/contracts/dnsregistrar/DNSClaimChecker.sol
@@ -48,9 +48,9 @@ library DNSClaimChecker {
         uint256 idx,
         uint256 endIdx
     ) internal pure returns (address, bool) {
-        while (idx < endIdx) {
+        do {
             uint256 len = rdata.readUint8(idx);
-            idx += 1;
+            ++idx;

             bool found;
             address addr;
@@ -58,7 +58,7 @@ library DNSClaimChecker {

             if (found) return (addr, true);
             idx += len;
-        }
+        } while (idx < endIdx);

         return (address(0x0), false);
     }
```

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/RRUtils.sol#L384-L399

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 406 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301072	339007	320026	2

```
File: contracts/dnssec-oracle/RRUtils.sol
384:             for (uint256 i = 0; i < data.length + 31; i += 3
385:                 uint256 word;
386:                 assembly {
387:                     word := mload(add(add(data, 32), i))
388:                 }
389:                 if (i + 32 > data.length) {
390:                     uint256 unused = 256 - (data.length - i)
391:                     word = (word >> unused) << unused;
392:                 }
393:                 ac1 +=
394:                     (word &
395:                         0xFF00FF00FF00FF00FF00FF00FF00FF00FF00FF
396:                     8;
397:                 ac2 += (word &
398:                     0x00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF0
399:             }
```

```
diff --git a/contracts/dnssec-oracle/RRUtils.sol b/contracts/dns
index 20fbf15..e9309d6 100644
--- a/contracts/dnssec-oracle/RRUtils.sol
+++ b/contracts/dnssec-oracle/RRUtils.sol
@@ -381,13 +381,15 @@ library RRUtils {
             require(data.length <= 8192, "Long keys not permitt
             uint256 ac1;
             uint256 ac2;
-            for (uint256 i = 0; i < data.length + 31; i += 32)
+            uint256 length = data.length;
+            uint256 i;
+            do {
                 uint256 word;
                 assembly {
                     word := mload(add(add(data, 32), i))
                 }
-                if (i + 32 > data.length) {
-                    uint256 unused = 256 - (data.length - i) *
+                if (i + 32 > length) {
+                    uint256 unused = 256 - (length - i) * 8;
```

```
                    word = (word >> unused) << unused;
                }
                ac1 +=
@@ -396,7 +398,8 @@ library RRUtils {
                    8;
                ac2 += (word &
                    0x00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF00FF
-           }
+               i += 32;
+           } while (i < data.length + 31);
            ac1 =
                (ac1 &
                    0x0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000
```

[G-13] Don't cache value if it is only used once

If a value is only intended to be used once then it should not be cached. Caching the value will result in unnecessary stack manipulation.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L304-L307

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 90 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301374	339309	320342	2

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
304:         uint16 computedkeytag = keyrdata.computeKeytag();
305:         if (computedkeytag != rrset.keytag) {
306:             return false;
307:         }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..c6c03fc 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
```

```
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -301,8 +301,7 @@ contract DNSSECImpl is DNSSEC, Owned {
         if (dnskey.algorithm != rrset.algorithm) {
             return false;
         }
-        uint16 computedkeytag = keyrdata.computeKeytag();
-        if (computedkeytag != rrset.keytag) {
+        if (keyrdata.computeKeytag() != rrset.keytag) {
             return false;
         }
```

[G-14] Refactor code to avoid instantiating memory struct within loop

In the instance below, instead of instantiating the `SignedSets` struct within memory in the loop we can refactor the `validateSignedSet` function so that only the needed struct values are returned and used in the loop.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L118-L174

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver` *, obtained via protocol's tests: Avg 322 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301142	339077	320110	2

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
118:         for (uint256 i = 0; i < input.length; i++) {
119:             RRUtils.SignedSet memory rrset = validateSigned
120:                 input[i],
121:                 proof,
122:                 now
123:             );
124:             proof = rrset.data;
125:             inception = rrset.inception;
126:         }
```

```
140:     function validateSignedSet(
141:         RRSetWithSignature memory input,
142:         bytes memory proof,
143:         uint256 now
144:     ) internal view returns (RRUtils.SignedSet memory rrset)
145:         rrset = input.rrset.readSignedSet();

173:         return rrset;
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..a6a184c 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -116,13 +116,13 @@ contract DNSSECImpl is DNSSEC, Owned {
     {
         bytes memory proof = anchors;
         for (uint256 i = 0; i < input.length; i++) {
-            RRUtils.SignedSet memory rrset = validateSignedSet(
+            (bytes memory rrsetData, uint32 rrsetInception) = v
                 input[i],
                 proof,
                 now
             );
-            proof = rrset.data;
-            inception = rrset.inception;
+            proof = rrsetData;
+            inception = rrsetInception;
         }
         return (proof, inception);
     }
@@ -141,8 +141,8 @@ contract DNSSECImpl is DNSSEC, Owned {
         RRSetWithSignature memory input,
         bytes memory proof,
         uint256 now
-    ) internal view returns (RRUtils.SignedSet memory rrset) {
-        rrset = input.rrset.readSignedSet();
+    ) internal view returns (bytes memory, uint32) {
+        RRUtils.SignedSet memory rrset = input.rrset.readSignec

         // Do some basic checks on the RRs and extract the name
         bytes memory name = validateRRs(rrset, rrset.typeCovere
@@ -170,7 +170,7 @@ contract DNSSECImpl is DNSSEC, Owned {
         // Validate the signature
         verifySignature(name, rrset, input, proof);
```

```
-         return rrset;
+         return (rrset.data, rrset.inception);
      }
```

[G-15] `If` statements that use `&` can be refactored into nested `if` statements

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnssec-oracle/DNSSECImpl.sol#L312-L314

Gas Savings for `DNSRegistrar.proveAndClaimWithResolver`*, obtained via protocol's tests: Avg 64 gas*

	Min	Max	Avg	# calls
Before	301464	339399	320432	2
After	301400	339335	320368	2

```
File: contracts/dnssec-oracle/DNSSECImpl.sol
312:         if (dnskey.flags & DNSKEY_FLAG_ZONEKEY == 0) {
313:             return false;
314:         }
```

```
diff --git a/contracts/dnssec-oracle/DNSSECImpl.sol b/contracts/
index a3e4e5f..e442944 100644
--- a/contracts/dnssec-oracle/DNSSECImpl.sol
+++ b/contracts/dnssec-oracle/DNSSECImpl.sol
@@ -309,8 +309,10 @@ contract DNSSECImpl is DNSSEC, Owned {
         // o The matching DNSKEY RR MUST be present in the zone
         //   RRset, and MUST have the Zone Flag bit (DNSKEY RDA
         //   set.
-        if (dnskey.flags & DNSKEY_FLAG_ZONEKEY == 0) {
-            return false;
+        if (dnskey.flags == 0) {
+            if (DNSKEY_FLAG_ZONEKEY == 0) {
+                return false;
+            }
         }
```

[G-16] Refactor modifier to avoid two external calls when calling `setPublicSuffixList`

The `onlyOwner` modifier performs two external calls. In order to avoid these two calls everytime `setPublicSufficList` is called, we can perform these two calls in the constructor and create immutable variables for the return values.

https://github.com/code-423n4/2023-04-ens/blob/main/contracts/dnsregistrar/DNSRegistrar.sol#L73-L78

```
File: contracts/dnsregistrar/DNSRegistrar.sol
73:     modifier onlyOwner() {
74:         Root root = Root(ens.owner(bytes32(0)));
75:         address owner = root.owner();
76:         require(msg.sender == owner);
77:         _;
78:     }
```

```
diff --git a/contracts/dnsregistrar/DNSRegistrar.sol b/contracts
index 953a9a3..3779292 100644
--- a/contracts/dnsregistrar/DNSRegistrar.sol
+++ b/contracts/dnsregistrar/DNSRegistrar.sol
@@ -28,6 +28,7 @@ contract DNSRegistrar is IDNSRegistrar, IERC16
      PublicSuffixList public suffixes;
      address public immutable previousRegistrar;
      address public immutable resolver;
+     address private immutable rootOwner;
      // A mapping of the most recent signatures seen for each cl
      mapping(bytes32 => uint32) public inceptions;

@@ -65,15 +66,14 @@ contract DNSRegistrar is IDNSRegistrar, IERC
          suffixes = _suffixes;
          emit NewPublicSuffixList(address(suffixes));
          ens = _ens;
+         rootOwner = Root(_ens.owner(bytes32(0))).owner();
      }

      /**
       * @dev This contract's owner-only functions can be invoked
       */
      modifier onlyOwner() {
```

```
-         Root root = Root(ens.owner(bytes32(0)));
-         address owner = root.owner();
-         require(msg.sender == owner);
+         require(msg.sender == rootOwner);
          _;
      }
```

`GasReport` output, with all optimizations applied

```
·-------------------------------------------------------|------
|                  Solc version: 0.8.17                 ·  Opti
······················|·····························|·····
|  Methods
·····················|·····························|·····
|  Contract          ·  Method                     ·  Min
·····················|·····························|·····
|  DNSRegistrar      ·  proveAndClaim              ·     1
·····················|·····························|·····
|  DNSRegistrar      ·  proveAndClaimWithResolver  ·     2
·····················|·····························|·····
|  DNSSECImpl        ·  setAlgorithm               ·
·····················|·····························|·····
|  DNSSECImpl        ·  setDigest                  ·
·····················|·····························|·····
|  ENSRegistry       ·  setOwner                   ·
·····················|·····························|·····
|  ENSRegistry       ·  setResolver                ·
·····················|·····························|·····
|  ENSRegistry       ·  setSubnodeOwner            ·
·····················|·····························|·····
|  ERC20Recoverable  ·  recoverFunds               ·
·····················|·····························|·····
|  MockERC20         ·  transfer                   ·
·····················|·····························|·····
|  OwnedResolver     ·  setAddr                    ·
·····················|·····························|·····
|  PublicResolver    ·  setAddr                    ·
·····················|·····························|·····
|  PublicResolver    ·  setApprovalForAll          ·
·····················|·····························|·····
|  PublicResolver    ·  setName                    ·
·····················|·····························|·····
|  PublicResolver    ·  setText                    ·
```

ReverseRegistrar	claim
Root	setController
Root	setSubnodeOwner
SimplePublicSuffixList	addPublicSuffixes

Arachnid (ENS) acknowledged

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top